



May 2001
Revision 0.4
Confidential

XpressROM Firmware Geode™ Technology Software

General Description

XpressROM is the firmware offering from the Information Appliance Group of National Semiconductor®. The XpressROM firmware is a set of components designed to initialize the devices, provide certain interrupt services, and emulate traditional hardware functionality. XpressROM is targeted for platform designs using a processor and I/O companion device from the Geode™ family of products (e.g., GX1 processor and CS5530A I/O companion device as well as the Geode™ SC1200 Single Chip processor family).

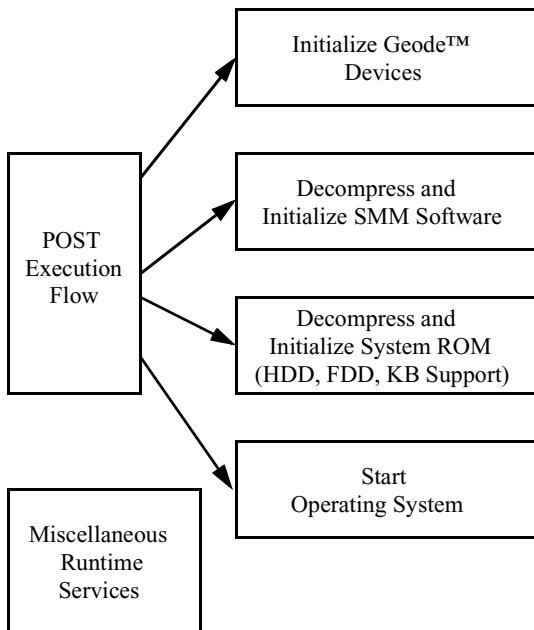
The two diagrams below illustrate the general functionality of the XpressROM firmware.

The functional layout shows the operational components of the XpressROM Power-On Self Test (POST), as well as other services. During POST, the Geode processor and companion chip are initialized along with other devices, for example a SuperI/O. Installed SDRAM is identified, configured, and optimized. The XpressROM System Management Mode (SMM) software from

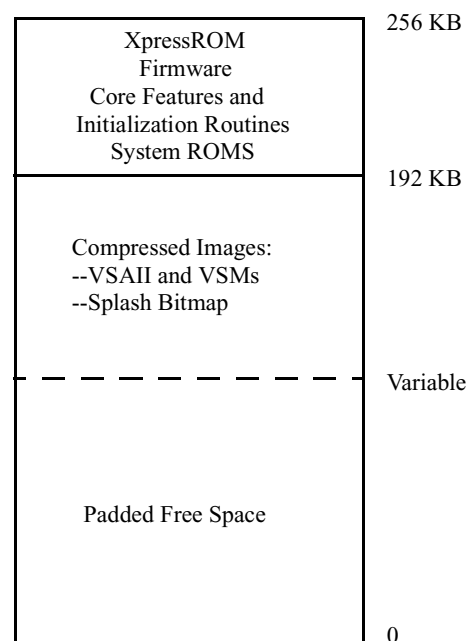
this point on referred to as VSAIL, including any emulated hardware, is decompressed and initialized. Other special functionality is supported by including system ROMs. There may also be adapter ROMs present in the system or ROM binaries embedded in the XpressROM image. These ROMs are decompressed, shadowed and initialized. Finally an operating system is started. The XpressROM firmware also contains some miscellaneous runtime support services.

The XpressROM ROM image is made up of POST code, initialization routines, and other features. The initialization code and various runtime services are found in the upper 64 KB of the XpressROM image. Other features, for example, option ROMs, SMM software, or a splash screen bitmap are compressed. The compressed binaries are prepadded to a 192 KB image. The initialization and support code is concatenated to the 192 KB image.

XpressROM Functional Layout



XpressROM Flash Image



Features

XpressROM is provided for support of Internet Appliance

Each Platform XpressROM provides the following initialization and run-time API (application programming interface) support:

- n Legacy Power Management
 - XpressROM firmware supports operating system independent power management. Each platform device is power managed based upon that device's activity as well as system activity.
- n APM Real Mode/Protected Mode API's
 - The XpressROM firmware INT 15h API supports the APM 1.2 (Advanced Power Management) functions listed in the "Advanced Power Management Specification" document available from Microsoft, Intel, and other sources.
- n Boot Menu
 - The system designer may optionally install a menu that can be activated at POST. This allows the user to select from a list of devices or embedded option ROMs from which the bootstrap is attempted. This can be especially useful for systems requiring a recovery mechanism, or may be used for demonstrating various bootstrap options.
- n Build Environment is Embedded in ROM Image
 - The XpressROM flash image contains an encoded list of all the user-defined options. Additionally, any overridden filename and version number of the file it was based on is also embedded into the ROM image. This provides a means to determine exactly how the image was created and makes the debugging process easier.
- n Enhanced/Large IDE Detection and Configuration
 - Contains a power-on sequence of firmware that identifies and configures ATA devices, such as IDE drives and compact flash devices. During a power-on or reset sequence, the XpressROM firmware scans the primary and secondary IDE channels for ATA compliant devices. For each ATA device that is present, the XpressROM device performs the ATA "Identify" command to determine the drive's geometry, capabilities, and vendor-specific information. In cases where a drive's geometry exceeds the capabilities of the standard INT 13h interface, the XpressROM firmware performs CHS (Cylinder/Head/Sector) translation to enable the legacy operating system to utilize the full capacity of the device.
- n INT 11h (Equipment List)
 - Provides a traditional INT 11h service for operating systems that determines the number and type of installed PC/AT style peripherals.
- n INT 12h (Get Memory Size)
 - This XpressROM firmware interrupt service is provided to report to drivers, applications, and operating systems the amount of base (< 640 KB) memory that is available in the system. Typically, this service returns the value 639 (KB) which represents the entire low ten segments of real mode RAM minus 1 KB which is allocated to an EBDA (Extended BIOS Data Area).
- n INT 13h (1.44 MB Floppy Disk Support)
 - Provides INT 13h support for a 3.5"/1.44 MB floppy disk drive. The XpressROM firmware INT 13h floppy disk services enable the user to read, write, format, verify, and boot from a 1.44 MB floppy diskette. Additionally, the XpressROM firmware provides Disk Change Line support for the floppy disk drive.
- n INT 13h (Fixed Disk Services w/IBM Extensions)
 - In addition to the standard IBM's PC/AT-style INT 13h interface, the XpressROM firmware implements the fixed-disk subset of those INT 13h extensions defined by the IBM document, "INT 13h Extensions API." The inclusion of these extensions enables the XpressROM firmware to, in conjunction with certain operating systems, support drives whose capacity exceeds 8.4 GB. Additionally, the XpressROM firmware supports LBA (Logical Block Addressing) for devices larger than 8.4 GB provided that the installed operation contains support to invoke the INT 13h extensions.
- n INT 14h (Serial Port BIOS Support)
 - Supports the standard INT 14h interface for initializing, reading, and writing data from the serial port as well as checking the serial port status.
- n INT 15h Functions
 - Function 24h (A20 Gate Support)
 - Contains functions for enabling and disabling the A20 Gate support as well as querying the status and the support for the A20 Gate.
 - Function 87h (Extended Memory Block Move)
 - Provides this traditional, 80x286 PC/AT-style service to applications, drivers, and operating systems that move blocks of memory to, from, or within extended RAM while executing in real mode.
 - Function 88h (Get Extended Memory Size)
 - Provides this traditional, PC/AT service as a means of reporting the amount of installed, extended system DRAM (up to 64 MB) to the operating system.
 - Function 89h (Enter Protected Mode)
 - Switches the system into Protected Mode with support for 286 and higher technology.
 - Function BEh (I²C Access Method)
 - XpressROM contains National-specific BIOS calls for accessing devices on an I²C bus. A caller provides an I²C address and can read or write the device.
 - XpressROM contains National-specific BIOS calls for accessing data in NVRAM. The caller is not required to know details about the location of this data, but may use the predefined tokens to read or write data in NVRAM.
 - Function C0h (Get ROM Configuration)
 - Returns a pointer to the ROM configuration tables which contain system information such as model, sub-model, and other features.
 - Function C1h (Get EBDA)
 - Returns the segment address of the Extended BIOS Data Area.
 - Function C2h (PS/2 Mouse Support)

Features (Continued)

- There is optional support for a standard PS/2 mouse in the XpressROM firmware. This is comprised of code to initialize the mouse, the INT 15h function C2h calls, and the INT 74h code to handle IRQ 12.
- Function E8h (Memory Size and Memory Map)
 - XpressROM contains support for the E8h functions for determining the system memory size (function E801h) and for building a system memory map (function E820h).
- Function 4358h (VSA Init Callbacks)
 - For backwards compatibility, the XpressROM firmware provides VSA™ (Virtual System Architecture®) software INT 15h callbacks. For a description of the INT 15h callback mechanism and the subfunctions that it supports, consult the “VSA/BIOS Porting Guide”, available from National Semiconductor.
- n INT 16h (IRQ1 Keyboard Services)
 - Provides INT 16h keyboard services via a small IBM-style option ROM. Additionally, the XpressROM firmware includes an IRQ1 handler to process incoming scan codes and manage the circular scan code buffer that resides in the BIOS data area. To support applications and drivers that intercept scan codes, the IRQ1 handler forwards all scan codes to the INT 15h Function 4Fh scan code redirect. The keyboard code also supports calling INT 1Bh for Ctrl-Break functionality.
- n INT 17h (Parallel Port BIOS Support)
 - Supports the standard INT 17h interface for initializing, getting status, and writing data to the parallel port.
- n INT 1Ah (PCI BIOS INT Support)
 - Provides a real-mode, INT 1Ah interface to many of the B1h class subfunctions defined in revisions 2.2.0 and newer of the PCI BIOS Specification.
- n INT 1Ah (System Time/Date Functions)
 - Supports most of the standard functions for getting and setting the time and date in the system, including the alarm.
- n I/O Companion Register Initialization
 - Downloads a table of default values into the PCI-ISA bridge portion of the I/O companion device following a power-on reset. Subsequent sequences of code (e.g., SMM software) may change the default values as features are enabled, disabled, or re-configured.
- n LCD Panel Initialization
 - XpressROM can configure and enable an LCD flat panel. TFT and DSTN panels are supported, by default. The system designer, using the configuration tool must simply select the panel type, resolution, and DSTN controller access method (if appropriate) in order to include this support.
- n Legacy USB
 - XpressROM may optionally be configured to support legacy USB, thereby allowing the use of a USB keyboard and mouse before a driver is loaded by the operating system. Associated with this functionality is a “virtual keyboard controller” module, allowing for a standard keyboard BIOS look and feel, even in the absence of an keyboard controller.
- n Legacy BIOS Entry Points
 - XpressROM provides the legacy BIOS function entry points for use by legacy software as a compatibility feature.
- n Movable GX_BASE
 - The location of GX_BASE (see the definition of the Graphics Control Register in the Geode GX1 databook) can be configured at build time. This can allow extra flexibility for the system designer.
- n PCI ‘_32_’ Services Directory Table
 - Provides a ‘_32_’ bit services directory that provides the operating system with information regarding the location of the 32-bit PCI BIOS API. Some operating systems use the information in this table to determine the presence and location of 32-bit, “flat-mode” PCI BIOS services. The ‘_32_’ bit services directory is described in revisions 2.1 and newer of the PCI BIOS Specification.
- n PCI BIOS 32-Bit Protected Mode API
 - Provides a flat-mode, 32-bit protected mode interface to those PCI bus access functions defined in revisions 2.1 and newer of the PCI BIOS Specification.
- n Power-On Splash Screen (Supports .BMP Format)
 - Contains special sequences of ROM-based code for displaying OEM-supplied “splash” screens during the power-on sequence, rather than the traditional text-mode DRAM count and similar displays. The current XpressROM firmware implementation supports 640x480, 800x600, and 1024x768 .BMP file formats.
- n Power-On Time Option ROM Invocation/Initialization
 - Shadows and invokes any option ROMs that adhere to the format specified in documents such as “The IBM PS/2 Model 60 Technical Reference Manual.” Such option ROMs contain a special “55h/AAh” header, size byte, and initialization entry point. The XpressROM firmware searches for and invokes any valid ROMs that reside on 2 KB boundaries in the region C800:0h-DF80:0h. Option ROMs can be found on ISA cards, PCI devices, or may be embedded in the XpressROM image itself.
- n Power-On Time PCI Enumeration/Configuration
 - Automatically detects, initializes, and configures any PCI 2.1 compliant devices that appear in PCI configuration space following a power-on sequence. The XpressROM firmware contains a resource allocation module that assigns I/O, address space, and IRQ resources to PCI devices and ensures that the resources assigned to these devices are conflict free. Additionally, the XpressROM firmware programs the steering and edge/level values of any IRQs assigned to PCI devices during the power-on sequence.

Features (Continued)

- n Protected Mode Reset
 - XpressROM supports the 286-style reset typically used for returning the microprocessor from Protected Mode to Real Mode. Currently functions 05h and 0Ah are supported. The caller resets the microprocessor by writing the function value into CMOS location 0Fh and then resetting the system.
- n RLE Compression/Decompression for BIOS Modules (Plus RLE Utility)
 - Provides developers the capability of storing code and data modules in compressed form within the system flash or EEPROM device. Compressed modules contain a header that identifies the type, size, target decompression segment, compression method, and other information used to build or manipulate that particular compressed module. During a power-on or reset sequence, and following DRAM bank configuration, the XpressROM firmware decompresses such modules into system shadow RAM space. Compression usage has the advantage of conserving build time ROM device space, enabling developers to include significantly more functionality in a ROM device than would be possible without the use of compression. A DOS utility (RLE.EXE) is provided that compresses source ROM modules and supplies these modules with the correct header information.
- n SDRAM Bank Sizing/Geode Processor DRAM Controller Initialization
 - Determines the amount of memory installed in the system's SDRAM banks, configures the processor's SDRAM controller with the appropriate row/column information, and initializes the SDRAM timings. If the SDRAM modules support SPD (Serial Presence Detection) EEPROMs, special XpressROM firmware code queries the SPD EEPROMs via a standard I²C interface and may optimize the SDRAM timings based on the reported capabilities of a particular SDRAM module.
- n SMI-Based Memory Access Capability
 - A feature of the SMI handler most often useful for debugging in unfamiliar or highly protected environments is the ability to read or write any addressable location in the system. This is done through a virtual register access.
- n SMM (System Management Mode) Initialization
 - Contains sequences of code to decompress, load, and invoke the SMI handler. These sequences of code also initialize SMI-specific registers, such as SMAR (System Management Address Register) SMHR (System Management Header Register) and GX_BASE, each of which is required for successful initialization of the SMM software.
- n Summary Screen
 - Displays a summary of vital system information immediately before starting the operating system. This includes a list of hardware found, CPU revision and speed, memory size, etc.
- n SuperI/O Device Initialization
 - The SuperI/O device initialization portion of the XpressROM firmware is a platform-dependent module that initializes the traditional ISA bus components that reside in the SuperI/O device. For example, in a system that contains National Semiconductor's PC97317 SuperI/O, this portion of the XpressROM firmware code configures and assigns resources to the floppy disk controller, serial COM ports, parallel LPT port, and 8042 keyboard controller.
- n Text-Based Configuration Utility
 - The XpressROM build process includes a series of questions in which the system designer provides information about the board wiring, options to install, and generally how XpressROM should behave. The tool runs automatically the first time the build process takes place and on subsequent builds uses the information from a saved option file (XPRES CFG.OPT)
- n Virtual Real-Time Clock and CMOS
 - Using the highly configurable SMI generation capabilities of the Geode devices, the SMI handler of XpressROM may optionally contain a module to emulate the functionality of an RTC/CMOS. While the CMOS and clock states cannot be maintained across a power loss, a system designer may elect to forego a physical RTC, still allowing many types of software to run correctly.
- n Virtual UART
 - The SMI handler of XpressROM may also contain a module to emulate the functionality of one or more UARTs. The module provides the look and feel of a standard UART, and allows flexibility to add the functionality for delivering or retrieving data from some hardware source (e.g., an on-board microcontroller).
- n CMOS Setup
 - Setup is new to XpressROM and provides a configurable engine for configuring a target platform.

Table of Contents

1.0	Specification Introduction	8
1.1	SUPPORTED INT/FUNCTIONS	8
1.2	ACRONYMS AND DEFINITIONS	11
1.3	REFERENCES	11
2.0	Code Structure	12
2.1	USER DIRECTORY	12
2.2	OVERRIDES	13
3.0	POST	15
4.0	Removable Media	18
4.1	FLOPPY DISK DRIVE SUPPORT	18
4.2	INT 13H FUNCTION DESCRIPTIONS	18
5.0	Non-Removable Media	27
5.1	ATA DEVICE INT 13H SUPPORT	27
5.2	INT 13H FUNCTIONS DESCRIPTIONS	28
5.2.1	INT 13h Extensions API	36
5.2.1.1	Calling Conventions	37
5.2.1.2	INT 13h Extensions API Subsets	37
5.2.1.3	INT 13h Functions Extensions Descriptions	38
5.3	INT 13H ATA DEVICE (FIXED DISK) DATA STRUCTURES	42
5.3.1	BIOS Data Area (BDA)	42
5.3.2	Extended BIOS Data Area	46
5.4	INT 41H/46H VECTORS	47
5.4.1	Standard Disk Drive Parameter Table	47
5.4.2	Enhanced Disk Drive Parameter Table	47
5.4.3	Fixed Disk Parameter Table Extensions	48
6.0	PCI Support	52
6.1	INT 1AH/FUNCTION B1H PCI BIOS SUBFUNCTIONS DESCRIPTIONS	53
6.1.1	Reading and Writing Configuration Space	56
7.0	Serial Port BIOS Support	59
7.1	INT 14H SERIAL PORT BIOS FUNCTIONS DESCRIPTIONS	59
8.0	Parallel Port BIOS Support	62
8.1	INT 17H PARALLEL PORT BIOS FUNCTIONS DESCRIPTIONS	62
9.0	Human Interface Device Support	64
9.1	INT 16H FUNCTIONS DESCRIPTIONS	64
9.2	PS/2 POINTING DEVICE SUPPORT	66
10.0	System Time and Date	67
10.1	INT 1AH TIME AND DATE FUNCTIONS DESCRIPTIONS	67

Table of Contents (Continued)

11.0	System Services	72
11.1	INT 15H SYSTEM SERVICES FUNCTIONS DESCRIPTIONS	72
11.1.1	I ² C Access Functions Specific to Geode processors	77
11.1.2	NVRAM Access Functions Specific to Geode Processors:	78
11.1.3	INT 15h function For XpressROm information	85
11.1.4	New heading and description for Cxh functions	85
11.1.5	PS/2 Mouse Support	86
12.0	Miscellaneous Services	92
12.1	MISCELLANEOUS SERVICES FUNCTIONS DESCRIPTIONS	92
13.0	System Management Mode Software	93
13.1	ARCHITECTURE	93
13.2	FIRMWARE REQUIREMENTS	94
13.3	SYSTEM MANAGER	95
13.4	EVENTS	95
13.5	MESSAGING	95
13.6	SYSTEM CALLS	95
13.7	RESOURCE MANAGEMENT	96
13.7.1	Reentrancy	96
13.7.2	Event Registration	96
13.7.3	Nested SMIs	97
13.8	VIRTUAL SUPPORT MODULES	97
13.8.1	Memory Model	99
13.8.2	Initialization	99
13.8.3	VSM Header	99
13.9	VIRTUAL REGISTERS	101
13.10	SYSTEM CALLS AND MACROS	107
13.11	MESSAGES	114
13.12	EVENTS	117
13.13	MACROS	121
13.13.1	ENTER_CRITICAL_SECTION & EXIT_CRITICAL_SECTION	123
13.13.2	General Purpose Register Access	123
13.13.3	Shorthand Macros	125
13.13.4	PCI Register Access	126
14.0	Virtual Hardware	129
14.1	VIRTUAL AUDIO	129
14.2	VIRTUAL VIDEO	130
14.3	VIRTUAL RTC/CMOS	131
15.0	Power Management	132
15.1	POWER STATES	132
15.2	PM INITIALIZATION	132
16.0	Configuring/Customizing	134

Table of Contents (Continued)

16.1	QUICK START	134
16.2	BUILD PROCESS:	134
16.3	ENVIRONMENT VARIABLE(S)	134
16.4	THE CONFIGURATOR	134
16.5	BUILD FILES	135
16.6	BUILD OPTIONS	135
16.7	SPLASH SCREEN	135
16.8	SUMMARY SCREEN	136
16.9	CMOS SETUP	137
16.9.1	SETUP MENUS	137
16.9.1.1	General Menu Navigation.	137
16.9.1.2	Setup Main Menu	138
16.9.1.3	Motherboard I/O Device Configuration Menu	139
16.9.1.4	Memory Optimization Menu	140
16.9.1.5	TV Output Menu.	142
16.9.1.6	LPC Card Devices Menu	143
16.9.2	Programming Reference	144
16.9.2.1	Sample SMENU.ASM file:	144
16.9.2.2	MACRO REFERENCE	145
17.0	Debugging	152
17.1	MTEST PINS:	152
17.2	GPCS#	152
18.0	Utilities	153
18.1	CUSTOM BUILD UTILITIES BY NATIONAL SEMICONDUCTOR	153
18.1.1	X2ROM.EXE – .EXE to .ROM Conversion Utility Version 1.0	153
18.1.2	FLASHROM.COM – Flash ROM Device Update Utility	155
18.1.3	RLE.EXE – Run Length Encoded compression Utility	156
18.1.4	BMPFLIP.EXE – Windows .BMP File Conditioning Utility	157
18.1.5	CPRPAD.EXE – ComPressed ROM Pad Utility	157
18.1.6	ROMPAD.EXE – Boot Loader Image Pad Utility	157
18.1.7	BINCOPY.EXE	158
18.2	OFF-THE-SHELF BUILD UTILITIES	158
Appendix A	Support Documentation	159
A.1	SPECIFICATION REVISION HISTORY	159

1.0 Specification Introduction

This specification documents the software interfaces supported within the components of XpressROM firmware. Most of these are standard interfaces, for which information is readily available from external sources, but there are some unique concepts as well. This specification also discusses how to use the XpressROM source code, including building projects and modifying the current functionality. The XpressROM Functional Specification is intended for BIOS, or deployment engineers working on National Semiconductor Geode technology based designs.

The remaining subsections of this chapter describe:

- Summary of supported INTs, functions, and subfunctions.
- Acronyms and definitions.
- A list of related reference material.

1.1 SUPPORTED INT/FUNCTIONS

The XpressROM firmware supports several INT instructions and associated functions/subfunctions which are summarized in Table 1-1. Included in the table is a page reference where the corresponding INT/Function/Subfunction's description is located.

Table 1-1. Summary of Supported INTx Functions

INTx	Function	Subfunction	Function Description	Reference
INT 11h: Miscellaneous Service	---	---	Equipment List	Page 92
INT 12h: Miscellaneous Services	---	---	Get Memory Size	Page 92
INT 13h: Removable and Non-Removable Media Support	00h	---	Reset Disk Subsystem	Page 18 Page 28
	01h	---	Get Disk Subsystem Status	Page 19 Page 29
	02h	---	Read Sectors	Page 20 Page 30
	03h	---	Write Sectors	Page 21 Page 31
	04h	---	Verify Sectors	Page 22 Page 32
	05h	---	Format Track (Floppy Disk)	Page 23
	08h	---	Get Drive Parameters	Page 23 Page 33
	09h	---	Set Drive Parameters	Page 34
	0Ch	---	Seek To Cylinder	Page 34
	0Dh	---	Alternate Disk Subsystem Reset	Page 24 Page 35
	10h	---	Test Drive Ready	Page 35
	11h	---	Recalibrate Drive	Page 35
	14h	---	Perform Disk Self-Diagnostic	Page 36
	15h	---	Get Disk Type	Page 24 Page 36
	16h	---	Get Disk Change Status	Page 25
	17h	---	Set Disk Type	Page 25
	18h	---	Set Media Type	Page 26
	41h	---	Check Extensions Present	Page 38
	42h	---	Extended Read Sectors	Page 38
	43h	---	Extended Write Sectors	Page 39
	44h	---	Extended Verify Sectors	Page 39
	47h	---	Extended Seek To Cylinder	Page 40
	48h	---	Extended Get Drive Parameters	Page 40

Specification Introduction (Continued)**Table 1-1. Summary of Supported INTx Functions (Continued)**

INTx	Function	Subfunction	Function Description	Reference
INT 14h: Serial Port BIOS Support	00h	---	Initialize Port	Page 59
	01h	---	Write Character to Port	Page 60
	02h	---	Read Character from Port	Page 60
	03h	---	Get Port Status	Page 61
INT 15h: System Service Support	24h	00h	Disable A20 Gate	Page 72
		01h	Enable A20 Gate	Page 74
		02h	Get A20 Gate Status	Page 74
	87h	---	Extended Memory Block Move	Page 75
	88h	---	Get Extended Memory Size	Page 75
	89h	---	Enter Protected Mode	Page 76
	BEh	00h	Read I ² C Byte	Page 77
		01h	Write I ² C Byte	Page 77
		02h	Write I ² C Block	Page 77
		03h	Read NV Data	Page 78
		04h	Write NV Data	Page 78
		05h	List NV Tokens	Page 79
		06h	Get NVRAM Checksum	Page 79
		07h	Set NVRAM Checksum	Page 79
		08h	Reset NVRAM Default	Page 79
		09h	Set NVRAM Table Address	Page 80
		99h	Read XpressROM Information	Page 85
	C0h	---	Get ROM Configuration	Page 85
	C1h	---	Get EBDA Address	Page 86
	C2h	00h	Enable/Disable Pointing Device	Page 86
		01h	Reset Pointing Device	Page 87
		02h	Set Sample Rate	Page 87
		03h	Set Resolution	Page 88
		04h	Read Device Type	Page 88
		05h	Initialize Pointing Device Interface	Page 88
		06h	Set Scaling or Get Status	Page 88
		07h	Set Pointing Device Handler Address	Page 89
	E8h	01h	Get System Memory Size	Page 89
		20h	Get System Memory Map	Page 90
INT 16h: Human Interface Device Support	00h	---	Wait For Character	Page 64
	01h	---	Check For Key Present	Page 64
	02h	---	Get Shift Status	Page 65
	05h	---	Insert Key Code In Keyboard Buffer	Page 65
	10h	---	Get Extended Key Code	Page 65
	11h	---	Check For Enhanced Key Code	Page 66
	12h	---	Get Extended Shift Status	Page 66

Specification Introduction (Continued)**Table 1-1. Summary of Supported INTx Functions (Continued)**

INTx	Function	Subfunction	Function Description	Reference
INT 17h Parallel Port BIOS Support	00h	---	Print a Character	Page 62
	01h	---	Initialize Parallel Port	Page 62
	02h	---	Get Port Status	Page 63
INT 1Ah: System Time and Date Support	00h	---	Get System Time Status	Page 67
	01h	---	Set System Time	Page 67
	02h	---	Get Real-Time Clock Time Status	Page 68
	03h	---	Set Real-Time Clock Time	Page 68
	04h	---	Get Real-Time Clock Date Status	Page 68
	05h	---	Set Real-Time Clock Date	Page 69
	06h	---	Set Real-Time Clock Alarm	Page 69
	07h	---	Cancel Real-Time Clock Alarm	Page 69
	09h	---	Get Real-Time Clock Alarm Status	Page 70
	0Ah	---	Read System Day Counter	Page 70
	0Bh	---	Set System Day Counter	Page 70
	0Eh	---	Get Real-Time Clock Date/Time Alarm and Status	Page 71
	0Fh	---	Initialize Real-Time Clock	Page 71
INT 1Ah: PCI BIOS Support	B1h	01h	PCI_BIOS_PRESENT	Page 53
		02h	FIND_PCI_DEVICE	Page 54
		03h	FIND_PCI_CLASS_CODE	Page 54
		06h	GENERATE_SPECIAL_CYCLE	Page 55
		08h	READ_CONFIG_BYTE	Page 56
		09h	READ_CONFIG_WORD	Page 56
		0Ah	READ_CONFIG_DWORD	Page 57
		0Bh	WRITE_CONFIG_BYTE	Page 57
		0Ch	WRITE_CONFIG_WORD	Page 57
		0Dh	WRITE_CONFIG_DWORD	Page 58
INT 1Bh: Miscellaneous Services	---	---	Control Break Handler	Page 92

Specification Introduction (Continued)

1.2 ACRONYMS AND DEFINITIONS

This specification uses several acronyms. Table 1-2 lists, in alphabetical order, the acronyms and their definitions.

Table 1-2. Acronyms and Definitions

Acronym	Definition
API	Applications Programming Interface
APM	Advanced Power Management
BDA	BIOS Data Area
BIOS	Basic I/O System
CHS	Cylinder/Head/Sector
EBDA	Extended BIOS Data Area
HID	Human Interface Device
LBA	Large Block Address
OEM	Original Equipment Manufacturer
PM	Power Management
POST	Power-On Self Test
SIO	SuperI/O
SMAR	System Management Address Register
SMHR	System Management Header Register
SMI	System Management Interrupt
SMM	System Management Mode
SoftVGA	SMI-based emulation of VGA controller functionality
TSR	Terminate, Stay Resident
VSA	Virtual System Architecture
VSM	Virtual Support Module

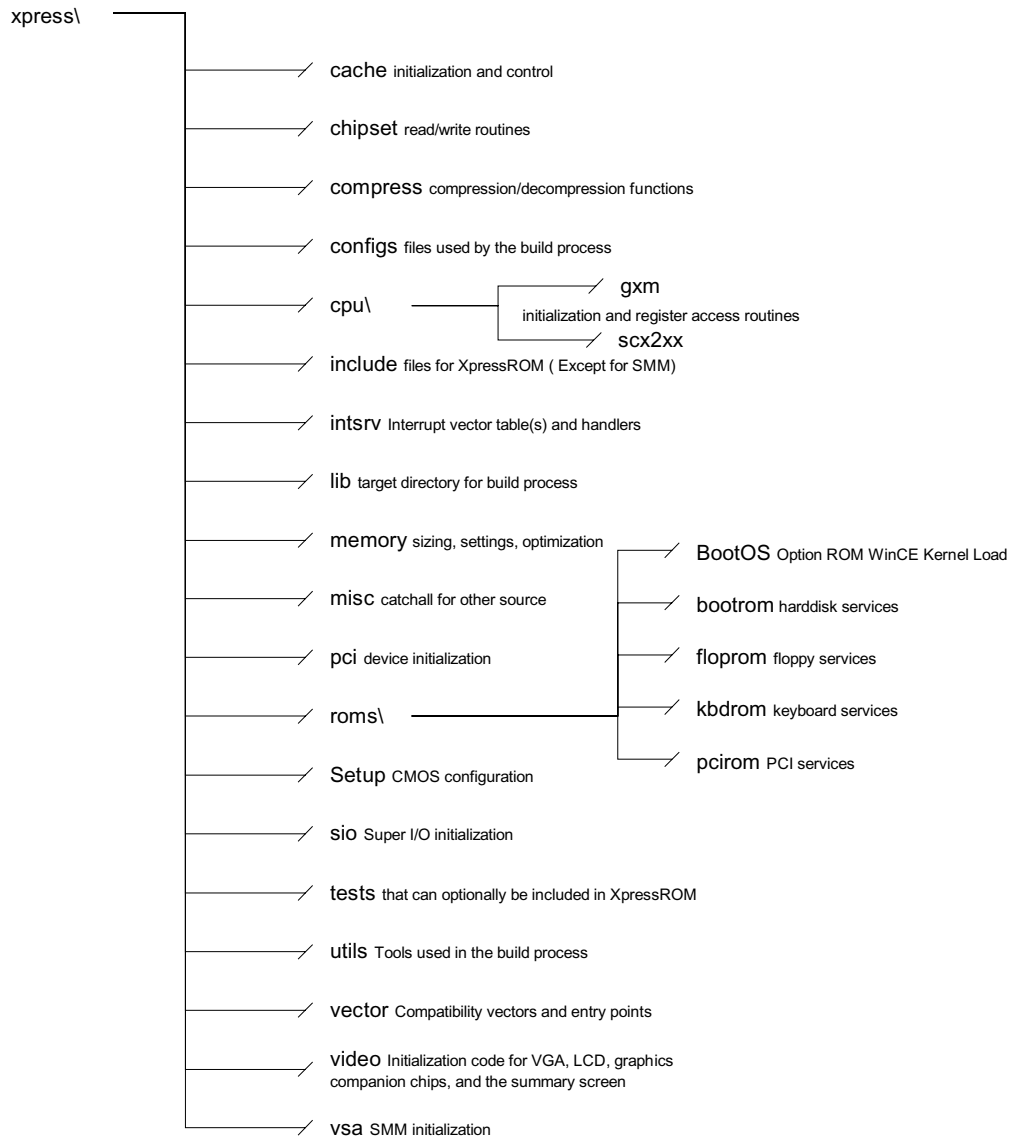
1.3 REFERENCES

Listed below is specific documentation regarding devices, standards, and architecture that is either mentioned in this functional specification, or that the reader may find useful.

- National Semiconductor Geode™ GX1 Processor Data Book
Revision 2.0, July 2000
(and addenda)
- National Semiconductor Geode™ CS5530 I/O Companion Data Book
Revision 4.0, July 1999
- National Semiconductor Virtual System Architecture® BIOS Porting Guide
Revision 4.0, March 23, 1999
- PCI BIOS Specification
Revision 2.1, August 26, 1994
- Advanced Power Management (APM) BIOS Interface Specification
Revision 1.2, February, 1996
- BIOS Enhanced Disk Drive Specification
Version 3.0, March 12, 1998

2.0 Code Structure

The directory structure for the XpressROM firmware is as follows:



2.1 USER DIRECTORY

All files specific to a project are stored in the user directory. The default user directory is XPRESS\USER, but this generally should not be used as the project's user directory. It is recommended that a separate subdirectory is made to hold the user directories outside of the Xpress tree. A sample makefile is included in the XPRESS\USER directory. A recommended directory structure for user directories is as follows:

```

XPRESS\
PROJECTS\PROJ1
        \PROJ2
  
```

The files normally found in the user directories are:

- Makefile – The make file sets the user directory, then calls the master makefile.

Code Structure (Continued)

- Splash.bmp – This is a standard Windows bitmap file which is used if splash screen support is needed.
- Userule.mak – Any additional things which need to be compiled may be added here.
- Xromcfg.opt – The configurator save file for the project.
- User option ROMS
- Any overrides

2.2 OVERRIDES

Any source file (.asm) in the build may be overridden simply by copying the file into the user directory or the project directory. When XpressROM is re-built, the copy of the file in one of these directories is used instead of the copy in the Xpress tree. The SMI handler may be overridden by the files VSA.ROM or VSA2.ROM being copied into the user directory. This allows easy changes on a project by project basis. Any files which need to be changed for a project should be changed this way. Editing the files in the Xpress tree is discouraged as it creates problems when upgrading to a new XpressROM release.

Code Structure (Continued)

3.0 POST

This section outlines the POST (Power-On Self Test) execution flow for the XpressROM firmware, along with the associated checkpoints which are sent to I/O Port 80h. POST is executed out of xpress.asm. In general, the numbering scheme is as follows:

- Reset - 69h, 71h
- Core functions - Numbered in chronological order, starting at 00h
- User functions - 80h and 81h
- Intermediate checkpoints - Numbered in chronological order, beginning on a 10h boundary
- Failing tests - Numbered with xFh and the system halts

Table 3-1. POST Codes

Checkpoint	Function Name	Intermediate Codes	Function Description
F0h	Reset Vector	F0h	XpressROM code execution start
80h	UserPreInit	---	User added code to be run first
00h	PreSioInit	60h	First SIO access, Core SIO initialize, LPT can be mapped to I/O 80h for embedded devices to provide POST code output.
01h	ClockInit	---	Chipset level clock initialize
02h	GxmRegInit	---	I/O based Geode initialize
03h	UnReal	---	Create a 32-bit real mode descriptor
04h	gxmMemRegInit	---	Memory mapped I/O based Geode initialize
05h	gxmTest	---	Geode CPU tests
		20h	Test entry
		28h- 2Fh	Test stepping ID (2Fh = Test Failed)
		2Ah	Test feature
		2Eh	Test passed exit
06h	MemSetup	---	Autosize memory controller DIMM1 and DIMM0
		70h	Set the clock drive strength and shift value, mask the clocks
		72h	Set the data, address, and control drive, clear reference timer and VGA wrap
		73h	Set the register to indicate no DIMMs installed
		74h	Default CAS latency to 3
		75h or 7Fh	Memory sizing (7Fh = No memory located)
		76h	Do refresh
		7E	MemSetup exit
07h	memSetUpStack	---	Setup a stack
		90h or 9Fh	Create stack (9Fh = Stack creation failure)
		9Eh	Stack setup exit
08h	memTest	---	Test memory address lines
		B0h	Test memory
		BEh	Memory test exit
		BFh	Memory test failed (i.e., B0h)
09h	shadowRom	---	Copy ROM from F000:0000 to RAM at F000:0000

POST (Continued)**Table 3-1. POST Codes (Continued)**

Checkpoint	Function Name	Intermediate Codes	Function Description
0Ah	memRAMOptimize	---	SDRAM optimization for CPU speed
0Bh	cacheInit	---	Autosize and initialize Cache
		CFh	Cache failed to size to 16 KB (i.e., 0Bh failed)
0Ch	northBridgeInit	---	Load Geode North Bridge PCI register space
		E8h	Begin register initialize
0Dh	chipsetInit	---	Load CS55x0 with values aka south bridge
		E0	Begin register initialize
0Eh	sioTest	---	SIO test/initialize
		60h	SIO test entry
		61h	SIO present / register initialize
		6Eh	SIO test exit
0Fh	pcATjunk	---	Various PC/BIOS compatibility pieces
10h	intTable	---	Set up generic interrupt table
16h	equip_check		Set up BDA for installed hardware
11h	memInfo	---	Query memory controller for RAM size, and store it
14h	keyboardInit	---	Wakeup the keyboard controller
12h	romCopy	---	SMI Handler, video, keyboard, hard drives, splash screen, LCD
		D0h	Get shadow ready for copy
		D1h	Decompress all images
		D2h	Call SMI Handler initialization code
		D3h	Return from SMI Handler initialize VGA BIOS
		D4h	Initialize LCD Panel
		D5h	Display splash screen
		D6h	Hard Disk Drive Initialize
		D7h	System option ROM scan and initialize / PCI32 fixup
		DEh	romCopy exit
15h	gxmCacheOff gxmCacheOn	---	Wiggle cache - if MTEST enabled, cache remains off
82h	EquipmentCheck	---	Additional BDA initialization, CMOS initialization
17h	pciScan	---	Scan PCI bus and enumerate
18h	optionRomInit	---	Scan and call options ROMs from C800-EFF0
19h	ResetLimits	---	Reset all descriptors to real mode 1 MB size
81h	UserPostInit	---	User added code to be run right before boot
1Ah	Summary_screen	---	Display summary screen
1Bh	---	---	Attempt to boot via INT 18h then INT 19h
1Fh	---	---	Boot failure - XpressROM halted

POST (Continued)

4.0 Removable Media

This chapter discusses the support that XpressROM firmware provides for removable media devices, such as floppy disk drives.

4.1 FLOPPY DISK DRIVE SUPPORT

The XpressROM firmware provides INT 40h and boot support for one 1.44 MB, 3.5" floppy disk drive. As in a traditional boot ROM, only device 0 (A: drive) is bootable. The proper access method for the INT 40h functions is through the INT 13h service routines. The XpressROM firmware supports the following INT 40h functions for the floppy drive:

- Function 00h: Reset Disk Subsystem
- Function 01h: Get Disk Subsystem Status
- Function 02h: Read Sectors
- Function 03h: Write Sectors
- Function 04h: Verify Sectors
- Function 05h: Format Track (Floppy Disk)
- Function 08h: Get Drive Parameters
- Function 0Dh: Alternate Disk Subsystem Reset
- Function 15h: Get Disk Type
- Function 16h: Get Disk Change Status
- Function 17h: Set Disk Type
- Function 18h: Set Media Type

The format of the caller-supplied parameters and return values adheres to standard industry descriptions of the INT 13h API for floppy drives as outlined in innumerable references, such as the "IBM/PC Technical Reference Manual," "PC Interrupts", etc. A brief description of each function is described next.

4.2 INT 13H FUNCTION DESCRIPTIONS

This section describes the parameters and return values for each floppy disk-specific INT 13h function in the XpressROM device. This section is intended as a guide; developers interested in extending and/or debugging the XpressROM firmware INT 13h interface should consult a document such as those described in Section 1.3 "References" on page 11.

INT 13h provides the operating system with a series of functions for communicating with the system's fixed and floppy disk drives. The INT 13h interface is a cornerstone of PC/AT compatibility because it ensures that software applications such as DOS and Microsoft's Windows can read and write disk-based data in a uniform fashion across a broad variety of architectures.

Function 00h – Reset Disk Subsystem

Supports:

Fixed disks, floppy disks

Description:

In an ATA drive-equipped system, this function toggles bit four in register six of the Control Register Block. This causes the disk(s) (see "Special") to recalibrate internally and seek to cylinder zero.

Passed:

AH = 00h
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = Status Code - see AH in "Function 01h – Get Disk Subsystem Status" on page 19

Special:

If DL ≤ 80h, both the fixed disks and floppy disks reset.

Related Functions:

INT 13h "Function 01h – Get Disk Subsystem Status" on page 19
INT 13h "Function 0Dh – Alternate Disk Subsystem Reset" on page 24

Removable Media (Continued)**Function 01h – Get Disk Subsystem Status****Supports:**

Fixed disks, floppy disks (see “Returns” below)

Description:

Returns the status of the last disk operation in the AH register.

Passed:

AH = 01h
DL = Drive number

Returns:

AH = Status of the last disk operation.
 00h = Operation completed successfully (fixed, floppy)
 01h = Illegal/Unrecognized command (fixed, floppy)
 02h = Sector address mark not found (fixed, floppy)
 03h = Attempt to write to write-protected disk (fixed, floppy)
 04h = Sector not found (fixed, floppy)
 05h = Reset failure (fixed only)
 06h = Diskette change signal (floppy only)
 07h = Parameter activity failed (fixed only)
 08h = DMA overrun (floppy only)
 09h = DMA operation would cross a 64 KB boundary (floppy only)
 0Ah = Bad or invalid sector (fixed only)
 0Bh = Bad or invalid cylinder (fixed only)
 0Ch = Invalid cylinder number (fixed only)
 0Dh = Invalid number of sectors supplied during format (fixed only)
 0Eh = Controller detected control data address mark (fixed only)
 0Fh = DMA arbitration failure (fixed only)
 10h = CRC/ECC bad (fixed only)
 11h = Data corrected using ECC (fixed only)
 20h = Controller failed self test (fixed only)
 40h = Error during seek (fixed only)
 80h = Command timed out (fixed only)
 AAh = Drive not ready (fixed only)
 BBh = Unknown error (fixed only)
 CCh = Write error (fixed only)
 E0h = Status register error (fixed only)
 FFh = Media sense failed (fixed only)

Special:

The system BIOS stores the status of the last fixed disk operation in location 40:74h in the BDA. The preferable method for retrieving disk status is the INT 13h interface which enables system software other than the system BIOS to provide accurate disk subsystem status.

Related Functions:

None

Removable Media (Continued)**Function 02h – Read Sectors****Supports:**

Fixed disk, floppy disk

Description:

Reads a caller-specified number of sectors in the buffer specified by the ES:BX register pair.

Passed:

AH = 02h
 AL = Sector count
 CH = Cylinder bits [7:0]
 CL = Bits [7:6] = Cylinder bits [9:8]
 Bits [5:0] = Start sector (1-based)
 DH = Drive Head
 DL = Drive number
 ES:BX = Far pointer to caller-supplied buffer

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 19
 AL = Total number of sectors read

Special:

During a floppy diskette read, the caller ensures that the read operation:

- 1) Does not cross a 64 KB DMA page boundary (e.g., 2000:0h, 3000:0h, etc.).
- 2) Does not extend past the end of a physical cylinder.

The IBM PC/AT technical reference manual specifies the head register as a 4-bit value; the upper four bits of DH are undefined on this type of system. Newer systems allow for a head value of 0-0FFh, or a maximum 255 heads. The INT 13h functions in this type of system are capable of translating between physical (drive reported) and logical cylinder, head, sector geometries.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 19
 INT13h “Function 03h – Write Sectors” on page 21

Removable Media (Continued)**Function 03h – Write Sectors****Supports:**

Fixed disk, floppy disk

Description:

Writes a caller-specified number of sectors in the buffer specified by the ES:BX register pair.

Passed:

AH = 03h
 AL = Sector count
 CH = Cylinder bits [7:0]
 CL = Bits [7:6] = Cylinder Bits [9:8]
 Bits [5:0] = Start sector (1-based)
 DH = Drive head
 DL = Drive number
 ES:BX = Far pointer to caller-supplied buffer

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 19
 AL = Total number of sectors written

Special:

During a floppy diskette write, the caller ensures that the write operation:

- 1) Does not cross a 64 KB DMA page boundary (e.g., 2000:0h, 3000:0, etc.).
- 2) Does not extend past the end of a physical cylinder.

The IBM PC/AT technical reference manual specifies the head register as a 4-bit value; the upper four bits of DH are undefined on this type of system. Newer systems allow for a head value of 0-0FFh, or a maximum 255 heads. The INT 13h functions in this type of system are capable of translating between physical (drive reported) and logical cylinder, head, sector geometries.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 19
 INT 13h “Function 02h – Read Sectors” on page 20

Removable Media (Continued)**Function 04h – Verify Sectors****Supports:**

Fixed disk, floppy disk

Description:

Verifies the existence and integrity of the sectors specified by the caller.

Passed:

AH = 04h
 AL = Sector count
 CH = Bits [7:0] = Cylinder bits [7:0]
 CL = Bits [7:6] = Cylinder bits [9:8]
 Bits [5:0] = Start sector (1-based)
 DH = Drive head
 DL = Drive number

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 19
 AL = Number of sectors successfully written

Special:

During a floppy diskette verify, the caller ensures that the verify operation:
 1) Does not cross a 64 KB DMA page boundary (e.g., 2000:0h, 3000:0, etc.).
 2) Does not extend past the end of a physical cylinder.

The IBM PC/AT technical reference manual specifies the head register as a 4-bit value; the upper four bits of DH are undefined on this type of system. Newer systems allow for a head value of 0-0FFh, or a maximum 255 heads. The INT 13h functions in this type of system are capable of translating between physical (drive reported) and logical cylinder, head, sector geometries.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 19
 INT 13h “Function 02h – Read Sectors” on page 20
 INT 13h “Function 03h – Write Sectors” on page 21

Removable Media (Continued)**Function 05h – Format Track (Floppy Disk)****Supports:**

Floppy disk only

Description:

Performs a low level function on a single floppy diskette track. The calling application passes a far pointer to a 4-byte table specifying low level format parameters.

Passed:

AH = 05h
 AL = Number of sectors to create on this cylinder
 CH = Cylinder number
 CL = Starting sector (1-based)
 DH = Drive head (0 or 1)
 DL = Drive number
 ES:BX = Far pointer to array of 4-byte parameter tables

The format of a single, 4-byte parameter table appears here:

BYTE Track
 BYTE Head
 BYTE Sector
 BYTE Bytes/sector 0 = 128, 1 = 256, 2 = 512, 3 = 1024

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 19

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 19

Function 08h – Get Drive Parameters**Supports:**

Fixed disk, floppy disk

Description:

Returns the geometry of the caller-specified drive in terms of cylinders, heads, and sectors. If the specified drive is a “large” IDE drive whose physical cylinder count exceeds 1024, this function returns “logical” or “translated” parameters.

Passed:

AH = 08h
 DL = Drive number

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 19
 BL = Floppy drive type (AT, PS/2 only)
 DL = Number of installed drives of this type
 DH = Maximum head number (0-based)
 CH = Bits [7:0] = Maximum cylinder number bits [7:0]
 CL = Bits [7:6] = Maximum cylinder number bits [9:8]
 Bits [5:0] = Maximum sector number bits [5:0]
 ES:DI = Far pointer to drive parameter table

Special:

The DL register returns what some manuals refer to as “the number of consecutive acknowledging drives”. This means, “the number of drives of this type”. For example, if your application calls this function with DL = 0, upon return DL contains the number of installed floppy drives.

Related Functions:

None

Removable Media (Continued)**Function 0Dh – Alternate Disk Subsystem Reset****Supports:**

Fixed disk, floppy disk

Description:

Reinitializes the fixed disk controller, resets the specified drive's parameters, and recalibrates the drive's heads (seek to track 0). Both the master drive and the slave drive respond to the reset function that is issued to either drive.

Passed:

AH = 0Dh
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = Status Code - see AH in "Function 01h – Get Disk Subsystem Status" on page 19

Related Functions:

INT 13h "Function 00h – Reset Disk Subsystem" on page 18

Function 15h – Get Disk Type**Supports:**

Fixed disk, floppy disk

Description:

Returns the type of disk that applies to the caller's DL register value.

Passed:

AH = 15h
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = If CF = 0: Type of drive corresponding to caller's DL register
0 = Disk not present
1 = Floppy disk drive without disk change detect
2 = Floppy disk drive with disk change detect
3 = Fixed disk
If CF = 1: Status Code - see AH in "Function 01h – Get Disk Subsystem Status" on page 19
CX:DX = Number of 512 byte sectors on disk

Related Functions:

INT 13h "Function 17h – Set Disk Type" on page 25

Removable Media (Continued)**Function 16h – Get Disk Change Status****Supports:**

Floppy disk only

Description:

Returns the value of the floppy controller disk status change line, indicating whether the user has removed or inserted a floppy diskette in the specified drive since the last read, write, or verify operation on that drive.

Passed:

AH = 16h
DL = Drive number

Returns:

AH = Disk change status, where:
00h = No status change has occurred
06h = Diskette status has changed

Special:

The operating system issues this function before each floppy diskette read or write to determine if its current FAT (file allocation table) record is valid. The value returned by this function is valid only if the system supports the disk change status line. Operating systems first invoke INT 13h Function 15h (Get Disk Type) to determine if the installed floppy disk controller and/or BIOS supports status change detection.

Related Functions:

INT 13h “Function 15h – Get Disk Type” on page 24

Function 17h – Set Disk Type**Supports:**

Floppy disk only

Description:

Used by older operating systems to specify the physical geometry of a floppy disk that is to be formatted. Modern operating systems use INT 13h Function 18h (Set Media Type) instead because it allows for non-standard geometries, such as those used to support 3-mode floppies in the Japan marketplace for NEC and Toshiba systems.

Passed:

AH = 17h
AL = Disk geometry, where:
00h = Diskette not present
01h = 360 KB disk in 360 KB drive
02h = 360 KB disk in 1.2 MB drive
03h = 1.2 MB disk in 1.2 MB drive
04h = 720 KB disk in 720 KB or 1.44 MB drive
DL = Drive number

Related Functions:

INT 13h Function 15h (Get Disk Type, Page 24)

Removable Media (Continued)**Function 18h – Set Media Type****Supports:**

Floppy disk only

Description:

This function is invoked by the operating system to specify the geometry of the diskette about to be formatted. It supersedes INT 13h Function 17h, although both are still valid for standard (non 3-mode) 5 1/4" and 3 1/2" disks.

Passed:

AH = 18h
DL = Drive number
CH = Cylinder count
CL = Sectors per track

Returns:

AH = 00h if specified geometry is not supported
01h if function is not supported
80h if drive is empty
ES:DI = Far pointer to 11-byte floppy diskette parameter table

Related Functions:

INT 13h "Function 05h – Format Track (Floppy Disk)" on page 23
INT 13h "Function 17h – Set Disk Type" on page 25

5.0 Non-Removable Media

The XpressROM firmware provides support for the following, non-removable media devices.

- SanDisk's CompactFlash (e.g., SanDisk) devices
- Traditional, rotating media IDE drives

Note: The term "ATA Device" describes those devices that support the command protocol described in the Small Form Factor Committee's ATA-4 Specification. Compact flash and rotating media IDE drives both belong to the family of ATA devices. This chapter uses the terminology "ATA device" to refer to IDE, compact flash, and any other similar devices.

The XpressROM firmware is a fully-featured implementation of the traditional (e.g., PC/AT) and extended INT 13h functions as described in the following (as well as countless other) documents concerning PC/AT architecture and programming:

- Enhanced Disk Drive Specification, Revision 1.1, Phoenix Technologies, 1995.
- INT 13h Extensions Reference, Revision 1.0, IBM Corporation, 1993.
- IBM PC/AT Technical Reference Manual, IBM Corporation, 1984.

5.1 ATA DEVICE INT 13H SUPPORT

The XpressROM firmware provides INT 13h run-time and boot support for as many as four ATA fixed disks.

As in a traditional boot ROM, only device 0x80 (C: drive) is bootable. The XpressROM firmware supports the following INT 13h functions for each installed ATA device. Functions that do not appear in this list are not supported by the XpressROM firmware.

- Function 00h: Reset Disk Subsystem
- Function 01h: Get Disk Subsystem Status
- Function 02h: Read Sectors
- Function 03h: Write Sectors
- Function 04h: Verify Sectors
- Function 08h: Get Drive Parameters
- Function 09h: Set Drive Parameters
- Function 0Ch: Seek To Cylinder
- Function 0Dh: Alternate Disk Subsystem Reset
- Function 10h: Test Drive Ready
- Function 11h: Recalibrate Drive
- Function 14h: Perform Disk Self-Diagnostic
- Function 15h: Get Disk Type
- Function 41h: Check Extensions Present
- Function 42h: Extended Read Sectors
- Function 43h: Extended Write Sectors
- Function 44h: Extended Verify Sectors
- Function 47h: Extended Seek To Cylinder
- Function 48h: Extended Get Drive Parameters

Note: Functions 41h-48h are INT 13h extensions and are listed in Section 5.2.1.3 "INT 13h Functions Extensions Descriptions" on page 38.

The format of the caller-supplied parameters and return values adheres to standard, industry descriptions of the INT 13h API for floppy drives as outlined in innumerable references, such as the "IBM/PC Technical Reference Manual," "PC Interrupts", etc. A description of each function is provided next.

Non-Removable Media (Continued)**5.2 INT 13H FUNCTIONS DESCRIPTIONS**

This section describes the parameters and return values for each ATA device-specific INT 13h function in the XpressROM firmware. This section is intended as a guide; developers interested in extending and/or debugging the XpressROM firmware INT 13h interface should consult a document such as those described in the previous section.

INT 13h provides the operating system with a series of functions for communicating with the system's fixed and floppy disk drives. The INT 13h interface is a cornerstone of PC/AT compatibility because it ensures that software applications such as DOS and Windows can read and write disk-based data in a uniform fashion across a broad variety of architectures.

Function 00h – Reset Disk Subsystem**Supports:**

Fixed disks, floppy disks

Description:

In an ATA drive-equipped system, this function toggles bit four in register six of the Control Register Block. This causes the disk(s) (see “Special”) to recalibrate internally and seek to cylinder zero.

Passed:

AH = 00h
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29

Special:

If DL ≤ 80h, both the fixed disks and floppy disks reset.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 29
INT 13h “Function 0Dh – Alternate Disk Subsystem Reset” on page 35

Non-Removable Media (Continued)**Function 01h – Get Disk Subsystem Status****Supports:**

Fixed disks, floppy disks (see “Returns” below)

Description:

Returns the status of the last disk operation in the AH register.

Passed:

AH = 01h
DL = Drive number

Returns:

AH = Status of the last disk operation.
 00h = Operation completed successfully (fixed, floppy)
 01h = Illegal/Unrecognized command (fixed, floppy)
 02h = Sector address mark not found (fixed, floppy)
 03h = Attempt to write to write-protected disk (fixed, floppy)
 04h = Sector not found (fixed, floppy)
 05h = Reset failure (fixed only)
 06h = Diskette change signal (floppy only)
 07h = Parameter activity failed (fixed only)
 08h = DMA overrun (floppy only)
 09h = DMA operation would cross a 64 KB boundary (floppy only)
 0Ah = Bad or invalid sector (fixed only)
 0Bh = Bad or invalid cylinder (fixed only)
 0Ch = Invalid cylinder number (fixed only)
 0Dh = Invalid number of sectors supplied during format (fixed only)
 0Eh = Controller detected control data address mark (fixed only)
 0Fh = DMA arbitration failure (fixed only)
 10h = CRC/ECC bad (fixed only)
 11h = Data corrected using ECC (fixed only)
 20h = Controller failed self test (fixed only)
 40h = Error during seek (fixed only)
 80h = Command timed out (fixed only)
 AAh = Drive not ready (fixed only)
 BBh = Unknown error (fixed only)
 CCh = Write error (fixed only)
 E0h = Status register error (fixed only)
 FFh = Media sense failed (fixed only)

Special:

The system BIOS stores the status of the last fixed disk operation in location 40:74h in the BDA. The preferable method for retrieving disk status is, however, the INT 13h interface which enables system software other than the system BIOS to provide accurate disk subsystem status.

Related Functions:

None

Non-Removable Media (Continued)**Function 02h – Read Sectors****Supports:**

Fixed disk, floppy disk

Description:

Reads a caller-specified number of sectors in the buffer specified by the ES:BX register pair.

Passed:

AH = 02h
 AL = Sector count
 CH = Cylinder bits [7:0]
 CL = Bits [7:6] = Cylinder bits [9:8]
 Bits [5:0] = Start sector (1-based)
 DH = Drive Head
 DL = Drive number
 ES:BX = Far pointer to caller-supplied buffer

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29
 AL = Total number of sectors read

Special:

During a floppy diskette read, the caller ensures that the read operation:

- 1) Does not cross a 64 KB DMA page boundary (e.g., 2000:0h, 3000:0h, etc.).
- 2) Does not extend past the end of a physical cylinder.

The IBM PC/AT technical reference manual specifies the head register as a 4-bit value; the upper four bits of DH are undefined on this type of system. Newer systems allow for a head value of 0-0FFh, or a maximum 255 heads. The INT 13h functions in this type of system are capable of translating between physical (drive reported) and logical cylinder, head, sector geometries.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 29
 INT13h “Function 03h – Write Sectors” on page 31

Non-Removable Media (Continued)**Function 03h – Write Sectors****Supports:**

Fixed disk, floppy disk

Description:

Writes a caller-specified number of sectors in the buffer specified by the ES:BX register pair.

Passed:

AH = 03h
 AL = Sector count
 CH = Cylinder bits [7:0]
 CL = Bits [7:6] = Cylinder Bits [9:8]
 Bits [5:0] = Start sector (1-based)
 DH = Drive head
 DL = Drive number
 ES:BX = Far pointer to caller-supplied buffer

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29
 AL = Total number of sectors written

Special:

During a floppy diskette write, the caller ensures that the write operation:

- 1) Does not cross a 64 KB DMA page boundary (e.g., 2000:0h, 3000:0, etc.).
- 2) Does not extend past the end of a physical cylinder.

The IBM PC/AT technical reference manual specifies the head register as a 4-bit value; the upper four bits of DH are undefined on this type of system. Newer systems allow for a head value of 0-0FFh, or a maximum 255 heads. The INT 13h functions in this type of system are capable of translating between physical (drive reported) and logical cylinder, head, sector geometries.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 29
 INT 13h “Function 02h – Read Sectors” on page 30

Non-Removable Media (Continued)**Function 04h – Verify Sectors****Supports:**

Fixed disk, floppy disk

Description:

Verifies the existence and integrity of the sectors specified by the caller.

Passed:

AH = 04h
 AL = Sector count
 CH = Bits [7:0] = Cylinder bits [7:0]
 CL = Bits [7:6] = Cylinder bits [9:8]
 Bits [5:0] = Start sector (1-based)
 DH = Drive head
 DL = Drive number

Returns:

CF = 0 if operation completed successfully
 1 if operation failed
 AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29
 AL = Number of sectors successfully written

Special:

During a floppy diskette verify, the caller ensures that the verify operation:
 1) Does not cross a 64 KB DMA page boundary (e.g., 2000:0h, 3000:0, etc.).
 2) Does not extend past the end of a physical cylinder.

The IBM PC/AT technical reference manual specifies the head register as a 4-bit value; the upper four bits of DH are undefined on this type of system. Newer systems allow for a head value of 0-0FFh, or a maximum 255 heads. The INT 13h functions in this type of system are capable of translating between physical (drive reported) and logical cylinder, head, sector geometries.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 29
 INT 13h “Function 02h – Read Sectors” on page 30
 INT 13h “Function 03h – Write Sectors” on page 31

Non-Removable Media (Continued)**Function 08h – Get Drive Parameters****Supports:**

Fixed disk, floppy disk

Description:

Returns the geometry of the caller-specified drive in terms of cylinders, heads, and sectors. If the specified drive is a “large” IDE drive whose physical cylinder count exceeds 1024, this function returns “logical” or “translated” parameters.

Passed:

AH = 08h
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29
BL = Floppy drive type (AT, PS/2 only)
DL = Number of installed drives of this type
DH = Maximum head number (0-based)
CH = Bits [7:0] = Maximum cylinder number bits [7:0]
CL = Bits [7:6] = Maximum cylinder number bits [9:8]
Bits [5:0] = Maximum sector number bits [5:0]
ES:DI = Far pointer to drive parameter table

Special:

The DL register returns what some manuals refer to as “the number of consecutive acknowledging drives”. This means, “the number of drives of this type”. For example, if your application calls this function with DL = 0, upon return DL contains the number of installed floppy drives.

Related Functions:

None.

Non-Removable Media (Continued)**Function 09h – Set Drive Parameters****Supports:**

Fixed disk only

Description:

Initializes the fixed disk controller with the parameters that correspond to the installed drive. The intent of this function was originally to allow a plug in ISA drive controller to service many different types of drives. Since IDE drives have an integrated controller, this function is theoretically no longer necessary, because the controller is specifically designed to support the geometry of the drive to which it is physically attached. However, many drive controllers still require that the BIOS issue this interrupt during POST.

Passed:

AH = 09h
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29

Special:

The vectors at 0:[41h*4] (INT 41h) and 0:[46h*4] (INT 46h) point to the parameter tables for drives 80h (C:) and 81h (D:), respectively. Issuing INT 13h Function 09h with DL = 80h or 81h causes the BIOS to download those parameters pointed to by the corresponding vector to the drive's controller. In a four drive system, there are no such interrupt vectors, and the actual location of the parameter tables for drives 82h (E:) and 83h (F:) are unknown to the caller.

Traditionally, fixed disks have a “native” cylinder/head/sector count. Ideally, it is these values that are used by the BIOS to initialize the drive during POST. This guarantees that the drive's interpretation of the caller's CHS values during a read/write/verify operation are always correct. Some drives don't “like” being initialized with non-native parameters and map sectors differently (and sometimes incorrectly) when initialized with non-native parameters.

Related Functions:

None

Function 0Ch – Seek to Cylinder**Supports:**

Fixed disk only

Description:

Moves the drive's read/write head to the caller-specified cylinder. In modern systems, it is used primarily for testing the performance of the drive.

Passed:

AH = 0Ch
DL = Drive number
DH = Head number
CH = Bits [7:0] = Cylinder bits [7:0]
CL = Bits [7:6] = Cylinder bits [9:8]

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29

Special:

In modern drives, a READ/WRITE/VERIFY command will cause the drive to seek to the caller-specified cylinder automatically. However, this function remains a favorite of disk performance benchmark applications.

Related Functions:

INT 13h “Function 01h – Get Disk Subsystem Status” on page 29

Non-Removable Media (Continued)**Function 0Dh – Alternate Disk Subsystem Reset****Supports:**

Fixed disk, floppy disk

Description:

Reinitializes the fixed disk controller, resets the specified drive's parameters, and recalibrates the drive's heads (seek to track 0). Both the master drive and the slave drive respond to the reset function that is issued to either drive.

Passed:

AH = 0Dh
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed
AH = Status Code - see AH in "Function 01h – Get Disk Subsystem Status" on page 29

Related Functions:

INT 13h "Function 00h – Reset Disk Subsystem" on page 28

Function 10h – Test Drive Ready**Supports:**

Fixed disk only

Description:

Returns whether the disk is ready.

Passed:

AH = 10h
DL = Drive number (80h = primary HDD, 81h = secondary HDD)

Returns:

CF = 0 if operation completed successfully
1 if operation failed (e.g., drive not ready)
AH = Status Code - see AH in "Function 01h – Get Disk Subsystem Status" on page 29

Related Functions:

None

Function 11h – Recalibrate Drive**Supports:**

XT, AT, PS/2 fixed disk only

Description:

Issues a recalibrate ATA command to the fixed disk specified in the DL register and returns the status of the operation.

Passed:

AH = 11h
DL = Drive number

Returns:

CF = 0 if operation completed successfully
1 if operation failed (e.g., drive not ready)
AH = Status Code - see AH in "Function 01h – Get Disk Subsystem Status" on page 29

Related Functions:

INT 13h "Function 01h – Get Disk Subsystem Status" on page 29

Non-Removable Media (Continued)**Function 14h – Perform Disk Self-Diagnostic****Supports:**

Fixed disk only

Description:

Performs controller diagnostics and returns the result.

Passed:

AH = 14h

Returns:CF = 0 if operation completed successfully
1 if operation failed

AH = Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29

Related Functions:

None

Function 15h – Get Disk Type**Supports:**

Fixed disk, floppy disk

Description:

Returns the type of disk that applies to the caller’s DL register value.

Passed:

AH = 15h

DL = Drive number

Returns:CF = 0 if operation completed successfully
1 if operation failed

AH = If CF = 0: Type of drive corresponding to caller’s DL register

0 = Disk not present

1 = Floppy disk drive without disk change detect

2 = Floppy disk drive with disk change detect

3 = Fixed disk

If CF = 1: Status Code - see AH in “Function 01h – Get Disk Subsystem Status” on page 29

CX:DX = Number of 512 byte sectors on disk

Related Functions:

None.

5.2.1 INT 13h Extensions API

The functions in this section were defined by IBM in the document entitled, “INT 13h Extensions API” as an alternate mechanism for handling large IDE drives and to provide a uniform API for devices that support removable media. The functions in this extended API all share a common data structure called the disk address packet, whose format appears here:

Disk_Address_Packet	STRUCT 1		
Packet_Size	db	16	; Size Of This Packet In Bytes
Reserved	db	0	; Reserved / Unused
Block_Count	dw	0	; Number Of Blocks (Sectors) To Transfer
pBuffer	dd	?	; SEG:OFFSET Address Of Transfer Buffer
Block_Offset	dq	?	; Starting Device Block Number

Disk_Address_Packer ENDS

Non-Removable Media (Continued)

Packet_Size — The supporting firmware examines the Packet_Size field to determine if it can support the caller's request. A value other than sixteen would indicate that the caller is requesting support for a revision of this API that does not yet exist. In this case, the BIOS would return with CF = 1 and an error code.

Block_Count — On input, Block_Count is the number of blocks (sectors) the caller wants to transfer. On output, it represents the number of blocks actually transferred. Zero is a legal value, and implies that no data is to be transferred.

pBuffer — The segment offset pointer to the start of the caller-supplied data transfer buffer.

Block_Offset — The absolute logical block number on the device.

5.2.1.1 Calling Conventions

Wherever possible, these extensions share the following register usage convention:

- DS:SI: Far pointer to disk address packet structure
- DL: Drive number (80h = C:, 81h = D:, etc.)
- AH: On input, contains the function index. On Output, contains the function return status.

The following return codes have been added to support these extensions:

- B0h: Volume not locked in drive
- B1h: Volume locked in drive
- B2h: Volume not removable
- B3h: Volume in use
- B4h: Lock count exceeded
- B5h: Valid eject request failed

5.2.1.2 INT 13h Extensions API Subsets

The INT 13h Extensions API allows implementors to support certain subsets of the entire interface without violating that specification. Each subset is well defined, and if supported must be done so in its entirety.

Large IDE Drive Support API Subset

This subset consists of the following functions:

- Function 41h: Check Extensions Present
- Function 42h: Extended Read Sectors
- Function 43h: Extended Write Sectors
- Function 44h: Extended Verify Sectors
- Function 47h: Extended Seek to Cylinder
- Function 48h: Extended Get Drive Parameters

Removable Media Lock/Eject API Subset

This subset consists of the following functions:

- Function 41h: Check Extensions Present
- Function 45h: Lock/Unlock Drive (not supported)
- Function 46h: Eject Drive (not supported)
- Function 48h: Extended Get Drive Parameters
- Function 49h: Extended Get Disk Change Status (not supported)
- INT 15h: Removable Media Eject Interrupt (not supported)

Note: If the system contains drives that support removable media and these drives have been assigned a drive index greater than 80h, the system firmware must support the Removable Media API subset for each of these drives.

The operating system determines API subset support by repeatedly issuing the Check Extensions Present function for each valid drive in the system.

Non-Removable Media (Continued)**5.2.1.3 INT 13h Functions Extensions Descriptions**

This section lists those INT 13h functions extensions that are supported in the XpressROM firmware, the parameters that system software supplies to each INT 13h function and the values that each function returns.

Function 41h – Check Extensions Present**Supports:**

Fixed disks only

Description:

Used to detect the presence or lack of INT 13h extensions support for a particular drive.

Passed:

AH = 41h
 BX = 55AAh
 DL = Drive index

Returns:

CF = 0 if extensions are supported for this drive, and:
 AX = Version of extensions (= 0100h)
 BX = 55AAh
 CX = API subset support bitmap
 Bit 0 = 1 if large IDE extensions are supported
 Bit 1 = 1 if Removable/Lock extensions are supported
 Bits [15:2] = Reserved/Unused in this specification revision
 CF = 1 if extensions support is not present for this drive, and:
 AH = 01h

Related Functions:

None

Function 42h – Extended Read Sectors**Supports:**

Fixed disks only

Description:

Used to read zero or more blocks of data into host memory.

Passed:

AH = 42h
 DL = Drive index
 DS:SI = Far pointer to caller-supplied data transfer buffer

Returns:

CF = 0 if the operation succeeded, and:
 AH = 0
 CF = 1 if the operation failed, and:
 AH = error code, and:
 Block_Count within the Disk_Address_Packet contains the number of sectors successfully transferred

Related Functions:

None

Non-Removable Media (Continued)**Function 43h – Extended Write Sectors****Supports:**

Fixed disks only

Description:

Used to write zero or more blocks of data into host memory.

Passed:

AH = 43h

DL = Drive index

DS:SI = Far pointer to caller-supplied data transfer buffer

Returns:CF = 0 if the operation succeeded, and:
AH = 0CF = 1 if the operation failed, and:
AH = error code, and:
Block_Count within the Disk_Address_Packet contains the number of sectors successfully transferred**Special:**

This function fails if the BIOS does not support Write w/Verify for the caller-specified device.

Related Functions:

None

Function 44h – Extended Verify Sectors**Supports:**

Fixed disks only

Description:

Used to verify zero or more blocks of data into host memory.

Passed:

AH = 44h

DL = Drive index

DS:SI = Far pointer to caller-supplied data transfer buffer

Returns:CF = 0 if the operation succeeded, and:
AH = 0CF = 1 if the operation failed, and:
AH = error code, and:
Block_Count within the Disk_Address_Packet contains the number of sectors successfully transferred**Special:**

This function fails if the BIOS does not support Write w/Verify for the caller-specified device.

Related Functions:

None

Non-Removable Media (Continued)**Function 47h – Extended Seek To Cylinder****Supports:**

Fixed disks only

Description:

Positions disk head at a specified sector/block.

Passed:

AH = 47h
 DL = Drive index
 DS:SI = Far point to caller supplied disk address packet

Returns:

CF = 0 if the function succeeded and:
 AH = 0
 CF = 1 if the function failed and:
 AH = error code

Related Functions:

None

Function 48h – Extended Get Drive Parameters**Supports:**

Fixed disks only

Description:

Returns physical characteristics of the caller-specified drive.

Passed:

AH = 48h
 DL = Logical drive number (80h = drive 0, 81h = drive 1, etc.)
 DS:SI = Far pointer to caller's buffer to receive drive information structure.

Returns:

CF = 0 if function succeeded
 CF = 1 if function failed
 DS:SI = Pointer to returned drive information structure as follows:

DRIVE_INFO STRUCT 1

info_size	dw	30	; Size of the DRIVE_INFO structure
info_flags	dw	?	; Information flag word (see Table 5-1)
cylinders	dw	?	; Number of drive cylinders
heads	dw	?	; Number of drive heads
sectors	dw	?	; Number of drive sectors per track
sectors	dq	?	; Total number of drive sectors
sector_size	dw	?	; Number of bytes per sector on drive

DRIVE_INFO ENDS**Special:**

The info_size field contains the size in bytes of the buffer passed in and returned, including this field. It must be set by the caller before invoking this function, indicating the size of the buffer passed in. It will also be filled in on return, indicating the size of the buffer returned. The returned size cannot be larger than the passed-in size. This mechanism allows the returned values of this function to be extended in the future with a minimum of compatibility problems. The flags field is used to return additional information about the disk characteristics. Currently bits [6:0] are defined. The others are reserved and must be set to zero by the implementor. Table 5-1 on page 41 gives the formats for bits [6:0] of the flags field.

Non-Removable Media (Continued)**Table 5-1. Flag Field**

Bit	Description
6	= 0 if CHS values are for the media that is currently in the drive (removable media only); values must be returned for the media if the media is in the drive. = 1 if CHS values are maximum supported by the drive (when there is no media in the drive)
5	= 0 if drive is not lockable. = 1 if drive is lockable. Bit 5 must be set for removable drives accessed via drive numbers 80h and above, as well as appropriate drives numbered 0-7Fh.
4	= 0 if drive does not have change line support. = 1 if drive has change line support. Bit 4 must be set for removable drives accessed via drive numbers 80h and above, as well as appropriate drives numbered 0-7Fh.
3	= 0 if drive does not support write with verify. = 1 if drive supports write with verify. Bit 3 indicates whether Write With Verify (4301h) is supported on this drive.
2	= 0 if drive is not removable = 1 if drive is removable
1	= 0 if Cylinder/Head/Sector-per-track info not supplied. = 1 if Cylinder/Head/Sector-per-track info is valid. Bit 1 is clear for block devices for which traditional geometry descriptions are inappropriate.
0	= 0 if DMA boundary errors may occur. = 1 if DMA boundary conditions are handled transparently by the BIOS.

Non-Removable Media (Continued)**5.3 INT 13H ATA DEVICE (FIXED DISK) DATA STRUCTURES**

The lower 1 MB of RAM in a PC/AT compatible system contains a hodgepodge of “compatible” locations and data structures. Several of these locations are related directly to fixed disk support.

5.3.1 BIOS Data Area (BDA)

Initially, the BIOS existed in ROM. It had no RAM in which to store information about the machine in which it was executing. For this reason, early BIOS designers chose to “steal” a 256-byte region in lower RAM for themselves. They named this region the BDA or BIOS Data Area.

The BDA resides between 40:0h and 40:FFh. It contains a variety of dissimilar items, ranging in function from the INT 09h scan code buffer, to serial/parallel port timeout values, to installed RAM and equipment descriptors.

The BDA is a 256 byte region starting at offset 0:400h. The BIOS creates and uses this area because the BIOS exists in ROM and has no dynamic RAM in its own address space in which to store dynamic information such as device status and configuration. Table 5-2 describes the layout of the BDA.

Table 5-2. Format of the BIOS Data Area

Offset	Size	Description
00h	WORD	Base I/O address of first serial I/O port, zero if none.
02h	WORD	Base I/O address of second serial I/O port, zero if none.
04h	WORD	Base I/O address of third serial I/O port, zero if none.
06h	WORD	Base I/O address of fourth serial I/O port, zero if none. Above fields filled in turn by POST as it finds serial ports. POST never leaves gaps. DOS and BIOS serial device numbers may be redefined by re-assigning these fields.
08h	WORD	Base I/O address of first parallel I/O port, zero if none.
0Ah	WORD	Base I/O address of second parallel I/O port, zero if none.
0Ch	WORD	Base I/O address of third parallel I/O port, zero if none.
0Eh	WORD	Segment of Extended BIOS Data Segment
10h	WORD	Installed hardware: Bits [15:14] = Number of parallel devices Bits [13:12] = Unused Bits [11:9] = Number of serial devices Bits [8:7] = Unused Bit 6 = Secondary floppy installed Bits [5:4] = Initial video mode: 00b = EGA,VGA 01b = 40 x 25 CGA 10b = 80 x 25 CGA 11b = 80 x 25 Monochrome Bit 3 = Unused Bit 2 = PS/2 pointing device installed Bit 1 = Coprocessor install Bit 0 = Primary (boot) floppy installed
12h	BYTE	POST scratchpad byte
13h	WORD	Base RAM (0-640K) size in KB
15h	BYTE	Reserved
16h	BYTE	Reserved

Non-Removable Media (Continued)**Table 5-2. Format of the BIOS Data Area (Continued)**

Offset	Size	Description
17h	BYTE	Keyboard Status byte (1 of 2): Bit 7 = INSert key active Bit 6 = CapsLock key active Bit 5 = NumLock key active Bit 4 = Scroll Lock key active Bit 3 = Either ALT key pressed Bit 2 = Either CTRL key pressed Bit 1 = Left Shift key pressed Bit 0 = Right Shift key pressed
18h	BYTE	Keyboard Status byte (2 of 2): Bit 7 = INSert key pressed Bit 6 = CapsLock key pressed Bit 5 = NumLock key pressed Bit 4 = Scroll Lock key pressed Bit 3 = Pause state is active Bit 2 = SysReq key pressed Bit 1 = Left ALT key pressed Bit 0 = Left CTRL key pressed
19h	BYTE	Keyboard Alt-<Number> Scratchpad: Used by the BIOS to build a three digit ALT-nnn ASCII code
1Ah	WORD	Keyboard: Offset of next character in the keyboard buffer that starts at 40:1Eh
1Ch	WORD	Keyboard: Offset of next empty location in the keyboard buffer that starts at 40:1Eh
1Eh	16 WORDs	16 Character keyboard buffer
3Eh	BYTE	Diskette recalibrate status: Bit 7 = Floppy controller IRQ occurred Bits [6:4] = Unused Bit 3 = 1 if diskette 3 requires recalibration Bit 2 = 1 if diskette 2 requires recalibration Bit 1 = 1 if diskette 1 requires recalibration Bit 0 = 1 if diskette 0 requires recalibration
3Fh	BYTE	Diskette Drive Motor Status: Bit 7 = 0 if current operation is verify or read Bit 7 = 1 if current operation is format or write Bit 6 = Unused Bits [5:4] = Currently selected diskette drive (0-3) Bit 3 = 1 if diskette 3 motor is on Bit 2 = 1 if diskette 2 motor is on Bit 1 = 1 if diskette 1 motor is on Bit 0 = 1 if diskette 0 motor is on
40h	BYTE	Number of 18.2 Hz timer ticks until floppy motor is shut off

Non-Removable Media (Continued)**Table 5-2. Format of the BIOS Data Area (Continued)**

Offset	Size	Description
41h	BYTE	Status of last diskette operation (0 = No Error): Bit 7 = 1 if drive not ready Bit 6 = 1 if a seek error occurred Bit 5 = 1 if a controller failure occurred Bits [4:0] = Other error conditions as shown here: 01h = Invalid command 02h = Sector address mark not found 03h = Disk write protect error 06h = Change line is active 08h = DMA overrun 09h = Attempt to DMA across 64K page boundary 0Ch = Unable to determine media type 0Eh = CRC error during read operation
42h ¹	7 BYTEs	PC/XT only disk subsystem command/status bytes
49h	BYTE	Current video mode
4Ah	WORD	Number of columns in current text page
4Ch	WORD	Size of current video page in bytes
4Eh	WORD	Current video page start address
50h	16 BYTEs	Row/column of cursor position in each of eight text video pages
60h	WORD	Video cursor type, 6845 compatible: MSB = Startline, LSB = Endline
62h	BYTE	Video current page number
63h	WORD	Video CRT controller base address: Color = 03D4h, Mono = 03B4h
65h	BYTE	Video current setting of mode select register 03D8h/03B8h
66h	BYTE	Video current setting of CGA palette register 03D9h
67h	DWORD	Real mode re-entry address for BIOS shutdown from protected mode sequences
6Bh	BYTE	Ordinal value of last spurious interrupt
6Ch	DWORD	Number of IRQ 0 timer ticks since 12:00AM
70h	BYTE	Date rollover flag: Non-zero if timer has overflowed (crossed midnight) since the last INT 1Ah GET_TIME call
71h	BYTE	Ctrl-Break status: Bit 7 = 1 if Ctrl-Break key has been pressed
72h	WORD	System reset signature: = 1234h during CTRL-ALT-DEL (warm) boot

Non-Removable Media (Continued)**Table 5-2. Format of the BIOS Data Area (Continued)**

Offset	Size	Description
74h ¹	BYTE	Status of last fixed disk operation: 00h = No error 01h = Invalid function requested 02h = Sector address mark not found 03h = Disk is write protected 04h = Sector not found 05h = Reset operation failed 07h = Drive parameters invalid 08h = DMA overrun 09h = Attempt to DMA across 64K page boundary 0Ah = Bad sector error 0Bh = Bad cylinder 0Dh = Invalid number of sectors requested 0Eh = Control data address mark found 0Fh = DMA arbitration error 10h = Unrecoverable CRC error 11h = ECC data correction occurred 20h = General controller failure 40h = Seek operation failed 80h = Drive timed out AAh = Drive not ready error BBh = Unknown error CCh = Error during write E0h = Unknown error FFh = Media sense failed
75h ¹	BYTE	Number of installed fixed disks
76h ¹	BYTE	Head control byte (= 8 if head count > 8)
77h	BYTE	Reserved
78h	3 BYTES	Timeout count bytes for LPT1-LPT3
79h	BYTE	Reserved
7Ch	4 BYTES	Timeout count bytes for COM1-COM3 devices.
80h	WORD	Keyboard buffer start location override (relative to 40:0h)
82h	WORD	Keyboard buffer end offset + 1 override (relative to 40:0h)
84h	BYTE	Maximum video text row
85h	WORD	Height of EGA/MCGA/VGA character in scan lines
87h	BYTE	Video EGA/VGA control byte
88h	BYTE	Video EGA/VGA switches
89h	BYTE	Reserved
8Ah	BYTE	Video [MCGA/VGA]: index into display code table
8Bh	BYTE	Diskette media control byte: Bits [7:6] = Data rate used during last operation. 00b = 500 KB/second 01b = 300 KB/second 10b = 250 KB/second 11 = Reserved Bits [5:4] = Diskette step rate used during last operation Bits [3:0] = Reserved
8Ch ¹	BYTE	Last BIOS read of Fixed Disk Status Register

Non-Removable Media (Continued)**Table 5-2. Format of the BIOS Data Area (Continued)**

Offset	Size	Description
8Dh ¹	BYTE	Last BIOS read of fixed disk error register
8Eh ¹	BYTE	INT 13h IRQ semaphore
8Fh	BYTE	Diskette controller information byte
90h	BYTE	Diskette drive 0 media state (used internally by INT 13h diskette drive state machine)
91h	BYTE	Diskette drive 1 media state (used internally by INT 13h diskette drive state machine)
92h	BYTE	Diskette drive 0 media state at start of operation
93h	BYTE	Diskette drive 1 media state at start of operation
94h	BYTE	Diskette drive 0 current track number
95h	BYTE	Diskette drive 1 current track number
96h	BYTE	Extended Keyboard Status byte (1 of 2): Bit 7 = 1 if a Read ID command is in progress Bit 6 = 1 if last byte read was first of 2 ID bytes Bit 5 = 1 force NumLock during Read ID command Bit 4 = 1 if enhanced PC/AT keyboard installed Bit 3 = 1 if right ALT key is pressed Bit 2 = 1 if right CTRL key is pressed Bit 1 = 1 if last byte read from KBC was E1h Bit 0 = 1 if last byte read from KBC was E0h
97h	BYTE	Extended Keyboard Status Byte (2 of 2): Bit 7 = 1 if a keyboard transmit error occurred Bit 6 = 1 if an LED update is in progress Bit 5 = 1 if the keyboard issued a RESEND Bit 4 = 1 if the keyboard issued an ACK Bit 3 = Reserved Bit 2 = Current status of CapsLock LED Bit 1 = Current status of NumLock LED Bit 0 = Current status of ScrollLock LED
98h	DWORD	Far pointer to user-specified elapsed time semaphore byte (see INT 15h function 83h, Wait For Event): Bit 7 = 1 if delay is elapsed
9Ch	DWORD	User-specified wait Timer2: [AT, PS exc Mod 30] user wait count in 972 ns increments
A0h-FFh	5Fh BYTES	Reserved for OEM-specific BIOS features
100h	BYTE	Print Screen Status byte

1. Reserved for use by the fixed disk subsystem.

5.3.2 Extended BIOS Data Area

Eventually, the 256 byte BDA proved to be too small and a second area, appropriately named the “Extended BIOS Data Area” or EBDA was created. The EBDA was originally intended for storing PS/2 pointing device data. Many current systems use this area for storing fixed disk information.

The EBDA is typically 1 KB in length. The EBDA may be larger, however it is always in multiples of 1 KB increments. The segment at which the EBDA is located is a WORD value stored at 40:0Eh. Typically this value is 9FC0h, or 1 KB below the end of lower 640 KB RAM. The BYTE located at EBDA:0h describes the length of this area in 1 KB increments.

Note: A value of zero in the WORD sized EBDA start segment at 40:0Eh does not mean the EBDA starts at physical location zero. It means the system does not use an EBDA.

There is no fixed format for EBDA usage other than for those locations reserved for the PS/2 pointing device.

Non-Removable Media (Continued)**5.4 INT 41H/46H VECTORS**

INT 41h and INT 46h are not interrupt vectors in the normal sense. Attempting to execute an INT 41h or INT 46h instruction will most likely crash the system immediately. These locations hold 32-bit segment:offset pointers to physical parameter tables for fixed disks 0 and 1, respectively. If the fixed disk count in location 40:75h is one, only the pointer in the INT 41h location (0:104h) is valid. If the value in 40:75h is two, both pointers are valid.

The INT 41h and INT 46h vectors are “well-behaved” low memory locations, meaning they should contain valid information on any PC/AT or compatible system. Each of these pointers references the same type of parameter table structure, which appears here in MASM-compatible form.

5.4.1 Standard Disk Drive Parameter Table

The following is the Standard IBM PC/AT style fixed disk parameter table.

```
;STRUCT FIXED_PARMS – THE ORIGINAL PC/AT FIXED DISK PARAMETER TABLE
```

```
FIXED_PARMS STRUCT 1
```

```
    FP_CYLS        WOR    ?           ; Maximum Cylinder
                                D
    FP_HEADS        BYTE    ?         ; Maximum Head
    FP_RES1          WOR    ?         ; Unused
                                D
    FP_WR_PRECMP     WOR    ?         ; Write Pre-Comp Start Cylinder
                                D
    FP_RES2          BYTE    ?         ; Unused
    FP_CTL           BYTE    ?         ; Extended Head Count Flag
    FP_RES3          BYTE    ?         ; Unused
    FP_RES4          BYTE    ?         ; Unused
    FP_RES5          BYTE    ?         ; Unused
    FP_LANDING       WOR    ?         ; Head Parking Cylinder
                                D
    FP_SPT           BYTE    ?         ; Sectors Per Track
    FP_RES6          BYTE    ?
```

```
FIXED_PARMS ENDS
```

The following is clarification of some of the fields in the Fixed Disk Parameter table shown above:

Write Pre-Comp — Early drives were formatted with the same number of sectors per cylinder and required the BIOS to compensate for the fact that the magnetic media near the edge of a disk is farther apart than the media located closer to the center of the disk. The Write Pre-Comp field tells the BIOS the exact cylinder at which to start compensating during write operations on older drives. Modern drives vary the number of sectors per track and automatically convert the caller’s CHS parameters to the correct physical byte on the disk and for this reason do not require the information in this field.

Landing Zone — Early drives required BIOS or application support to seek to the landing-zone cylinder, a cylinder designated as a safe place to leave the read/write heads if the drive was to be physically moved. Modern drives automatically park their read/write heads when power is removed and the information in this field is no longer needed.

Extended Head Count Flag — On really old machines, this flag signifies a drive that contains eight or more read/write heads. Although this bit is still supported, applications and utilities pay little or no attention to it any more.

5.4.2 Enhanced Disk Drive Parameter Table

The Enhanced Disk Drive (or EDD) Table is backwardly compatible with the standard parameter table described above. The preceding discussion (Section 5.4 “INT 41h/46h Vectors”) applies to this type of table as well as the standard IBM drive parameter table. Modern systems support the EDD table, which is a superset of IBM’s original table.

The EDD table format was introduced by Phoenix Technologies and is valuable for a few reasons. First, because it is checksummed, an EDD table allows applications to verify that the data being examined is really a disk parameter table and not just random garbage in the system’s ROM or RAM. Second, the EDD table provides the BIOS scratchpad space to record geometry translation information. Lastly, the format of the EDD table allows system software to determine if and what kind of translation is currently active based on the contents of its physical vs. logical cylinder, head, and sector counts.

The format of the EDD table appears here in MASM-compatible format:

Non-Removable Media (Continued)

; STRUCT EDD_PARMS – ENHANCED DISK DRIVE TABLE

EDD_PARMS STRUCT 1

LOGCYL	DW	?	; Maximum Logical Cylinder
LOGHEAD	DB	?	; Maximum Logical Head
SIGNATURE	DB	?	; Translation Active Signature
PHYSPT	DB	?	; Physical Sector/Track Count
WPRECOMP	DW	?	; Write Pre-Comp Cylinder
RSV2	DB	?	; Unused/Reserved
CTRLBYTE	DB	?	; Extended Head Count Flag
PHYCYLN	DW	?	; Maximum Physical Cylinder
PHYHEAD	DB	?	; Maximum Physical Head
LZONE	DW	?	; Head Park Cylinder
LOGSPT	DB	?	; Logical Sector/Track Count
CHKSUM	DB	?	; Checksum Byte

EDD_PARMS ENDS

The difference between “physical” and “logical” parameters is one of the sources of most confusion when learning about and interfacing with IDE devices.

The physical parameters that a drive reports are simply a reference point shared by the drive and the system software. It is hard to imagine a 2.5” portable fixed disk actually having fifteen individual read/write heads inside a housing that stands about a half inch tall, however, this is the number of “physical” heads that many such drives report.

Logical parameters are those parameters that the system BIOS reports to the operating system. They may or may not be the same as the physical parameters reported by the drive itself. If the logical parameters differ from the physical parameters, the BIOS has some kind of translation active. To understand why translation is important and often necessary, refer to Section “Large IDE Drive Support API Subset” on page 37, which describes the methods used by system firmware and drivers to support “Large” IDE drives.

5.4.3 Fixed Disk Parameter Table Extensions

In addition to determining the physical geometry of a fixed disk, it is often useful to determine other fixed disk configuration parameters such as base I/O addresses, IRQ assignments, etc. For this reason, many newer system BIOS implementations support Fixed Disk Parameter Table Extensions (FDPTs).

Within such a system, the FDPTs take much of the guesswork out of determining the physical geometry and resources assigned to the system’s fixed disks. If the system BIOS supports the INT 13h Functions Extensions as described by IBM, there exists for each fixed disk in the system a FDPT00 which the user can retrieve by issuing INT 13h Function 48h.

Note: The FDPTs are an enhancement introduced to IBM’s original INT 13h Function Extensions by Phoenix Technologies. In order to determine if the FDPTs support is present within the system, the caller should first issue INT 13h Function 41h as shown on Page 38.

The Fixed Disk Parameter Table Extension is a 16-byte table that adheres to the format described below. As mentioned, this table is available only if the Check Extensions Present function described here returns a major version of 20h or greater.

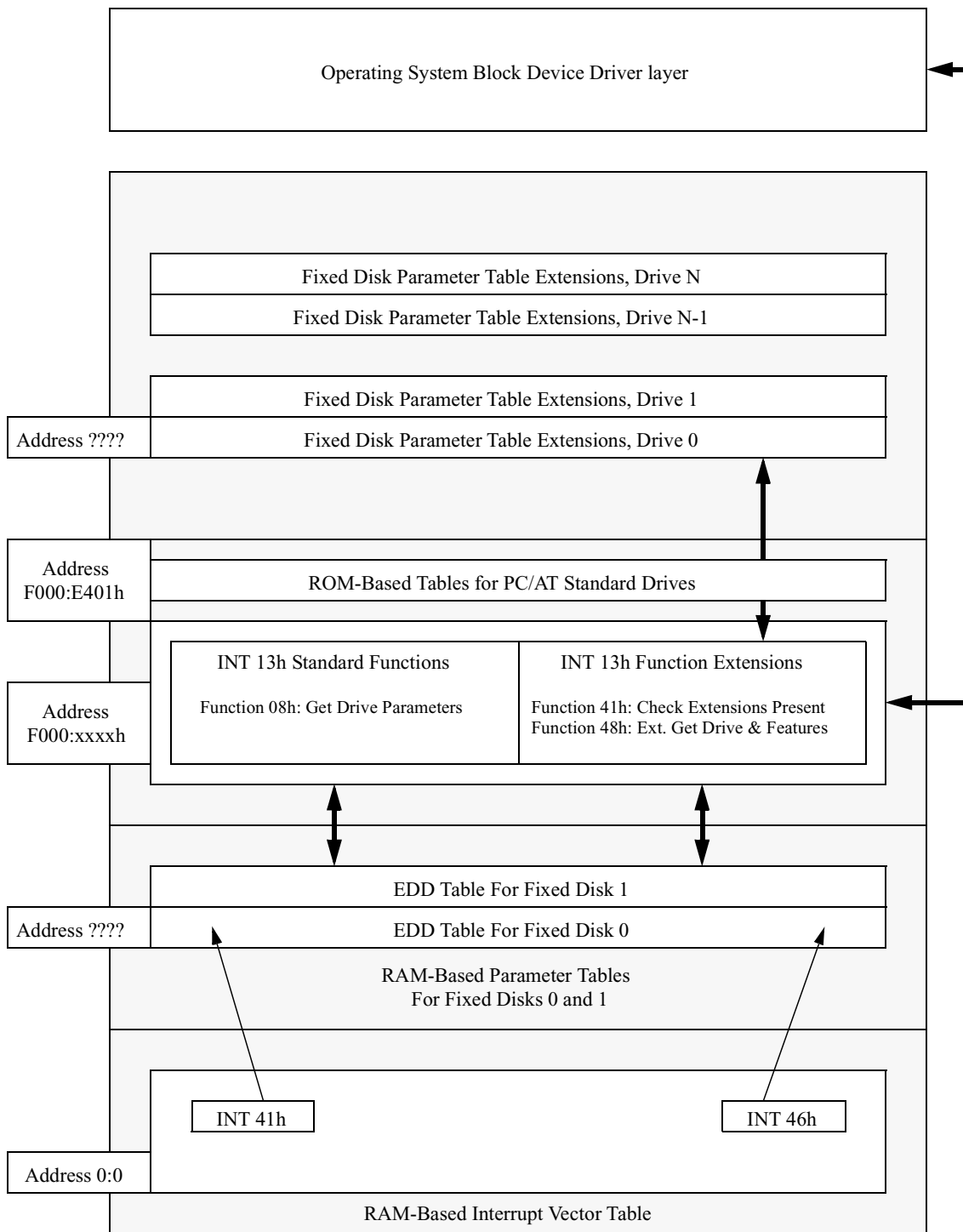
Non-Removable Media (Continued)**Table 5-3. Fixed Disk Parameter Table**

Offset	Size	Description
00h	WORD	Controller I/O Port Base Address (i.e., 1F0h): This field contains the I/O port base address for the drive. For example, if the drive is connected to the primary IDE controller in a PC/AT compatible system, this field contains the value 1F0h.
02h	WORD	Drive Control Port Address (i.e., 3F6h): This field contains the address of the control port for the drive. For example, if the drive is connected to the primary IDE controller in a PC/AT compatible system, this field contains the value 3F6h
04h	BYTE	Status Byte #1 — Drive Head Register Prefix: The drive head register prefix byte serves two purposes. First, it indicates to the caller of INT 13h Function 48h if the drive is in LBA mode, and whether the drive is a master or slave drive. Second, the BIOS stores the LBA and master/slave information in the same form as the upper four bits of the drive controller's head register. This simplifies the BIOS' programming of the drive's master/slave and LBA modes because during a disk read/write operation, the BIOS merely OR's this value with the desired drive head when it programs the head register. Bit 7 = Reserved: Must be set to 1. Bit 6 = If set, the drive is in LBA mode, otherwise the drive is CHS. Bit 5 = Reserved: Must be set to 1. Bit 3 = If set, the drive is an IDE master, otherwise the drive is a slave. Bits [3:0] = Reserved: Must be set to 0.
05h	BYTE	Reserved: Bits [7:4] are always 0.
06h	BYTE	Drive IRQ Assignment: This field contains the ordinal number of the IRQ assigned to the drive.
07h	BYTE	Multi-Sector Transfer Byte Count: This field contains the number of sectors the BIOS transfers during a read or write to the drive.
08h	BYTE	Drive DMA Assignment: Bits [7:4] = Specify the ATA-DMA mode in which the drive is currently operating. Bits [3:0] = Contain the ordinal number of the DMA channel currently assigned to the drive. If the drive does not support DMA, or is not currently in DMA mode, the value in this field is undefined.
09h	BYTE	Drive PIO Timings: This field contains the ordinal number of the Small Form Factor Committee-specified PIO mode in which the drive is currently operating. Legal values for this field are 1, 2, 3 and 4. The SFF Committee ATA-2 specification contains a comprehensive explanation of each PIO timing mode.
0Ah	BYTE	Drive Description Byte: Bit 7 = If set, the system has been configured to perform DWORD-wide transfers for this drive. Otherwise, the system performs WORD-wide transfers to and from the drive Bit 6 = If set, the drive is a CD-ROM device. Bit 5 = If set, the drive supports removable media Bit 4 = If set, the system has configured the drive to operate in LBA mode. Bit 3 = If set, the BIOS is currently configured to perform logical to physical translation of the drive's CHS parameters. Bit 2 = If set, the BIOS is configured to transfer data to and from the drive in multi-sector mode and the Multi-Sector Transfer Count Byte data is valid. Bit 1 = If set, the system is configured to transfer data to and from the drive via DMA and the DMA Assignment Byte data is valid. Bit 0 = If set, the system has configured the drive controller to operate in a fast PIO timing mode, and the PIO Mode Type Byte is valid.
0Bh	3 BYTES	Reserved: Must be set to 0.

Non-Removable Media (Continued)**Table 5-3. Fixed Disk Parameter Table**

Offset	Size	Description
0Eh	BYTE	Extension Table Revision ID (= 10h for this structure): This field contains the revision ID for the extension table. Currently, the only value defined for this field is 10h. FDPTs do not replace standard drive parameter tables; the information contained in each table is mutually exclusive. Instead, these tables allow the operating system to bypass its comparatively time-consuming and inaccurate process of snooping drive capabilities on systems that do not provide these extensions. Unlike the two standard drive parameter tables in a PC/AT-compatible BIOS, which are available to applications and operating system enumeration software via the INT 41h and INT 46h vectors, only the system BIOS knows the exact location of the system's fixed disk parameter table extensions. Instead of examining drive parameters via interrupt vectors, applications and operating systems that are capable of managing more than two drives invoke INT 13h Function 48h to receive a far pointer to the fixed disk parameter table extension for each of the system's fixed disks, regardless of whether the appropriate INT 41h and INT 46h vectors are currently available for the specified drive. In addition to verifying that the FDPTs are present, the calling application or device driver should retrieve all physical disk information via the extended INT 13h Function 48h (Extended Get Drive Parameters, see Page 40).
0Fh	BYTE	Checksum of this Extension Table

Figure 5-1 on page 51 shows the presence of all possible tables in a system that supports INT 41h, INT 46h, EDD, ROM-based and FDPT Extensions.

Non-Removable Media (Continued)**Figure 5-1. System BIOS INT 13h Interface with Enhanced IDE Data Structures**

6.0 PCI Support

This chapter describes the support that the XpressROM firmware provides for PCI device enumeration, configuration and run-time access.

PCI bus architecture is outlined in detail by the following documents:

- PCI Local Bus Specification, revision 2.1 or newer
- PCI BIOS Specification, revision 2.1 or newer
- PCI IDE Specification, revision 1.0
- PCI-PCI Bridge Specification, revision 1.1 or newer

Each of these documents is available through the PCI SIG (Special Interest Group) located at <http://www.pcisig.com>.

The XpressROM firmware supports all PCI 2.1 BIOS functions except those designed to support getting/setting IRQ routing options. This chapter assumes that the reader is familiar with the layout and usage of PCI configuration space.

Directory of 32-Bit Services

The XpressROM firmware provides a 32-bit services table, which is characterized by the presence of the ‘_32_’ ASCII signature on a 16-byte boundary in the address range E0000h-FFFF0h.

The purpose of this table is to inform the operating system of the presence and physical location of the 32-bit PCI BIOS services entry point. The layout of this table is described in the PCI BIOS Specification.

PCI IRQ Routing Support

Currently, IRQs are assigned to each device that requires one. All devices are assigned the same IRQ (typically 11) in the I/O companion device, and their INTLINE registers are programmed to reflect this assignment. Therefore, any operating systems that execute in conjunction with the XpressROM firmware must provide drivers that are capable of sharing IRQs.

Power-On PCI Device Configuration Support

During its power-on sequence, the XpressROM firmware assigns IRQs, I/O, and memory ranges to PCI adapters requesting such system resources. The XpressROM firmware scans the PCI bus in ascending order. I/O and memory ranges are assigned in a “top-down” fashion, starting at FFFFh and FFE0000h, respectively.

Assignment of PCI IRQ Channels

The same IRQ channel is assigned to all PCI devices and it is expected that the operating system can share IRQ channels between devices. In a basic implementation, INTA# is routed to IRQ11. Similarly, IRQ11 is programmed to be level sensitive in the I/O companion device’s ELCR (Edge/Level Configuration Register). In cases where the operating system does not support IRQ sharing, the XpressROM firmware can be built to steer PCI interrupt channels other than INTA# to different PC/AT IRQ channels.

Assignment of PCI I/O Ranges

PCI devices are assigned I/O ranges starting above the ISA limit of 0FFFh. Such a strategy ensures that there are no resource conflicts between PCI and legacy devices. The XpressROM firmware contains a simple resource allocation module that assigns I/O ranges in descending order as they are required by installed PCI devices.

Assignment of PCI Address Space Ranges

PCI devices are assigned address ranges starting above the ISA limit of 0FFFFFFh. Such a strategy ensures that there are no resource conflicts between PCI and legacy devices. Additionally, the XpressROM firmware ensures that there are no conflicts between those address ranges assigned to PCI devices and the ranges used by SMM (System Management Mode) software related functions. The XpressROM firmware contains a simple resource allocation module that assigns address ranges in descending order as they are required by installed PCI devices.

Support for PCI Option ROM.

The XpressROM firmware contains functionality to detect, shadow and initialize PCI option ROMs. As required by the PCI specification, PCI option ROMs are left writable during their initialization sequence to allow such ROMs to modify their own code space.

Support for PCI Bridges.

The XpressROM firmware does not contain PCI Bridge Support except to avoid corrupting a PCI Bridge device.

PCI Support (Continued)**6.1 INT 1Ah/FUNCTION B1h PCI BIOS SUBFUNCTIONS DESCRIPTIONS**

The XpressROM firmware supports real (INT 1Ah) and 32-bit protected mode access to the set of functions defined by revision 2.1 of the document, “PCI BIOS Specification.” This section lists those INT 16h/Function B1h subfunctions that are supported in the XpressROM firmware, the parameters that system software supplies to each subfunction and the values that each function returns.

Subfunction 01h – PCI_BIOS_PRESENT**Description:**

The primary purpose of this function is to signal the presence and revision level of the system’s PCI BIOS. Additionally, PCI_BIOS_PRESENT reports the actual hardware mechanism that the PCI BIOS uses to read and write configuration space registers, as well as what type of special cycle support exists in the chipset.

Passed:

AH = B1h (PCI_FUNCTION_ID)
AL = 01h (PCI_BIOS_PRESENT)

Returns:

EDX = 20494350h (4-byte ASCII string ‘PCI ’) if PCI BIOS present
AH = If 0, and CF and EDX are correct, PCI BIOS is present
AL = Supported hardware mechanism
BH = BCD-encoded major revision of interface supported
BL = BCD-encoded minor revision of interface supported
CL = Zero-based number of last PCI bus in the system
CF = 0 if PCI BIOS is present, and AH and EDX are correct
1 if PCI BIOS is not present

If the PCI_BIOS_PRESENT function returns successfully, the AL register contains I/O companion device-specific information regarding special cycle support and the programming mechanism that the system’s I/O companion supports.

Table 6-1. The AL Register Upon Return From A Successful Call to PCI_BIOS_PRESENT

Bit	Description
7:6	Reserved
5	1: Special cycles are supported via Mechanism #2
4	1: Special cycles are supported via Mechanism #1
3:2	Reserved
1	1: System chipset supports Mechanism #1
0	1: System chipset supports Mechanism #2

PCI Support (Continued)**Subfunction 02h – FIND_PCI_DEVICE****Description:**

Given a Vendor ID, Device ID, and an Index N, the function FIND_PCI_DEVICE finds the Nth device or function whose Vendor ID and Device ID match those passed into the function. Additionally, FIND_PCI_DEVICE will return the Bus Number, Device Number, and function number of the device if it successfully finds a match.

This function is useful for locating all PCI devices in the system that perform the same function. The first return value of DEVICE_NOT_FOUND indicates that there are no more devices of the type specified in the system.

Passed:

AH = B1h (PCI_FUNCTION_ID)
 AL = 02h (PCI_FIND_DEVICE)
 CX = 16-bit Device ID
 DX = 16-bit Vendor ID
 SI = Index N (Nth occurrence of PCI device)

Returns:

BH = Bus Number of device (0-based)
 BL = Bits [7:3] = Device number
 Bits [2:0] = Function number within located device
 AH = Function return status code
 SUCCESSFUL
 DEVICE_NOT_FOUND
 BAD_VENDOR_ID
 CF = Function completion status
 0 if device was located
 1 if an error occurred, AH contains error code

Subfunction 03h – FIND_PCI_CLASS_CODE**Description:**

The function FIND_PCI_CLASS_CODE returns the Nth occurrence of the device or device function whose class code matches the value passed in the ECX register.

The FIND_PCI_CLASS_CODE function is useful for locating all system PCI devices that belong to the same device class. The first return value of DEVICE_NOT_FOUND indicates that there are no more devices in the system with the specified class code.

Passed:

AH = B1h (PCI_FUNCTION_ID)
 AL = 03h (FIND_PCI_CLASS_CODE)
 ECX = Class code in lower three bytes
 SI = Index N (Nth occurrence of PCI device or function whose class code matches that passed in the ECX register)

Returns:

BH = Bus Number of device (0-based)
 BL = Bits [7:3] = Device number
 Bits [2:0] = Function number within located device
 AH = Function return status code
 SUCCESSFUL
 DEVICE_NOT_FOUND
 CF = Function completion status
 0 if device was located
 1 if an error occurred, AH contains error code

PCI Support (Continued)**Subfunction 06h – GENERATE_SPECIAL_CYCLE****Description:**

The PCI host controller has the ability to broadcast messages across the entire bus by issuing special cycles. When a special cycle occurs, devices on the bus may monitor the message, but must not acknowledge the message by asserting their DEVSEL# (Device Select) signal. PCI devices do not have to respond to special cycles. If a device is capable of responding to special cycles, the system configurator or PCI BIOS enables this capability in a device by setting bit three of its Configuration Space Command Register to a '1'.

Passed:

AH = B1h (PCI_FUNCTION_ID)
 AL = 06h (GENERATE_SPECIAL_CYCLE)
 BH = Number of bus on which to assert the cycle (0-FFh)
 EDX = Special Cycle Data
 Bits [15:0] = Special cycle message
 Bits [31:16] = Message specific data

Returns:

AH = Function return status code
 SUCCESSFUL
 FUNC_NOT_SUPPORTED
 CF = Function completion status
 0 if function completed successfully
 1 if an error occurred, AH contains error code

Special cycle messages consist of a 16-bit message field, which the host controller asserts on address/data lines AD[15:0], and a message field which is asserted in signals AD[31:16]. The meaning of each message field is determined by the PCI SIG and a list of all valid special cycles appears in the PCI Local Bus Specification. Revisions 2.0 and 2.1 of the PCI Local Bus Specification define the following special cycle messages:

- 0000h: System Shutdown
- 0001h: PCI Bus Halt
- 0002h: Intel x86 Architecture-Specific message
- 0003h-FFFFh: Reserved

One useful application of special cycle messages is to broadcast a pending system shutdown to PCI bus devices. For example, a PCI network controller should immediately abort its current network activity in the event of a system shutdown. It is the responsibility of the operating system to issue a PCI BIOS call to generate the system shutdown message (0000h).

PCI Support (Continued)

6.1.1 Reading and Writing Configuration Space

The PCI BIOS provides applications and drivers for six subfunctions for reading and writing configuration space registers:

- Subfunction 08h: READ_CONFIG_BYTE
- Subfunction 09h: READ_CONFIG_WORD
- Subfunction 0Ah: READ_CONFIG_DWORD
- Subfunction 0Bh: WRITE_CONFIG_BYTE
- Subfunction 0Ch: WRITE_CONFIG_WORD
- Subfunction 0Dh: WRITE_CONFIG_DWORD

When reading or writing a DWORD-sized value in configuration space, the register number specified in the function WRITE_CONFIG_DWORD or READ_CONFIG_DWORD must be on a DWORD boundary, such as 00h, 04h, 08h, etc. Similarly, when reading or writing a WORD-sized value in configuration space, the register number specified in the function WRITE_CONFIG_WORD or READ_CONFIG_WORD must be on a WORD boundary, such as 00h, 02h, 04h, etc. The remaining functions, READ_CONFIG_BYTE and WRITE_CONFIG_BYTE operate correctly on any register boundary within configuration space.

If the calling program attempts an illegal I/O operation, such as a DWORD read of configuration space location 02h, the PCI BIOS returns the value 87h (BAD_REGISTER_NUMBER) in the AH register and sets the carry flag.

Subfunction 08h – READ_CONFIG_BYTE

Description

This subfunction reads one BYTE from the configuration space of the caller-specified PCI device.

Passed:

- AH = B1h (PCI_FUNCTION_ID)
- AL = 08h (READ_CONFIG_BYTE)
- BH = Number of bus on which the device resides
- BL = Bits [7:3] = Device number
Bits [2:0] = Device function number
- DI = Register index (0-FFh) within the device's or function's configuration space

Returns:

- AH = Function return status code
SUCCESSFUL
- CL = Byte read from configuration space
- CF = Function completion status
0 if function completed successfully
1 if an error occurred

Subfunction 09h – READ_CONFIG_WORD

Description:

This subfunction reads one WORD from the configuration space of the caller-specified PCI device.

Passed:

- AH = B1h (PCI_FUNCTION_ID)
- AL = 09h (READ_CONFIG_WORD)
- BH = Number of bus on which the device resides
- BL = Bits [7:3] = Device number
Bits [2:0] = Device function number
- DI = Register index (0-FFh) within the device's or function's configuration space

Returns:

- AH = Function return status code
SUCCESSFUL
BAD_REGISTER_NUMBER
- CX = WORD read from configuration space
- CF = Function completion status
0 if function completed successfully
1 if an error occurred, AH contains error code

PCI Support (Continued)**Subfunction 0Ah – READ_CONFIG_DWORD****Description:**

This subfunction reads one DWORD from the configuration space of the caller-specified PCI device.

Passed:

AH = B1h (PCI_FUNCTION_ID)
 AL = 0Ah (READ_CONFIG_DWORD)
 BH = Number of bus on which the device resides
 BL = Bits [7:3] = Device number
 Bits [2:0] = Device function number
 DI = Register index (0-FFh) within the device's or function's configuration space

Returns:

AH = Function return status code
 SUCCESSFUL
 BAD_REGISTER_NUMBER
 ECX = DWORD read from configuration space
 CF = Function completion status
 0 if function completed successfully
 1 if an error occurred, AH contains error code

Subfunction 0Bh – WRITE_CONFIG_BYTE**Description:**

This subfunction writes one BYTE to the configuration space of the caller-specified PCI device.

Passed:

AH = B1h (PCI_FUNCTION_ID)
 AL = 0Bh (WRITE_CONFIG_BYTE)
 BH = Number of bus on which the device resides
 BL = Bits [7:3] = Device number
 Bits [2:0] = Device function number
 CL = Byte value to write to configuration space
 DI = Register index (0-FFh) within the device's or function's configuration space

Returns:

AH = Function return status code
 SUCCESSFUL
 CF = Function completion status
 0 if function completed successfully
 1 if an error occurred, AH contains error code

Subfunction 0Ch – WRITE_CONFIG_WORD**Description:**

This subfunction writes one WORD to the configuration space of the caller-specified PCI device.

Passed:

AH = B1h (PCI_FUNCTION_ID)
 AL = 0Ch (WRITE_CONFIG_WORD)
 BH = Number of bus on which the device resides
 BL = Bits [7:3] = Device number
 Bits [2:0] = Device function number
 CX = WORD value to write to configuration space
 DI = Register index (0-FFh) within the device's or function's configuration space

Returns:

AH = Function return status code
 SUCCESSFUL
 BAD_REGISTER_NUMBER
 CF = Function completion status
 0 if function completed successfully
 1 if an error occurred

PCI Support (Continued)

Subfunction 0Dh – WRITE_CONFIG_DWORD

Description:

This subfunction writes one DWORD to the configuration space of the caller-specified PCI device.

Passed:

AH = B1h (PCI_FUNCTION_ID)
AL = 0Dh (WRITE_CONFIG_DWORD)
BH = Number of bus on which the device resides
BL = Bits [7:3] = Device number
 Bits [2:0] = Device function number
DI = Register index (0-FFh) within the device's or function's configuration space
ECX = DWORD value to write to configuration space

Returns:

AH = Function return status code
 SUCCESSFUL
 BAD_REGISTER_NUMBER
CF = Function completion status
 0 if function completed successfully
 1 if an error occurred, AH contains error code

7.0 Serial Port BIOS Support

The XpressROM firmware supports the following standard INT 14h functions pertaining to the operation of up to four serial ports in the system.

- Function 00h: Initialize Port
- Function 01h: Write Character to Port
- Function 02h: Read Character from Port
- Function 03h: Get Port Status

7.1 INT 14H SERIAL PORT BIOS FUNCTIONS DESCRIPTIONS

This section lists those INT 14h functions that are supported in the XpressROM firmware, the parameters that system software supplies to each INT 14h function, and the values that each function returns. The I/O address(es) of any port(s) in the system are stored in the BDA by POST. The WORD-length addresses begin at 40:0 and can go through 40:6.

Function 00h – Initialize Port

Description:

Sets up the serial port for transmission or reception of data.

Passed:

AH = 00h

AL = Port parameters

7-5 Data rate:

Value: Rate

000 = 110 bps

001 = 150 bps

010 = 300 bps

011 = 600 bps

100 = 1200 bps

101 = 2400

110 = 4800

111 = 9600

4-3 Parity

Value: Parity

00 = None

10 = None

01 = Odd

11 = Even

2 Stop bits

Value: Number of Stop Bits

00 = None

10 = None

01 = Odd

11 = Even

1-0 Data size

Value: Character Length

00 = 5

01 = 6

10 = 7

11 = 8

DX = Port number

Value: BDA Location

0 = 40:0

1 = 40:2

2 = 40:4

3 = 40:6

Serial Port BIOS Support (Continued)**Returns:**

- AH = Line status
- 7 = Timeout error occurred
 - 6 = Transfer shift register empty
 - 5 = Transfer holding register empty
 - 4 = Break occurred
 - 3 = Framing error occurred
 - 2 = Parity error occurred
 - 1 = Overrun error occurred
 - 0 = Data ready
- AL = Modem status
- 7 = Carrier detect
 - 6 = Ring indicator
 - 5 = Data set ready
 - 4 = Clear to send
 - 3 = Delta carrier detect
 - 2 = Trailing edge of ring indicator
 - 1 = Delta data set ready
 - 0 = Delta clear to send

Function 01h – Write Character to Port**Description:**

Sends a character of data to the serial port.

Passed:

- AH = 01h
- AL = ASCII code for the character
- DX = Port number (see description in function 00)

Returns:

- AH = 00h if no error occurred
- 8xh indicates an error occurred
- x represents:
- 6 = Transfer shift register empty
 - 5 = Transfer holding register empty
 - 4 = Break occurred
 - 3 = Framing error occurred
 - 2 = Parity error occurred
 - 1 = Overrun error occurred
 - 0 = Data ready

Function 02h – Read Character from Port**Description:**

Reads a byte of data from the serial port. Function will not return until data is available or a timeout has occurred. The caller may invoke function 03 (Get Port Status) to check whether data is available to be read.

Passed:

- AX = 02
- DX = Port number (see description in function 00)

Returns:

- AL = ASCII code for the character
- AH = 00h if no error occurred
- 8xh indicates an error occurred
- x represents:
- 6 = Transfer shift register empty
 - 5 = Transfer holding register empty
 - 4 = Break occurred
 - 3 = Framing error occurred
 - 2 = Parity error occurred
 - 1 = Overrun error occurred
 - 0 = Data ready

Serial Port BIOS Support (Continued)**Function 03h – Get Port Status****Description:**

Reports the status of the serial port. Most of the function reports errors, however it can be used to check whether data is available for reading (Data Ready flag, AH = 0).

Passed:

AX = 03
 DX = Port number (see description in function 00)

Returns:

AH = Encoded line status
 7 = Timeout error occurred
 6 = Transfer shift register empty
 5 = Transfer holding register empty
 4 = Break occurred
 3 = Framing error occurred
 2 = Parity error occurred
 1 = Overrun error occurred
 0 = Data ready

AL = Encoded modem status
 7 = Carrier detect
 6 = Ring indicator
 5 = Data set ready
 4 = Clear to send
 3 = Delta carrier detect
 2 = Trailing edge of ring indicator
 1 = Delta data set ready
 0 = Delta clear to send

8.0 Parallel Port BIOS Support

The XpressROM firmware supports the following standard INT 17h functions pertaining to the operation of up to three parallel ports in the system.

- Function 00h: Print a Character
- Function 01h: Initialize Parallel Port
- Function 02h: Get Port Status

8.1 INT 17H PARALLEL PORT BIOS FUNCTIONS DESCRIPTIONS

This section lists those INT 17h functions that are supported in the XpressROM firmware, the parameters that system software supplies to each INT 17h function and the values that each function returns. The I/O address(es) of any port(s) in the system are stored in the BDA by POST. The WORD-length addresses begin at 40:8 and can go through 40:C.

Function 00h – Print a Character

Description:

Sends a character to the parallel port. The function waits until the printer is not busy or until a timeout has occurred. The caller can invoke Function 02h to check whether the printer is ready to accept a character.

Passed:

AH = 00h
 AL = ASCII code of character to send
 DX = Port number
 Value BDA Location
 0 = 40:8
 1 = 40:A
 2 = 40:C

Returns:

AH = Port status
 7 = Printer ready
 6 = Acknowledgement of last character
 5 = Paper empty
 4 = Printer selected
 3 = I/O error
 2-1 = Reserved
 0 = Timeout error

Function 01h – Initialize Parallel Port

Description:

Resets the printer and sets the top of page to the current position of the paper by sending control codes 08h and 0Ch.

Passed:

AH = 01h
 DX = Port number (see description in Function 00h)

Returns:

AH = Encoded information indicating status of the operation.
 7 = Printer ready
 6 = Acknowledgement of last character
 5 = Paper empty
 4 = Printer selected
 3 = I/O Error
 2-1 = Reserved
 0 = Timeout error

Parallel Port BIOS Support (Continued)

Function 02h – Get Port Status

Description:

Reads the current status of the printer. This function can be used to determine if the printer is ready to accept data.

Passed:

AX = 02h

DX = Port number (see description in Function 00h)

Returns:

AH = Encoded information indicating status of the port.

7 = Printer ready

6 = Acknowledgement of last character

5 = Paper empty

4 = Printer selected

3 = I/O error

2-1 = Reserved

0 = Timeout error

9.0 Human Interface Device Support

This chapter describes the XpressROM firmware's support for human interface devices (HID), such as keyboards and pointing devices.

The XpressROM firmware provides full support for 101 key PC keyboards. It contains a full implementation of the INT 16h interface as described in numerous books on the subject of low level PC programming. Additionally, the XpressROM firmware contains an IRQ1 handler and supports INT 15h Function 4Fh (Scan Code Redirect) as a means of enabling TSRs (Terminate, Stay Resident) and any other software that intercepts keystrokes.

In addition to the standard PS/2 keyboard device the INT 16h interface transparently supports the Legacy USB Keyboard.

The INT 16h HID functions:

- Function 00h: Wait For Character
- Function 01h: Check For Key Present
- Function 02h: Get Shift Status
- Function 05h: Insert Key Code In Keyboard Buffer
- Function 10h: Get Extended Key Code
- Function 11h: Check For Enhanced Key Code
- Function 12h: Get Extended Shift Status

9.1 INT 16H FUNCTIONS DESCRIPTIONS

This section lists those INT 16h functions that are supported in the XpressROM firmware, the parameters that system software supplies to each INT 16h function and the values that each function returns.

Function 00h – Wait For Character

Description:

Returns after a keystroke occurs, ignoring any non-standard keystroke.

Passed:

AH = 00h

Returns:

AH = Scan code

AL = Character

Related Functions:

INT 16h “Function 10h – Get Extended Key Code” on page 65

Function 01h – Check For Key Present

Description:

Checks the keyboard buffer to determine whether a keystroke is available. The keystroke is not removed from the keyboard buffer.

Passed:

AH = 01h

Returns:

AH = Scan code

AL = Character

ZF = 0 if character in buffer
1 if no character in buffer

Related Functions:

INT 16h “Function 11h – Check For Enhanced Key Code” on page 66

Human Interface Device Support (Continued)**Function 02h – Get Shift Status****Description:**

Returns the state of the Left Shift, Right Shift, Ctrl, and Alt keys. It also indicates the state of the ScrollLock, NumLock, CapsLock, and Insert keys.

Passed:

AH = 02h

Returns:

AL = Shift status bits
 Bit 7 = Insert state active
 Bit 6 = CapsLock active
 Bit 5 = NumLock active
 Bit 4 = ScrollLock active
 Bit 3 = Alt depressed
 Bit 2 = Ctrl depressed
 Bit 1 = Left Shift key depressed
 Bit 0 = Right Shift key depressed

Related Functions:

INT 16h “Function 12h – Get Extended Shift Status” on page 66

Function 05h – Insert Key Code In Keyboard Buffer**Description:**

Adds a keystroke to the keyboard buffer and adjusts the pointer appropriately.

Passed:

AH = 05h
 CH = Scan code
 CL = Character

Returns:

AL = 01h if buffer full

Related Functions:

None

Function 10h – Get Extended Key Code**Description:**

Returns after a keystroke (including non-standard keys) occurs.

Passed:

AH = 10h

Returns:

AH = Scan code
 AL = Character

Related Functions:

INT 16h “Function 00h – Wait For Character” on page 64

Human Interface Device Support (Continued)**Function 11h – Check For Enhanced Key Code****Description:**

Checks the keyboard buffer to determine whether a keystroke is available. The keystroke is not removed from the keyboard buffer. This functions does not ignore extended keystrokes.

Passed:

AH = 11h

Returns:

AH = Scan code \ meaningless if ZF = 1
 AL = Character /
 ZF = 0 if keystroke available
 1 if keyboard scan code buffer empty

Related Functions:

INT 16h “Function 01h – Check For Key Present” on page 64

Function 12h – Get Extended Shift Status**Description:**

Returns the state of the Left Shift, Right Shift, Ctrl, and Alt keys. It also indicates the state of the ScrollLock, NumLock, CapsLock, and Insert keys. Additionally, the states of other keys are returned.

Passed:

AH = 12h

Returns:

AL = Shift flags (same as for AH = 02h)
 Bit 7 = Ins ON
 Bit 6 = CapsLock ON
 Bit 5 = NumLock ON
 Bit 4 = ScrollLock ON
 Bit 3 = Either ALT key down
 Bit 2 = Either CTRL key down
 Bit 1 = Left Shift key down
 Bit 0 = Right Shift key down

 AH = Additional Shift Flag Information
 Bit 7 = SysReq key down
 Bit 6 = CapsLock key down
 Bit 5 = NumLock key down
 Bit 4 = ScrollLock key down
 Bit 3 = Right Alt key down
 Bit 2 = Right Ctrl key down
 Bit 1 = Left Alt key down
 Bit 0 = Left Ctrl key down

Related Functions:

INT 16h “Function 02h – Get Shift Status” on page 65

9.2 PS/2 POINTING DEVICE SUPPORT

None

10.0 System Time and Date

The XpressROM firmware supports a majority of the standard system functions.

INT 1Ah time and date functions:

- Function 00h: Get System Time Status
- Function 01h: Set System Time
- Function 02h: Get Real-Time Clock Time Status
- Function 03h: Set Real-Time Clock Time
- Function 04h: Get Real-Time Clock Date Status
- Function 05h: Set Real-Time Clock Date
- Function 06h: Set Real-Time Clock Alarm
- Function 07h: Cancel Real-Time Clock Alarm
- Function 09h: Get Real-Time Clock Alarm Status
- Function 0Ah: Read System Day Counter
- Function 0Bh: Set System Day Counter
- Function 0Eh: Get Real-Time Clock Date/Time Alarm and Status
- Function 0Fh: Initialize Real-Time Clock

10.1 INT 1AH TIME AND DATE FUNCTIONS DESCRIPTIONS

This section lists those INT 1Ah functions that are supported in the XpressROM firmware, the parameters that system software supplies to each INT 1Ah function and the values that each function returns.

Function 00h – Get System Time Status

Description:

Returns the 4-byte count of the number of timer ticks that have occurred since midnight. The value is maintained in the BIOS Data Area (40:6C) (Refer to Section 5.3.1 “BIOS Data Area (BDA)” on page 42 for information on BDA.) It also checks whether the rollover byte has been set (stored in location 40:70) and returns appropriately.

Passed:

AH = 00h

Returns:

CX = The most significant two bytes of the 4-byte count
 DX = The least significant two bytes of the 4-byte count
 AL = Rollover value, nonzero if midnight has passed since the last time the count was read

Related Functions:

INT 1Ah “Function 01h – Set System Time” on page 67
 INT 1Ah “Function 02h – Get Real-Time Clock Time Status” on page 68

Function 01h – Set System Time

Description:

Sets the 4-byte count of the number of timer ticks that have occurred since midnight. The value is maintained in the BIOS Data Area (40:6C). It does not modify the rollover byte, stored at 40:70.

Passed:

AH = 01h
 CX = The most significant two bytes of the four-byte count
 DX = The least significant two bytes of the four-byte count

Returns:

None

Related Functions:

INT 1Ah “Function 03h – Set Real-Time Clock Time” on page 68

System Time and Date (Continued)**Function 02h – Get Real-Time Clock Time Status****Description:**

This function returns the time value stored in the system's real-time clock, in the form of hours, minutes, and seconds.

Passed:

AH = 02h

Returns:

CH = The hour of the current time, in BCD, stored in the RTC
 CL = The minutes of the current time, in BCD, stored in the RTC
 DH = The seconds of the current time, in BCD, stored in the RTC
 DL = If 0, the RTC ignores daylight savings time
 If 1, the RTC adjusts for daylight savings time

Related Functions:

INT 1Ah "Function 01h – Set System Time" on page 67
 INT 1Ah "Function 03h – Set Real-Time Clock Time" on page 68
 INT 1Ah "Function 04h – Get Real-Time Clock Date Status" on page 68

Function 03h – Set Real-Time Clock Time**Description:**

This function sets the time value stored in the system's real-time clock, in the form of hours, minutes, and seconds.

Passed:

AH = 03h
 CH = The hour of the current time, in BCD, stored in the RTC
 CL = The minutes of the current time, in BCD, stored in the RTC
 DH = The seconds of the current time, in BCD, stored in the RTC
 DL = Set to:
 0 to ignore daylight savings time
 1 if the RTC should adjust for daylight savings time

Returns:

None

Related Functions:

INT 1Ah "Function 01h – Set System Time" on page 67
 INT 1Ah "Function 05h – Set Real-Time Clock Date" on page 69

Function 04h – Get Real-Time Clock Date Status**Description:**

This function returns the date that is stored in the system's real-time clock.

Passed:

AH = 04h

Returns:

CH = The century of the current date, in BCD, stored in the RTC
 CL = The year of the current date, in BCD, stored in the RTC
 DH = The month of the current date, in BCD, stored in the RTC
 DL = The day of the current date, in BCD, stored in the RTC

Related Functions:

INT 1Ah "Function 02h – Get Real-Time Clock Time Status" on page 68
 INT 1Ah "Function 05h – Set Real-Time Clock Date" on page 69

System Time and Date (Continued)**Function 05h – Set Real-Time Clock Date****Description:**

This function sets the date that is stored in the system's real-time clock.

Passed:

AH = 05h
 CH = The century of the current date, in BCD, stored in the RTC
 CL = The year of the current date, in BCD, stored in the RTC
 DH = The month of the current date, in BCD, stored in the RTC
 DL = The day of the current date, in BCD, stored in the RTC

Returns:

None

Related Functions:

INT 1Ah "Function 04h – Get Real-Time Clock Date Status" on page 68

Function 06h – Set Real-Time Clock Alarm**Description:**

This function sets the alarm in the system's real-time clock. The alarm will occur every 24 hours, causing an INT 4Ah, until it is disabled.

Passed:

AH = 06h
 CH = The hour for the alarm time, in BCD
 CL = The minutes for the alarm time, in BCD
 DH = The seconds for the alarm time, in BCD

Returns:

None

Related Functions:

INT 1Ah "Function 07h – Cancel Real-Time Clock Alarm" on page 69

Function 07h – Cancel Real-Time Clock Alarm**Description:**

This function disables the alarm in the system's real-time clock.

Passed:

AH = 07h

Returns:

None

Related Functions:

INT 1Ah "Function 06h – Set Real-Time Clock Alarm" on page 69

System Time and Date (Continued)**Function 09h – Get Real-Time Clock Alarm Status****Description:**

This function sets the alarm in the system's real-time clock. The alarm will occur every 24 hours, causing an INT 4Ah, until it is disabled.

Passed:

AH = 09h

Returns:

CH = The hour for the alarm time, in BCD
 CL = The minutes for the alarm time, in BCD
 DH = The seconds for the alarm time, in BCD
 DL = Alarm Status:
 00h = Alarm not enabled
 01h = Alarm enabled, but will not power up system
 02h = Alarm will power up system

Related Functions:

None

Function 0Ah – Read System Day Counter**Description:**

This function returns the number of days since January 1, 1980, from the value stored in the BDA (40:CE).

Passed:

AH = 0Ah

Returns:

CX = Number of days since 1/1/1980

Related Functions:

INT 1Ah "Function 04h – Get Real-Time Clock Date Status" on page 68

Function 0Bh – Set System Day Counter**Description:**

This function returns the number of days since January 1, 1980, from the value stored in the BDA (40:CE).

Passed:

AH = 0Bh
 CX = Number of days since 1/1/1980

Returns:

CF = 0 if successful
 1 if error

Related functions:

INT 1Ah "Function 05h – Set Real-Time Clock Date" on page 69
 INT 1Ah "Function 0Ah – Read System Day Counter" on page 70

System Time and Date (Continued)**Function 0Eh – Get Real-Time Clock Date/Time Alarm and Status****Description:**

This function returns settings for the RTC Alarm.

Passed:

AH = 0Eh

Returns:

BH = Alarm Status
 00h = Alarm not enabled
 01h = Alarm enabled, but will not power up system
 02h = Alarm will power up system
 CH = The hour for the alarm time, in BCD
 CL = The minutes for the alarm time, in BCD
 DH = The seconds for the alarm time, in BCD
 DL = The day of the month for the alarm, in BCD

Related Functions:

None

Function 0Fh – Initialize Real-Time Clock**Description:**

This function causes the BIOS to update its century byte.

Passed:

AH = 0Fh

Returns:

None

Related Functions:

None

11.0 System Services

The XpressROM firmware supports INT 15h for certain standard system services. This section describes the interfaces for those services. INT 15h is also used for Advanced Power Management (APM). See Section 15.0 "Power Management" on page 132 for details. Additionally, INT 15h is used for backward compatibility with National Semiconductor's Virtual System Architecture (VSA) technology. Details of that interface can be found in the "Virtual System Architecture BIOS Porting Guide".

INT 15h System Services Support Functions:

- Function 24h: A20 Support
 - Subfunction 00h: Disable A20 Gate
 - Subfunction 01h: Enable A20 Gate
 - Subfunction 02h: Get A20 Gate Status
- Function 87h: Extended Memory Block Move
- Function 88h: Get Extended Memory Size
- Function 89h: Enter Protected Mode
- Function BEh: NSC OEM Functions
 - Subfunction 00h: Read I²C Byte
 - Subfunction 01h: Write I²C Byte
 - Subfunction 02h: Write I²C Block
 - Subfunction 03h: Read NVRAM Data
 - Subfunction 04h: Write NVRAM Data
 - Subfunction 05h: Get Default NVRAM Value
 - Subfunction 06h: Get NVRAM Checksum
 - Subfunction 07h: Set NVRAM Checksum
 - Subfunction 08h: Reset NVRAM Default
 - Subfunction 09h: Get NVRAM Table Address
 - Subfunction 0Bh: CS9211 Register Write
 - Subfunction 0Ch: CS9211 Register Read
 - Subfunction 0Dh: Reserved for COP8 Block Write
 - Subfunction 0Eh: Reserved for ADC12128 Read/Write
 - Subfunction 0Fh: Reserved for Serial Port 1 Read/Write
 - Subfunction 10h: Reserved for Touchscreen data read
 - Subfunction 20h: Get SCx2xx External Clock Speed
 - Subfunction 21h: Detect CPU version
 - Subfunction 31h: Read I2C Device
 - Subfunction 32h: Write I2C Device
 - Subfunction 35h: Owl board (SIO expansion) interface procedure
- Function C0h: Get ROM Configuration
- Function C1h: Get EBDA Address
- Function C2h: PS/2 Mouse Support
 - Subfunction 00h: Enable/Disable Pointing Device
 - Subfunction 01h: Reset Pointing Device
 - Subfunction 02h: Set Sample Rate
 - Subfunction 03h: Set Resolution
 - Subfunction 04h: Read Device Type
 - Subfunction 05h: Initialize Pointing Device Interface
 - Subfunction 06h: Set Scaling or Get Status
 - Subfunction 07h: Set Pointing Device Handler Address
- Function E8h: Memory Size and Map Support
 - Subfunction 01h: Get System Memory Size:
 - Subfunction 20h: Get System Memory Map

11.1 INT 15H SYSTEM SERVICES FUNCTIONS DESCRIPTIONS

This section lists those INT 15h functions that are supported in the XpressROM firmware, the parameters that system software supplies to each INT 15h function and the values that each function returns.

Function 24h, Subfunction 00h – Disable A20 Gate

Description:

This function disables the A20 functionality.

Passed:

AH = C0h
AL = 00h

System Services (Continued)**Returns:**

CF = 0 if successful
 1 if unsuccessful
AH = 00h if successful
AH = 01h if keyboard is in secure mode

System Services (Continued)**Function 24h, Subfunction 01h – Enable A20 Gate****Description:**

This function enables the A20 functionality.

Passed:

AH = C0h
AL = 01h

Returns:

CF = 0 if successful
1 if unsuccessful
AH = 00h if successful
AH = 01h if keyboard is in secure mode

Function 24h, Subfunction 02h – Get A20 Gate Status**Description:**

This function retrieves the current status of A20 gate.

Passed:

AH = C0h
AL = 02h

Returns:

CF = 0 if successful
1 if unsuccessful
AH = 00h if successful
AH = 01h if keyboard is in secure mode
AL = 00h if A20 gate is disabled
AH = 01h if A20 gate is enabled
CX = 00h

System Services (Continued)**Function 87h – Extended Memory Block Move****Description:**

This function puts the system into protected mode and copies memory from one location to another. The intent is to move data or code into an area that is not easily reached by a real-mode program.

Passed:

AH = 87h
 CX = Number of words to copy (maximum 8000h)
 ES:SI = Pointer to Memory Move Structure

Offset	Size	Description
00h	16 BYTES	Zeros (used by BIOS)
10h	WORD	Source segment length in bytes
12h	3 BYTES	24-bit linear source address, low byte first
15h	BYTE	Source segment access rights (93h)
16h	WORD	Extended access rights and high byte of source address
18h	WORD	Destination segment length in bytes
1Ah	3 BYTES	24-bit linear destination address, low byte first
1Dh	BYTE	Destination segment access rights (93h)
1Eh	WORD	Extended access rights and high byte of destination address
20h	16 BYTES	Zeros (used by BIOS)

Returns:

CF = 0 if operation completed successfully
 1 if operation failed, AH contains error codes
 AH = Status
 00h = Successful
 01h = Parity error
 02h = Interrupt error
 03h = Address line 20 gating failed
 86h = Unsupported function

Related Functions:

None

Function 88h – Get Extended Memory Size**Description:**

This function returns the amount of memory, above 1 MB, in the system. The nature of this interface limits it to reporting 65 MB total in the system.

Passed:

AH = 88h

Returns:

CF = 0 if operation completed successfully
 1 if operation failed, AH contains error code
 AX = Amount of continuous memory starting at an absolute address of 100000h
 AH = 86h (Unsupported Function)

System Services (Continued)**Function 89h – Enter Protected Mode****Description:**

This function forces the system into protected mode. The caller must construct a GDT (Global Descriptor Table) to be used by the BIOS.

Passed:

AH = 89h
 BL = Interrupt number for IRQ0
 BH = Interrupt number for IRQ8
 ES:SI = Pointer to GDT (see Table 11-1)

Returns:

CF = 0 if operation completed successfully
 1 if operation failed, AH contains error code
 AH = 0 if successful
 FFh error enabling A20
 BP = May be destroyed

Table 11-1. Global Descriptor Table (GDT)

Offset	Size	Description
00h	8 BYTES	Zeros (null descriptor)
08h	8 BYTES	GDT descriptor: See Table 11-2.
10h	8 BYTES	IDT descriptor
18h	8 BYTES	DS descriptor
20h	8 BYTES	ES descriptor
28h	8 BYTES	SS descriptor
30h	8 BYTES	CS descriptor
38h	8 BYTES	Uninitialized, used to build descriptor for BIOS CS

Table 11-2. Descriptor Table Format

Offset	Size	Description
00h	WORD	Segment limit, low WORD
02h	3 BYTES	Segment base address, low 24 bits
05h	BYTE	Access mode
06h	BYTE	Extended access mode
07h	BYTE	Segment base address, high 8 bits

System Services (Continued)**11.1.1 I²C Access Functions Specific to Geode processors**

The XpressROM firmware provides the following functions to access I²C devices installed in the system. These can be used in platform-specific modules or when writing tools and utilities.

I²C Access Functions specific to Geode processors, INT 15h, Function BEh:

- Subfunction 00h: Read I²C Byte
- Subfunction 01h: Write I²C Byte
- Subfunction 02h: Write I²C Block

Function BEh, Subfunction 00h – Read I²C Byte**Description:**

Read a byte of data from an I²C device.

Passed:

AX = BE00h
 BX = 4E53h (= 'NS')
 CL = Register offset
 CH = I²C address

Returns:

CF = Clear if successful, or set if error
 AL = Data read

Function BEh, Subfunction 01h – Write I²C Byte**Description:**

Write a byte of data to an I²C device.

Passed:

AX = BE00h
 BX = 4E53h (= 'NS')
 CL = Register offset
 CH = I²C address
 DL = Data read

Returns:

CF = Clear if successful, or set if error

Function BEh, Subfunction 02h – Write I²C Block**Description:**

Write a byte of data to an I²C device.

Passed:

AX = BE00h
 BX = 4E53h (= 'NS')
 CL = Register offset
 CH = I²C address
 DL = Number of bytes to transfer
 DS:SI = Pointer to transfer buffer

Returns:

CF = Clear if successful, or set if error

System Services (Continued)**11.1.2 NVRAM Access Functions Specific to Geode Processors:**

The XpressROM firmware provides the following functions (specific to Geode processors) to access the installed NVRAM devices in the system. Information is accessed through a token rather than an absolute address. This eliminates the necessity of knowing specific addresses, and allows for future reorganization of the NVRAM map.

Note: Although these BIOS calls are present and functional, they are intended for forward compatibility. Currently, the XpressROM firmware (POST) does not use information found in NVRAM. All initialization values are specified using the configurator (see Section 16.4 “THE CONFIGURATOR” on page 134).

NVRAM Access Functions specific to Geode processors, INT 15h, Function BEh:

- Subfunction 03h: Read NVRAM Data
- Subfunction 04h: Write NVRAM Data
- Subfunction 05h: Get Default NVRAM Value
- Subfunction 06h: Get NVRAM Checksum
- Subfunction 07h: Set NVRAM Checksum
- Subfunction 08h: Reset NVRAM Default
- Subfunction 09h: Get NVRAM Table Address

Function BEh, Subfunction 03h – Read NVRAM Data**Description:**

Reads data associated with a particular token from NVRAM. The data is right-justified and may be up to eight bits wide. The XpressROM firmware is responsible for determining the correct location from which information will be read. The caller is responsible for knowing the format of the data (see Table 11-3 on page 80).

Passed:

AX = BE03h
 BX = 4E53h (= ‘NS’)
 CX = Token
 DH = Media
 00 = CMOS
 01 = System ROM Flash
 02 = Serial EEPROM
 03 – FF = Reserved

Returns:

CF = Clear if successful, or set if error
 DL = Value read, or error code
 Error Codes:
 00 = Reserved
 01 = Token not supported
 02 = Software support not present (e.g. not implemented)
 03 = Hardware support not present (e.g. no CMOS in system)
 04 = Invalid media specified
 05 = Token not valid in particular media

Function BEh, Subfunction 04h – Write NVRAM Data**Description:**

Write data associated with a particular token, to NVRAM. The data passed to the function should be right justified and may be up to eight bits wide. The XpressROM firmware is responsible for determining the correct location to which information will be written. The caller is responsible for knowing the format of the data (see Table 11-3 on page 80).

Passed:

AX = BE04h
 BX = 4E53h (= ‘NS’)
 CX = Token
 DL = Value to write
 DH = Media (See Subfunction 03h)

Returns:

CF = Clear if successful, or set if error
 DL = Error code if failure (See Subfunction 03h)

System Services (Continued)**Function BEh, Subfunction 05h – Get Default NVRAM Value****Description:**

Retrieves the default setting for a particular token. See Table 11-3 on page 80 for the format of the tokens

Passed:

AX = BE05h
 BX = 4E53h (= 'NS')
 CX = Token
 DH = Media (See Subfunction 03h)

Returns:

CF = Clear if successful, or set if error
 DL = Default value, or error code (see Subfunction 03h)

Function BEh, Subfunction 06h – Get NVRAM Checksum**Description:**

Returns currently stored checksum. This function attempts to calculate the correct checksum also.

Passed:

AX = BE06h
 BX = 4E53h (= 'NS')
 DH = Media (See Subfunction 03h)

Returns:

CF = Clear if successful, or set if error
 AL = Calculated checksum
 AH = NVRAM checksum value
 DL = Default value, or error code (see Subfunction 03h)

Function BEh, Subfunction 07h – Set NVRAM Checksum**Description:**

Calculates the checksum for NVRAM and stores at the location for the checksum token.

Passed:

AX = BE07h
 BX = 4E53h (= 'NS')
 DH = Media (See Subfunction 03h)

Returns:

CF = Clear if successful, or set if error
 DL = Error code if failure (see Subfunction 03h)

Function BEh, Subfunction 08h – Reset NVRAM Default**Description:**

Resets all tokens in a particular type of media to the default values.

Passed:

AX = BE08h
 BX = 4E53h (= 'NS')
 DH = Media (See Subfunction 03h)

Returns:

CF = Clear if successful, or set if error
 DL = Error code if failure (see "Function BEh, Subfunction 03h – Read NVRAM Data" on page 78, errors 02, 03, 04 only)

System Services (Continued)**Function BEh, Subfunction 09h – Get NVRAM Table Address****Description:**

Intended for internal usage. This function returns the address for the NVRAM and media type tables.

Passed:

AX = BE09h
 BX = 4E53h (= 'NS')

Returns:

SI = Address of NVRAM map
 CX = Number of entries in NVRAM map
 DI = Address of media table
 DX = Number of entries in NVRAM table

Table 11-3. NVRAM Table

Token ¹	Description	Width	Definition
PM	Power management enable	1 bit	0 = Disabled 1 = Enabled
PS	Suspend state	2 bits	00 = Stop Clock 01 = 3V Suspend 10 = Suspend-to-Disk 11 = Suspend-to-RAM
TY	Standby timeout	1 byte	0000 = Disabled 0001 = 5 seconds 0010 = 10 seconds 0011 = 15 seconds 0100 = 30 seconds 0101 = 45 seconds 0110 = 1 minute 0111 = 2 minutes 1000 = 5 minutes 1001 = 10 minutes 1010 = 15 minutes 1011 = 30 minutes 1100 = 45 minutes 1101 = 60 minutes 1110 = 90 minutes 1111 = 120 minutes
TS	Suspend timeout	1 byte	See "Standby timeout"
T1	Idle timeout	1 byte	See "Standby timeout"
TK	Keyboard/mouse timeout	1 byte	See "Standby timeout"
T1	HDD1 timeout	1 byte	See "Standby timeout"
T2	HDD2 timeout	1 byte	See "Standby timeout"
TF	FDD timeout	1 byte	See "Standby timeout"
TV	Video timeout	1 byte	See "Standby timeout"
TU	Serial port timeout	1 byte	See "Standby timeout"
TP	Parallel port timeout	1 byte	See "Standby timeout"
W1	Wakeup mask PIC1	1 byte	Bitmask: bit0 = IRQ0, etc.
W2	Wakeup mask PIC2	1 byte	Bitmask: bit0 = IRQ8, etc.
WC	Wakeup century	1 bit	0 = 1999 1 = 2000
WY	Wakeup year	1 byte	BCD value for year mod 100 (0 - 99)
WM	Wakeup month	1 byte	BCD value for month (0 - 12)
WD	Wakeup day	1 byte	BCD value for day (0 - 31)
WH	Wakeup hour	1 byte	BCD value for hour (0-23)

System Services (Continued)

Table 11-3. NVRAM Table

Token ¹	Description	Width	Definition
WT	Wakeup minute	1 byte	BCD value for minute (0 - 59)
WS	Wakeup second	1 byte	BCD value for second (0 - 59)
WE	Wakeup enable	3 bits	000 = Disabled Need other values here, John Kabat
PS	S2D / S2R	1 bit	0 = Normal boot 1 = Wake from S2D or S2R
U1	UART1 configuration	3 bits	000 = Disabled 001 = 3F8, IRQ4 010 = 2F8, IRQ3 011 = 3E8, IRQ4 100 = 2E8, IRQ3
U2	UART2 configuration	3 bits	000 = Disabled 001 = 3F8, IRQ4 010 = 2F8, IRQ3 011 = 3E8, IRQ4 100 = 2E8, IRQ3
U3	UART3 configuration	3 bits	000 = Disabled 001 = 3F8, IRQ4 010 = 2F8, IRQ3 011 = 3E8, IRQ4 100 = 2E8, IRQ3
U4	UART4 configuration	3 bits	000 = Disabled 001 = 3F8, IRQ4 010 = 2F8, IRQ3 011 = 3E8, IRQ4 100 = 2E8, IRQ3
LA	LPT1 I/O port	2 bits	00 = Disabled 01 = 378 10 = 278 11 = 3BC
LI	LPT1 IRQ (only used when necessary, depending on port mode)	2 bits	00 = IRQ5 01 = IRQ7 10 = 11 = undefined
LD	LPT1 mode	3 bits	000 = Compatible 001 = PS/2 bidirectional 010 = EPP 1.7 011 = EPP 1.9 100 = ECP 101 - 111 = undefined
LB	LPT2 I/O port	2 bits	00 = Disabled 01 = 378 10 = 278 11 = 3BC
LQ	LPT2 IRQ (only used when necessary, depending on port mode)	2 bits	00 = IRQ5 01 = IRQ7 10 = 11 = undefined
LO	LPT2 mode	3 bits	000 = Compatible 001 = PS/2 bidirectional 010 = EPP 1.7 011 = EPP 1.9 100 = ECP 101 - 111 = undefined
ID	IDE controller enable	1 bit	0 = Disabled 1 = Enabled

System Services (Continued)**Table 11-3. NVRAM Table**

Token ¹	Description	Width	Definition
I1	Primary IDE enable (may not be available for all controllers)	1 bit	0 = Disabled 1 = Enabled
I2	Secondary IDE enable (may not be available for all controllers)	1 bit	0 = Disabled 1 = Enabled
FD	FDD enable	1 bit	0 = Disabled 1 = Enabled
AU	Audio enable	1 bit	0 = Disabled 1 = Enabled
AA	Audio base address	2 bits	00 = 220 01 = 240 10 = 260 11 = 280
AI	Audio IRQ	2 bits	00 = IRQ5 01 = IRQ7 10 = IRQ9 11 = IRQ10
A1	Audio 8-bit DMA	2 bits	00 = DMA0 01 = DMA1 10 = Undefined 11 = DMA3
A2	Audio 16-bit DMA	2 bits	00 = Undefined 01 = DMA5 10 = DMA6 11 = DMA7
US	USB Enable (may mean the device is present or not)	1 bit	0 = Disabled 1 = Enabled
UL	Legacy USB on/off	1 bit	0 = Disabled 1 = Enabled
CA	Cache on/off	1 bit	0 = Disabled 1 = Enabled
CM	Cache mode	1 bit	0 = Write-back 1 = Write-through
B1	Boot device 1	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved
B2	Boot device 2	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved
B3	Boot device 3	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved
B4	Boot device 4	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved

System Services (Continued)**Table 11-3. NVRAM Table**

Token ¹	Description	Width	Definition
B5	Boot device 5	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved
B6	Boot device 6	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved
B7	Boot device 7	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved
B8	Boot device 8	3 bits	000 = None 001 = HDD1 010 = FDD 011 = Network 100 – 111 = Reserved
MO	Memory optimization	1 bit	0 = Automatic optimization 1 = Manual settings
MR	SDRAM clock ratio	3 bits	000 = Reserved 001 = 2.0 010 = 2.5 011 = 3.0 100 = 3.5 101 = 4.0 110 = 4.5 111 = 5.0
MD	SDRAM drive/slew	3 bits	000 = Lowest drive strength 001... 111 = Highest drive strength
MS	SDRAM shift SDCLK	3 bits	000 = No shift 001 = Shift 0.5 core clocks 010 = 1.0 011 = 1.5 100 = 2.0 101 = 2.5 110 = 3.0 111 = Undefined
MC	CAS latency	3 bits	000 = Undefined 001 = Undefined 010 = 2 CLK 011 = 3 100 = 4 101 = 5 110 = 6 111 = 7
M1	tRC	4 bits	0000 = Undefined 0001 = 2 CLK 0010 = 3 1111 = 16
M2	tRAS	4 bits	0000 = Undefined 0001 = 2 CLK 0010 = 3 1111 = 16

System Services (Continued)**Table 11-3. NVRAM Table**

Token ¹	Description	Width	Definition
M3	tRP	3 bits	000 = Undefined 001 = 1 CLK 010 = 2 111 = 7
M4	tRCD	3 bits	000 = Undefined 001 = 1 CLK 010 = 2 111 = 7
M6	TDPL	3 bits	000 = Undefined 001 = 1 CLK 010 = 2 111 = 7
VO	TV-out	2 bits	00 = Disabled 01 = S-Video 10 = Composite 11 = Both
VF	TV format	2 bits	00 = NTSC 01 = PAL 10 = Reserved 11 = Reserved
VM	TV Mode	2 bits	00 = 640x480 01 = 800x600 10 = 1024x768 11 = Reserved
VH	TV horizontal position	1 byte	Need values here, John Kabat
VV	TV vertical position	1 byte	Need values here, John Kabat

1. Note: In general, tokens have been named by category
- Px = Power related
 - Tx = Timeouts
 - Wx = Wakeup related
 - Ux = UART and USB
 - Lx = LPT values
 - I x = IDE
 - Fx = FDD
 - Ax = Audio
 - Cx = Cache
 - Bx = Boot devices
 - Mx = Memory controller

System Services (Continued)**11.1.3 INT 15h function For XpressROM information**

This function allows for reading build and runtime information from XpressROM. Information includes everything shown on the summary screen, the “build strings”, files overridden in the build, boot errors, etc. The data is put into the buffer in a readable text format, with each line terminated with a carriage return (character 0Dh). The buffer is terminated with a null character.

Function BEh, Subfunction 99h – Read XpressROM Information**Passed:**

AX = BE99h
 DS:S = Buffer for XpressROM information
 If DS=0, The buffer is not written, but the size needed is returned in AX

Returns:

If successful:
 CF = 0
 AX = Buffer size in bytes
 If unsuccessful:
 CF = 1
 AX = error code**

Registers Not Preserved:

None

11.1.4 New heading and description for Cxh functions**Function C0h – Get ROM Configuration****Description:**

The ROM holds information about the system’s design. This function retrieves a pointer to the data.

Passed:

AH = C0h

Returns:

CF = 0 if operation completed successfully
 1 if 1 if operation failed, AH contains error code
 AH = 86h (Unsupported Function)
 ES:BX = Pointer to the ROM table:

Offset	Size	Description	Value
00h	WORD	Number of bytes following	08h
02h	BYTE	Model	FCh
03h	BYTE	Submodel	01h
04h	BYTE	BIOS revision	00h
05h	BYTE	Feature byte 1	74h
06h	BYTE	Feature byte 2	00h
07h	BYTE	Feature byte 3	00h
08h	BYTE	Feature byte 4	00h

System Services (Continued)**Function C1h – Get EBDA Address****Description:**

This function returns a pointer to the system's Extended BIOS Data Area.

Passed:

AH = C1h

Returns:

CF = 0 if operation completed successfully

1 if 1 if operation failed

ES = Segment of the EBDA

11.1.5 PS/2 Mouse Support

The XpressROM firmware provides support for detecting, initializing and handling PS/2 mouse activity. For information on PS/2 mouse support, please refer to IBM Mouse Technical Reference.

PS/2 Mouse Support Functions, INT 15h, Function C2h:

- Subfunction 00h: Enable/Disable Pointing Device
- Subfunction 01h: Reset Pointing Device
- Subfunction 02h: Set Sample Rate
- Subfunction 03h: Set Resolution
- Subfunction 04h: Read Device Type
- Subfunction 05h: Initialize Pointing Device Interface
- Subfunction 06h: Set Scaling Or Get Status
- Subfunction 07h: Set Pointing Device Handler Address

Function C2h, Subfunction 00h: – Enable/Disable Pointing Device**Description:**

This subfunction enables the pointing device, allowing its use, or disables the device so it can no longer be used.

Passed:

AX = C200h

BH = 00h – Disable

= 01h – Enable

Returns:

If successful:

CF = 0

AH = 0

Else:

CF = 1

AH = Status

00h = Successful

01h = Invalid function

02h = Invalid input

03h = Interface error

04h = Need to resend

05h = No device handler installed

System Services (Continued)**Function C2h, Subfunction 01h: – Reset Pointing Device****Description:**

Resets the pointing device: IRQ 12 is unmasked, the interface is disabled, the pointing device is reset (command FFh), and if found successfully, bit 4 of the equipment determination WORD in the BDA (40:10) is set.

Passed:

AH = C201h

Returns:

If successful:

CF = 0

AH = 0

BH = Device ID

BL = Value returned by pointing device (AAh if device is a mouse)

Else:

CF = 1

AH = Status (see Subfunction 00h)

Function C2h, Subfunction 02h: – Set Sample Rate**Description:**

Sets the sample rate for the pointing device. Note that if an invalid sample rate is passed, the return status reflects invalid input. Sending the F3h command to the device, followed by the rate value sets the sample.

Passed:

AH = C202h

BH = Sample rate

00h = 10 samples/second

01h = 20/second

02h = 40/second

03h = 60/second

04h = 80/second

05h = 100/second

06h = 200/second

Returns:

If successful:

CF = 0

AH = 0

Else:

CF = 1

AH = Status (see Subfunction 00h)

System Services (Continued)**Function C2h, Subfunction 03h: – Set Resolution****Description:**

Sets the resolution for the pointing device. Sending the E8h command to the device, followed by the resolution value sets the resolution.

Passed:

AH = C203h
 BH = Resolution
 00h = One count per mm
 01h = Two counts per mm
 02h = Four counts per mm
 03h = Eight counts per mm

Returns:

If successful:

CF = 0
 AH = 0

Else:

CF = 1
 AH = Status (see Subfunction 00h)

Function C2h, Subfunction 04h: – Read Device Type**Description:**

Reads the device type of the pointing device. Sending the F2h command to the device causes the device to return its ID. The ID for a mouse should be 00h.

Passed:

AH = C204h

Returns:

If successful:

CF = 0
 AH = 0
 BH = Device ID

Else:

CF = 1
 AH = Status (see Subfunction 00h)

Function C2h, Subfunction 05h: – Initialize Pointing Device Interface**Description:**

Initializes the pointing device interface flags by clearing the Acknowledge bit (bit 5) in the Extended BDA (offset 26h) and setting the package size (bits 0-3 in the EBDA offset 27h).

Passed:

AH = C205

Returns:

If successful:

CF = 0
 AH = 0

Else:

CF = 1
 AH = Status (see Subfunction 00h)

Function C2h, Subfunction 06h: – Pointing Device Extended Commands**Description:**

Blah, blah, blah, finish this and doublecheck all status

Passed:

AH = C206h

System Services (Continued)**Returns:**

If successful:

CF = 0

AH = 0

Else:

CF = 1

Function C2h, Subfunction 07h: – Set Pointing Device Handler Address**Description:**

Allows the caller to set the address for the handler. The vector is stored in the Extended BDA at offset 22h, and the driver flag is set in offset 27h.

Passed:

AH = C207h

ES:BX = Far pointer to the handler, or 0000:0000 to cancel

Returns:

If successful:

CF = 0

AH = 0

Else:

CF = 1

AH = Status (See Subfunction 00h)

Function E8h, Subfunction 01h – Get System Memory Size**Description:**

Reports the amount of system memory present and configured.

Passed:

AX = E801h

Returns:

AH = Extended memory between 1 MB and 16 MB (in KB blocks)

BX = Extended memory above 16 MB (in 64 KB blocks)

CX = Configured memory between 1 MB and 16 MB (in KB blocks)

DX = Configured memory above 16 MB (in 64 KB blocks)

CF = Clear if successful, set on error

System Services (Continued)**Function E8h, Subfunction 20h – Get System Memory Map****Description:**

Reports a memory map of the memory installed in the system. Data is in the form of table entries.

Passed:

EAX = E820h
 EDX = 534D4150h ('SMAP')
 EBX = Continuation value, or 00000000h the first time
 ECX = Size of buffer for result, in bytes (must be >= 20 bytes)
 ES:DI = Buffer for result

Returns:

EAX = 534D4150h ('SMAP')
 ES:DI = Buffer is filled
 EBX = Continuation value (=00000000h if no more entries)
 ECX = Actual length of data written to buffer (maximum of 20)

Table Entry Format:

Offset	Size	Description
00h	QWORD	Base Address
08h	QWORD	Length, in Bytes
0Ch	DWORD	Type of Address Range (see below)

Address Range Type:

Value	Description
01h	Memory Available for OS
02h	Reserved/Not Available (e.g. for System ROMs, etc.)
03h	ACPI Reclaim Memory (not used by XpressROM firmware)
04h	ACPI NVS Memory (not used by XpressROM firmware)

System Services (Continued)

12.0 Miscellaneous Services

The XpressROM firmware contains support for other miscellaneous software interrupts.

Miscellaneous Services Support:

- INT 11h – Equipment List
- INT 12h – Get Memory Size
- INT 1Bh – Control Break Handler

12.1 MISCELLANEOUS SERVICES FUNCTIONS DESCRIPTIONS

INT 11h – Equipment List

Description:

This function returns a bitmask describing the devices installed in the system.

Passed:

None

Returns:

AX = BIOS equipment list WORD:

- [15:14] = Number of parallel ports installed
- 13 = Reserved
- 12 = Set if game adapter is present
- [11:9] = Number of serial ports installed
- 8 = DMAC present
- [7:6] = If bit 0 is set, this is the number of FDD(s) – 1
- [5:4] = Initial video mode
- [3:2] = Number of 64 KB banks of RAM
- 1 = 80x87 coprocessor installed
- 0 = FDD(s) installed, number specified in bits [7:6]

INT 12h – Get Memory Size

Description:

This function the amount of RAM present in the system (up to 640 KB).

Passed:

None

Returns:

AX = Amount of usable RAM, in units of 1 KB

INT 1Bh – Control Break Handler

Description:

When Ctrl-Break is pressed, the keyboard code calls this interrupt.

Passed:

None

Returns:

None

13.0 System Management Mode Software

The System Management Mode (SMM) software for the Geode family of products has unique properties, which allow for many nontraditional features. The SMM software, along with specialized hardware allows the System Management Interrupt (SMI) handler to emulate functionality that is normally associated with discrete devices (e.g., VGA, audio, UARTs, etc.). This is accomplished via the ability to enter SMM mode very quickly, and also the ability to service nested SMIs. In addition, the SMM software controls power management, much like in a more traditional system.

There are several types of occurrences that can cause an SMI, thereby invoking the SMI handler. In general, these sources are configurable, so that they can be enabled or disabled.

- Configurable I/O and Memory Traps:
 - The hardware supports setting an I/O or memory address, range and attributes which causes an SMI when the criteria is met.
- Special Traps:
 - The hardware also contains specialized trapping ability, for example, when audio events or video events occur. Another example, is when UART or LPT traditional I/O addresses are read or written.
- General Purpose Timers:
 - The Geode I/O Companion devices contain several timers that are configurable. When a timer expires, the chip generates an SMI.
- Inactivity Timers:
 - The hardware also contains function-specific timers for power management. These are reset when user-activity occurs on the particular function, for example, the keyboard. When the timer expires, an SMI is generated.
- GPIO Activity:
 - The Geode I/O companion devices contain some number of General Purpose I/O pins. The pins are configurable, to generate an SMI if there is activity on a GPIO pin.
- PCI Configuration Cycle:
 - Some Geode I/O companion devices can generate an SMI on accesses to particular addresses in the PCI configuration space.

The SMM software is responsible for determining the cause of the SMI, generally by checking status registers. After determining the purpose for the SMI, the SMM software then dispatches to the appropriate VSA handler.

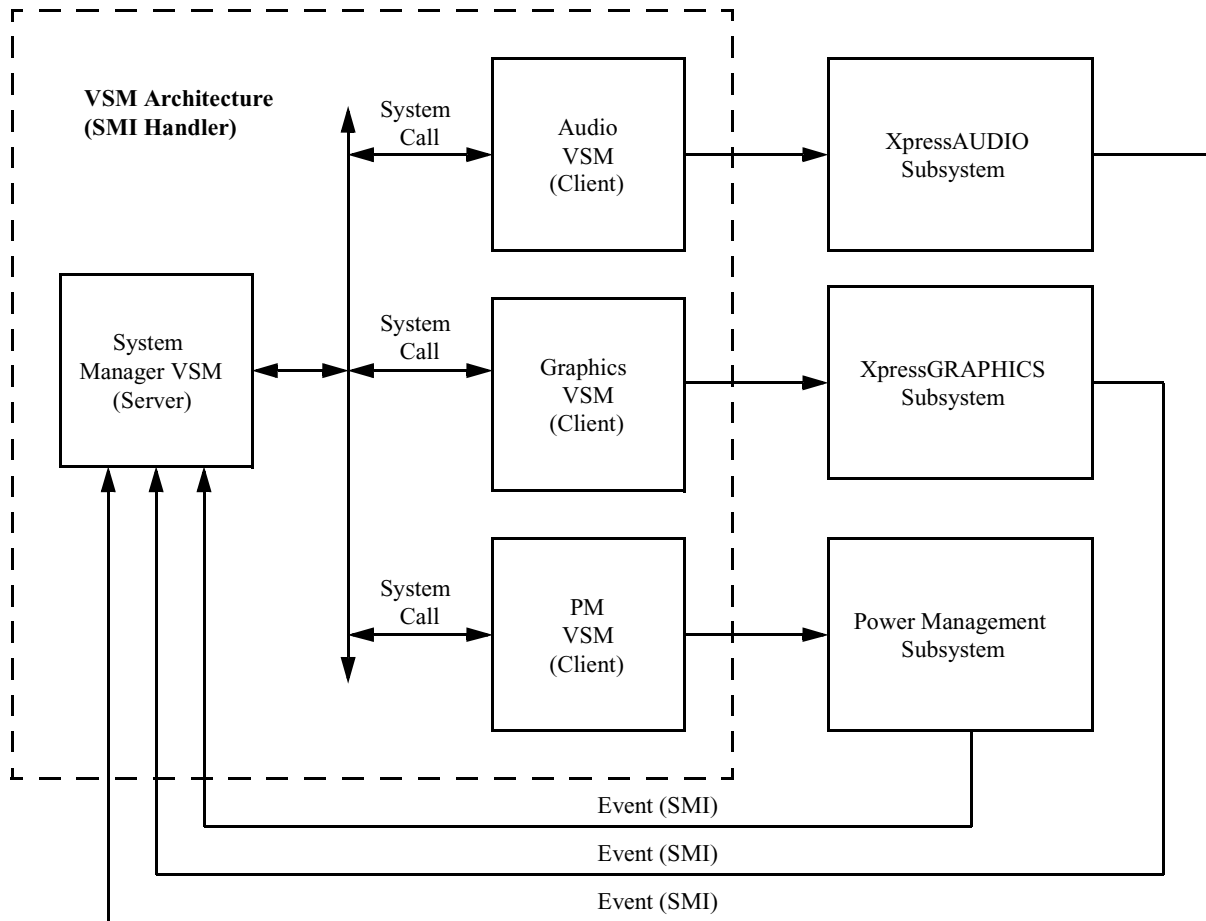
13.1 ARCHITECTURE

The SMI handler has been designed so that new functionality requires essentially no changes to the existing firmware. There are individual binary components, each called a Virtual Support Module (VSM), which provide specific pieces of functionality. Each VSM is isolated from every other VSM in terms of functionality and code. The SMM software has also been constructed following a client-server model. The dispatcher VSM is called the System Manager and acts as the server. The System Manager code begins executing as soon as an SMI occurs, and is responsible for saving the state of the system and preparing the SMM system to run. The System Manager is responsible for determining the cause for the SMI and then dispatching to the appropriate VSM(s). Figure 13-1 shows an example of the VSM client/server methodology.

Each of the remaining VSMs (clients) is simply an endless message-processing loop. As an example, in the course of processing an “initialization” message, a VSM may register to be notified upon the occurrence of various types of events (traps, timers, etc.). After processing the initialization message, the VSM retrieves the next message out of its queue via a system-call macro. The macro allows control to return to the System Manager which is then free to call any VSM, or cause the system to exit the SMI handler.

With this architecture, it is up to the System Manager to be knowledgeable of the hardware that is in the system and manage its SMI-generating resources. SMI resource management is accomplished by requiring the other VSMs to register or unregister for events that can cause an SMI to occur.

Two main mechanisms exist for communication with and within SMM. Virtual registers see “Virtual Registers” on page 101, aka VR, are used for interprocess, or module communication between VSMs. VRs are also used when external application or driver software desires to communicate with SMM. VSA messages are the other method for communicating in a standard fashion, with other VSMs and with the System Manager which is also a VSM. Both communication mechanisms are implemented in the System Manager VSM.

System Management Mode Software (Continued)**Figure 13-1. VSM Client-Server Model Example****13.2 FIRMWARE REQUIREMENTS**

The SMM software has several requirements that must be adhered to. The first is initialization. During POST, the firmware should copy the SMM software image to a location below 640 KB and then execute a far call to the initialization code (Offset 20h). At the end of POST, the firmware must notify the SMM software (via S/W SMI 5000h) so that any final initializations may take place.

System Management Mode Software (Continued)

13.3 SYSTEM MANAGER

When an SMI occurs, the processor microcode saves minimal amount of processor state into an SMM header. The processor is placed into real mode and execution is vectored to the physical address determined by the SMAR (SMM Address Region) registers. The SMM initialization code is responsible for setting these registers to the location of the System Manager entry point. The code at the entry point to the System Manager:

- Saves the processor state not already saved by the processor micro-code.
- Initializes the processor to the SMM software environment.
- Determines the source(s) of the SMI and enters the event(s) into message queue(s).
- Dispatches to the VSMs that have non-empty message queues.
- Restores the interrupted task's state and resumes execution.

The behavior of the first two steps and last step above is slightly different when the SMI is a nested SMI.

The System Manager also provides services to VSMs, such as:

- Manages inter-module message delivery.
- Allows chipset and/or processor dependency abstraction.
- Provides a mechanism for guarding critical sections of code.
- Handles memory allocation of stack and data segments.
- Ensures proper ordering of event handling.

13.4 EVENTS

Events are SMIs. A VSM may register as a handler of one or more SMI events. This normally occurs at VSM initialization, but dynamic registration is also allowed. Multiple VSMs may register for the same event. The order that VSMs are invoked for an event is determined by the priority specified at registration time. When the event occurs, the VSM that has registered with the highest priority for the event is invoked. If a VSM chooses to handle the event, nothing extra needs to be done. If a VSM chooses not to handle the event, it must pass the event back to the System Manager via the `SYS_PASS_EVENT` macro. The System Manager then sends a message to the next highest priority VSM that registered for the event, and so on until a VSM handles the event.

13.5 MESSAGING

All event notification happens via messages from the System Manager. Therefore, the code at the entry point of each VSM must be a message-handling loop. A VSM is called only to respond to a message from the System Manager. The message queue for a VSM is stored in the VSM header. The `SYS_GET_NEXT_MSG` macro is provided for accessing the message queue. When the message queue is empty, the `SYS_GET_NEXT_MSG` macro automatically returns control to the System Manager. Control will be returned to the VSM when there is another message in its queue. The supported messages are enumerated in Section 13.11 “Messages” on page 114.

13.6 SYSTEM CALLS

One of the design goals of the XpressROM SMM technology is the abstraction of implementation details. System calls and macros have been defined to provide control mechanisms and access to sensitive system values and structures. Although VSMs could directly perform certain tasks they should, instead, call the System Manager to perform these functions. This allows hardware bug fixes as well as companion chip details to be abstracted from the VSM. Because most chipset programming is performed at initialization where performance is not critical, the slight loss in performance due to system overhead is more than made up by readability, portability and the handling of shared resources. The available system calls are discussed in detail Section 13.10 “System Calls and Macros” on page 107.

Another “tool” that supports source code maintainability is the use of a set of predefined macros. These macros perform their specified functionality using the best possible method and again, abstracting the implementation details. The use of these macros facilitates portability when system call interfaces are changed or when better system call implementations are developed. The available macros are discussed in detail in Section 13.13 “Macros” on page 121.

System Management Mode Software (Continued)

13.7 RESOURCE MANAGEMENT

The System Manager owns the top level SMI source registers. A VSM owns its second level source registers and all resources related to the functional component that it controls. For example, the XpressAUDIO VSM does not poll the top level SMI source register directly. It receives an EVENT_AUDIO message from the System Manager when there is an audio-related event, and then reads the audio SMI sources register(s) to determine the specific audio event.

When resources are accessed through index/data pairs the code that uses those resources should save and restore the index/data within a critical section. The PCI configuration address register is an exception because it is part of a VSM's state. When a VSM uses floating-point instructions, the VSM is responsible for saving and restoring the floating-point state. Also, a VSM must not write to global resources (e.g., CPU configuration registers).

13.7.1 Reentrancy

VSM code is not required to be reentrant. While it is possible for a VSM to be executing when another registered event occurs, the System Manager's scheduler will not normally preempt the currently executing VSM to let the more recent SMI event be processed. Instead, the System Manager will service the SMI, and record the SMI event by entering the appropriate message into the message queue of the VSM(s) that are registered to handle that event. It then returns control to the interrupted VSM.

If the VSM registered an event as "preemptive", the System Manager interrupts the executing VSM, saves the pertinent context on the VSM's stack (Instruction Pointer, etc.), puts the corresponding event message into the VSM's message queue, and dispatches to the VSM's entry point. When the VSM completes the processing of the high-priority event, the System Manager restores the previous state of the VSM and continues execution.

13.7.2 Event Registration

A VSM may register as a handler of SMI event(s). This normally occurs at VSM initialization, but dynamic registration is also allowed. Multiple VSMs may register for the same event. The order that VSMs are invoked is determined by the priority specified at the time of event registration. If a VSM chooses to handle the event, nothing extra needs to be done. If a VSM chooses *not* to handle the event, it must pass the event back to the System Manager via the SYS_PASS_EVENT macro. The System Manager then sends a message to the next highest priority VSM that registered for the event, and so on until a VSM handles the event.

The architecture supports the assignment of a 16-bit priority to each registered event. These priorities are used to control the order that an event is serviced. Bits [16:8] specify the order that an SMI is processed relative to other VSMs. Bits [7:0] specify the order that an SMI event message is inserted relative to other messages within the VSM's message queue. Higher numbers indicate a more important priority. For example, assume VSM *X* registers an event with priority 0x10000 and VSM *Y* registers an event with priority 0x20000. If VSM *X*'s event is being processed and VSM *Y*'s event occurs, then VSM *Y* preempts VSM *X*. VSM *X* is not given control until the next message in VSM *Y*'s queue is lower priority than the priority of the message being handled by VSM *X*.

A value of 00h in bits [7:0] means the event message will be inserted at the end of a VSM's queue. If an event message is already pending in the queue with the same priority as a new event, the newer message will be inserted after the older message. That is, a VSM's event queue is sorted with the first key being priority and the second key being time of event.

An exception to this priority scheme occurs if a VSM is marked with "NoPreempt" in its header. In this case, no preemption will occur, and any higher priority event messages will simply be entered into the appropriate VSM message queue and control will be returned to the original VSM. This is effectively an override on inter-VSM priorities. Only VSMs that are truly high priority should be marked with the "NoPreempt" flag set. This feature is designed with the XpressAUDIO subsystem in mind.

Note: Some combinations of events may not be supported by the underlying hardware, but may be emulated by the System Manager. For example, a VSM may register as a handler of a GPIO falling and rising edge events. The hardware only supports generation of an SMI on one or the other GPIO edge. The System Manager detects such situations and dynamically reprograms the hardware in order to simulate the desired effect. A list of events are found in Section 13.12 "Events" on page 117.

System Management Mode Software (Continued)

13.7.3 Nested SMIs

The SMM software makes full use of nested SMIs. SMI nesting is enabled at all times, with the following exceptions:

- Within a critical section of a VSM.
- Within the System Manager.
- While handling virtual register accesses from VSMs.

The effect of SMI nesting is largely transparent to a VSM. Except for the small time-slice taken to record the event in the appropriate message queue, the interrupted VSM is unaffected. Control is normally returned to the interrupted VSM immediately. The two exceptions are event priorities and virtual registers.

Since it is possible that a VSM might be interrupted, a VSM must guard certain sections of code as a critical section. The most common example is when a VSM accesses an index/data I/O pair. It is possible for the VSM to be interrupted after writing the index but before writing the data. If an interrupting VSM accesses the same index/data pair, it will corrupt the index from the first VSM. To prevent this situation from occurring, `BEGIN_CRITICAL_SECTION` and `END_CRITICAL_SECTION` macros must surround such sections of code.

A VSM that accesses a virtual register expects the effects of that access to occur immediately. In order for this to happen, the VSM that handles the accessed virtual register must be dispatched immediately. This occurs as follows: An `EVENT_VIRTUAL_REGISTER` message is entered at the beginning of the appropriate VSM message queue. That VSM is given control until the message is retrieved and serviced. Control is then returned to the VSM that accessed the virtual register.

13.8 VIRTUAL SUPPORT MODULES

Each major functional component of the SMM software technology is implemented as a separately compiled VSM. This architectural feature is the key to the adaptability of XpressROM SMM technology. Concatenating the modules with the desired functionality produces a customized SMM image. For example, if a platform does not require the XpressAUDIO subsystem, the XpressAUDIO VSM is excluded from the SMM image. Furthermore, if SMM functionality is dependent on a platform-specific feature, the VSM writer has the freedom to implement these details in a custom VSM without requiring support from National Semiconductor.

A VSM consists of three major components: a VSM header, a VSM library, and a message handler. The first two components are supplied by National Semiconductor. The third, the message handler, is the code that determines the functionality of a VSM.

Typically, a VSM writer modifies only three fields in the VSM header file. These are the *VSM_Type*, *VSM_Version*, and *EntryPoint*; fields that are unique to each VSM. The default values for the other fields are usually sufficient for most applications and are initialized by the SMM software installation code.

Each VSM must be linked to the VSM library file. This library contains the code for the system calls that comprise the API to the System Manager.

The VSM message handler implements the functionality of the VSM with respect to messages sent by the System Manager. It typically consists of an infinite loop that receives messages, parses each message and any parameter(s) associated with the message, and dispatches to routines within the VSM to handle the messages. When there are no more messages waiting for the VSM, control is returned to the System Manager via the process of retrieving the next message. The VSM then goes dormant until an event occurs that results in a message being entered into its message queue. A sample message handler follows.

System Management Mode Software (Continued)**Sample VSM Message Handler**

```

unsigned char      temp;
unsigned short     Msg;
unsigned long      Param[MAX_MSG_PARAM];
unsigned short     VirtualRegs[10];
unsigned char      Index;

do {
    Msg = GET_NEXT_MSG(&Param);
    switch (Msg) {
    case MSG_INITIALIZE:
        // Register as handler of Power Management virtual registers
        SYS_REGISTER_EVENT(EVENT_VIRTUAL_REGISTER, VRC_PM, 0, 0);
        break;
    case MSG_EVENT:
        switch (Param[0]) {
        case EVENT_VIRTUAL_REGISTER:
            Index = (unsigned char) Param[1];
            if (Param[2] == 0) {
                SET_AX(VirtualRegs[Index]);           // I/O read
            } else {
                // I/O write
                if (Index != 0 && Index < 10)
                    // Version register is Read-Only
                    VirtualRegs[Index] = (unsigned short) Param[3];
            }
            break;
        } // end switch(Param[0])
        break;
    } // end switch(Msg)
} while (1);

```

System Management Mode Software (Continued)

13.8.1 Memory Model

VSMs run in real mode using the “tiny” mode (i.e., CS = DS = SS). There are descriptor fields in the VSM header, but these are for use by the System Manager and should not be modified by the VSM. VSMs written in C must obey the following rules:

- Must not have a main() routine
- Must not use C library functions (e.g. malloc, printf)

13.8.2 Initialization

The SMM initialization code scans the binary image on paragraph (16-byte) boundaries for the signature “VSM”. Each signature marks the beginning of a VSM header. The SMM initialization code calls each VSM entry point with a MSG_INITIALIZE message in the VSMs message queue. In response to this message, the VSM performs its own module specific initialization, including, but not limited to:

- Data initialization
- Registering events with the System Manager

13.8.3 VSM Header

The VSM header encapsulates all data that is specific to a VSM. This design frees the System Manager from having to size its data structures based on a maximum number of VSMs. The only fields that need to be modified in the header are the VSM_Type, VSM_Version, and EntryPoint. The VSM_Type field is only used for display purposes. The currently defined types, their values, and their use are listed in Table 13-1. Use VSM_NULL if a type cannot be found that matches a VSMs primary functionality.

System Management Mode Software (Continued)**Table 13-1. VSM Types**

Name	Defined as	Type
VSM_SYS_MGR	0	System Manager
VSM_AUDIO	1	XpressAUDIO handler
VSM_VGA	2	SoftVGA handler
VSM_LEGACY	3	Standard AT peripheral handler
VSM_PM	4	Legacy Power Management handler
VSM_OHCI	5	USB OHCI
VSM_I8042	6	8042 emulator
VSM_DEBUGGER	7	SMI based debugger
VSM_ACPI	8	Virtual ACPI handler
VSM_APM	9	APM 1.2 handler
VSM_OEM_ACPI	10	OEM ACPI customizations
VSM_SMB	11	System Management Bus handler
VSM_BATTERY	12	Battery controller
VSM_RTC	13	Virtual RTC
VSM_S2D	14	Save-To-Disk handler
VSM_EXT_AMP	15	External audio amplifier handler
VSM_PCMCIA	16	PCMCIA handler
VSM_SPY	17	Spy. Receives ALL messages first
VSM_NETWORK	18	Network handler
VSM_GPIO	19	GPIO handler
VSM_KEYBOARD	20	USB keyboard to PC/AT emulation
VSM_MOUSE	21	USB mouse to PS/2 emulation
VSM_USB	22	Universal Serial Bus handler
VSM_FLASH	23	FLASH programmer
VSM_INFRARED	24	Infrared handler
VSM_THERMAL	25	Thermal monitor
VSM_NULL	26	Unspecified (e.g., SAMPLE.VSM)
VSM_MPEG	27	MPEG video decoder (EMMA)
VSM_VIP	28	Video processor (VIDEC)
VSM_LPC	29	Low Pin Count bus handler
VSM_VUART	30	Virtual UART handler
VSM_uCONT	31	Micro Controller handler
VSM_USER1	32	Place holder for additional VSM
VSM_USER2	33	Place holder for additional VSM
VSM_USER3	34	Place holder for additional VSM

System Management Mode Software (Continued)

The format of a VSM header follows.

Sample VSM Header

```
typedef struct {
    unsigned long    Signature;           // 'VSM '
    unsigned char    VSM_Type;           // Type of VSM (e.g. Legacy, PM, Audio)
    unsigned char    ForCPU;
    unsigned short   ForChipset;
    unsigned short   VSM_Version;        // Version of VMS header format
    unsigned long    VSM_Length;        // Length of VSM module in bytes
    unsigned short   EntryPoint;         // Offset of initial VSM entry point
    unsigned long    DS_Limit;
    Requirements     Flag;               // Special requirements/capabilities
    unsigned short   VSA_Version;
    USHORT           AlignSystem         DWORD alignment adjustment
    System           SysStuff            Reserved for System Manager use.
} VSM_Header;
```

Sample VSM Header structures

The Requirements structure is a 16-bit flag value.

```
typedef struct {
    unsigned short   Alignment: 5;       // 2(n+5) (e.g. 00000 = 32-byteph)
    unsigned short   CodeHi: 1;          // 1 = must load Code above DRAM_TOP
    unsigned short   CodeLo: 1;          // 1 = must load Code below 1 MB
    unsigned short   DataHi: 1;          // 1 = must load Data above DRAM_TOP
    unsigned short   DataLo: 1;          // 1 = must load Data below 1 MB
    unsigned short   SmallModel: 1;      // 1 = if Small memory model
    unsigned short   NoPreempt: 1        // 1 = VSM may be not be preempted
    unsigned short   SkipMe: 1           // 1 = Skip this VSM (normally set by the BIOS)
    unsigned short   Reserved: 4;
} Requirements;
```

The Descriptor structure has the format used by the SVDC and RSDC instructions.

```
typedef struct {
    unsigned short   limit_15_0;
    unsigned short   base_15_0;
    unsigned char    base_23_16;
    unsigned char    attr;
    unsigned char    limit_19_16;
    unsigned char    base_31_24;
    unsigned short   selector;
} Descriptor;
```

13.9 VIRTUAL REGISTERS

System Management Mode Software (Continued)

Virtual registers are I/O locations accessed with 16-bit index/data operations. The Geode hardware can be configured to trap I/O accesses. The SMM software interprets the data values and either treats the data as parameters (e.g., PM timeouts) or performs some function (e.g., change the TV encoder brightness).

Virtual registers provide a uniform method for the BIOS or 3rd party software to customize SMM parameters and features (e.g., PM timeouts). Also, virtual registers are the mechanism by which one VSM communicates with another VSM.

The effects of accessing the virtual register should appear to the VSM to occur immediately. Accessing a virtual register from within a VSA critical section will produce unpredictable results.

Virtual registers (VR) may be used to read or write information to or from the SMM software. The registers consist of an index-data pair of I/O locations (AC1Ch, AC1Eh). The System Manager is responsible for setting up the I/O trapping to support virtual registers. A VSM should use the SET_VIRTUAL_REGISTER and GET_VIRTUAL_REGISTER macros rather than directly executing the I/O cycle.

An unlock cycle is required prior to accessing a virtual register and must be executed upon every access. This feature protects this I/O space from undesirable probing by various operating systems.

Index Value:

AH = Class
AL = Parameter_ID

Data Value:

AX = Data

The index is a WORD, consisting of a VR Class and Parameter ID. The data is also a WORD. Below is sample code for writing the Standby timeout. Table 13-2 on page 104 lists supported virtual registers.

System Management Mode Software (Continued)

```
MOV    AX, 0FC53h      ; unlock code
MOV    DX, 0AC1Ch
OUT    DX, AX          ;
MOV    AH, 4           ; power management class
MOV    AL, 3           ; standby timeout
MOV    DX, 0AC1Ch      ; index register
OUT    DX, AX
ADD    DX, 2           ; point to data register
MOV    AX, 60          ; set to 1 minute
OUT    DX, AX
```

System Management Mode Software (Continued)**Table 13-2. Virtual Registers**

Class	Parameter_Id	Description
00h: Miscellaneous	00h	SMM software version (read-only)
	01h	High Memory access
	02h	Information relative to VSMs
	03h	Signature
	04h	Disable Video Trap
	05h	VSM Version
	06	Ctrl - Alt -Del
01h: XpressAUDIO	00h	Version number (read-only)
	01h	PM state
	02h	Set XpressAUDIO I/O base
	03h	Set MIDI I/O base
	04h	CPU Usage
	05h	Codec Type
	06h	State Index
	07h	State Data
	08h	Native Audio IRQ
	09h	Native Audio Status Ptr
02h: SoftVGA	01h	Video memory size (512 KB units) See Section 14.2 “Virtual Video” on page 130.
	02h	Video PLL Ref (512 KB units)
03h Advanced Power management	00h	Report Event

System Management Mode Software (Continued)**Table 13-2. Virtual Registers (Continued)**

Class	Parameter_Id	Description
04h: Power Management	00h	PM Mode
	01h	Power State
	02h	Doze Timeout
	03h	Standby Timeout
	04h	Suspend Timeout
	05h	Keyboard Timeout
	06h	Mouse Timeout
	07h	Video Timeout
	08h	Disk Timeout
	09h	Floppy Timeout
	0Ah	Serial Timeout
	0Bh	Parallel Timeout
	0Ch	IRQ Wakeup Mask
	0Dh	Suspend Modulation
	0Eh	Video Speedup
	0Fh	IRQ Speedup
	10h	Wakeup SMI Mask
	11h	Inactivity Control
	12h	Mouse Control
	13h	Resume-On-Ring
	14h	SCI Control (ACPI Only)
	15h	SCI Routing (ACPI Only)
	16h	Vector Segment (ACPI Only)
	17h	Vector Offset (ACPI Only)
06h: TV Encoder	00h	Encoder Type
	01h	Callback Mask
	02h	TV mode
	03h	Position
	04h	Brightness
	05h	Contrast
	06h	Output
	10h-1Dh	Timing Values
07h: External Amplifier	00h	External Amp Version
	01h	Amplifier Power
	02h	Amplifier Type

System Management Mode Software (Continued)**Table 13-2. Virtual Registers (Continued)**

Class	Parameter_Id	Description
0Ah: Battery power	00h	Number battery units
	01h	Battery select
	02h	AC status
	03h	Battery status
	04h	Battery percentage
	05h	Battery time
0Bh: OHCI	00h	Set LED
0Ch: Keyboard	00h	Keyboard Present
	01h	Scancode
	02h	Mouse Present
	03h	Mouse Buttons
	04h	Mouse X-Y Coordinates
0Dh: Video DDC	00h	DDC Enable/Disable
	01h	DDC I/O
0Eh: Debugger	N/A	Not implemented/Class reserved
0Fh: Virtual UART	00h	UART address index
	01h	UART IRQ index
	02h	UART fill RCVBUF input side
	03h	UART COM port write
	04h	UART COM port read
010h: COP8	00h	HIB enable/disable
	01h	HIB send packet
	02h	Read byte from COP8 UART
	03h	Read COP8 version
	04h	Read 8 byte serial number
	05h	Read POST button pressed status
011h: OWL	00h	DAC (Backlight) Control
	01h	GPIO Control

System Management Mode Software (Continued)

13.10 SYSTEM CALLS AND MACROS

The use of system calls alleviates most requirements for a VSM to directly access a system structure. However, access to system data that is not within a VSM's data segment is occasionally necessary. The most common examples are fields within the SMI header and interrupted process' registers. Macros are provided to access (read and write) these fields. This allows the System Manager to enforce access restrictions. For example, a VSM responding to an asynchronous event is prevented from modifying an SMI header or trapped registers, since to do so is a programming error. Furthermore, since there may be several contexts active due to nesting of SMIs, the System Manager ensures the correct structure instance is accessed.

All processor states associated with an interrupted context will be stored either in the VSM's header or on its stack. This design simplifies the management of multiple VSM contexts.

There are no system call functions directly accessible to the programmer. System calls are always invoked by a corresponding macro. This provides the flexibility to change the implementation of a macro without changing the programming interface. At a minimum, a VSM must invoke `SYS_GET_NEXT_MSG` to retrieve messages. Most VSMs also invoke `SYS_REGISTER_EVENT` one or more times, typically at initialization time. The other system calls are used in specialized cases.

System Management Mode Software (Continued)**Table 13-3. System Calls**

System Call	Meaning
SYS_REGISTER_EVENT	Associates an event to a VSM
SYS_UNREGISTER_EVENT	Disassociates an event from a VSM
SYS_GET_NEXT_MSG	Retrieves the next message from the VSM's message queue
SYS_INSERT_MSG	Inserts a message into the message queue
SYS_BROADCAST_MSG	Sends a message to one or more VSMs
SYS_PASS_EVENT	Sends unhandled event to next registered VSM
SYS_SW_INTERRUPT	Performs a software interrupt call to the system BIOS
SYS_YIELD_CONTROL	Gives other VSMs a chance to run
SYS_GET_SYSTEM_INFO	Gets information about the system on which the VSM is currently executing

System Management Mode Software (Continued)

SYS_REGISTER_EVENT

Unless a VSM is written only to perform some action at initialization with no run-time processing requirements, it must invoke SYS_REGISTER_EVENT to request notification of desired events. Otherwise, the only events the VSM receives are those broadcast by the System Manager or other VSMs.

The SYS_REGISTER_EVENT macro is used to control a VSM's access to the specified event. The parameters passed are dependent on the event being registered. Re-issuing the SYS_REGISTER_EVENT request with parameters0-2 containing the same values as the prior event registration may change the priority of an event. Parameter3 would contain the new priority. Event codes and parameters are listed in Table 13-4.

Parameter0:	Event Code
Parameter1:	Dependent on the event
Parameter2:	Dependent on the event
Parameter3:	Priority

System Management Mode Software (Continued)**Table 13-4. Event Registration Parameters**

Event Code	Parameter1	Parameter2
EVENT_TIMER	Interval	Handle (optional)
EVENT_AUDIO		
EVENT_GRAPHICS		
EVENT_EXT_VBLANK		
EVENT_USB		
EVENT_ACPI_TIMER		
EVENT_ACPI		
EVENT_IRQ	IRQ (3 or 4 only)	
EVENT_A20		
EVENT_GAMEPORT		
EVENT_NMI		
EVENT_IO_TRAP	I/O base	Length of range (bytes)
EVENT_IO_TIMEOUT	I/O base	16 MSBs = Range (bytes) 16 LSBs = Interval (sec.)
EVENT_MEMORY_TRAP	Memory base	Length of range (bytes)
EVENT_MEMORY_TIMEOUT	Memory base	16 MSBs = Range (bytes) 16 LSBs = Interval (sec.)
EVENT_VIDEO_INACTIVE	Interval (seconds)	
EVENT_GPIO	GPIO pin number	0 = Falling edge 1 = Rising edge
EVENT_SOFTWARE_SMI	S/W SMI code (EAX)	Mask for AL (e.g. FFh=xx)
EVENT_GT1	Interval (seconds)	Mask of reload events
EVENT_PCI_TRAP	PCI Address	Address mask
EVENT_VIRTUAL_REGISTER	Virtual register class	

System Management Mode Software (Continued)

The EVENT_IO_TRAP and EVENT_IO_TIMEOUT events are not completely general-purpose. The intention is to provide a common interface to the variety of I/O traps and timers that are supported by the hardware. If the I/O base does not match one of the traps or timeouts directly supported by the Southbridge, the System Manager attempts to use the User-Defined Trap hardware. However, the User-Defined Traps are a limited resource and there are range restrictions, so it is possible for a registration request to be rejected.

Example of usage:

```
SYS_REGISTER_EVENT(EVENT_IO_TRAP, 0x1F6, 0x02, Priority); // Trap Primary IDE
```

The traps and timeouts for which there is direct hardware support are as follows in Table 13-5.

System Management Mode Software (Continued)**Table 13-5. Hardware Supported Traps and Timeouts**

Device	I/O Base Address	Range (Bytes)	Timeout Units
Port B (A20M/reset)	92h	1	N/A
Game port	200h	2	N/A
Audio	220h/240h/260h/280h 300h/330h (MIDI) 00h/C0h (DMA)	10h 2 10h each	N/A
LPT1 & LPT2	378h/278h	8 each	
COM2	2F8h	8	N/A
COM4	2E8h	8	N/A
Primary IDE	1F6h	2	Seconds
Secondary IDE	176h	2	Seconds
Floppy	3F5h or 375h	1	Seconds
LPT1/LPT2/COM1-4	Standard PC/AT addresses	8 each	Seconds
Keyboard/Mouse	60h/64h (62h/66h)	N/A	Seconds
PCI	0CFCh	4	N/A

SYS_UNREGISTER_EVENT

The SYS_UNREGISTER_EVENT macro cancels a prior event registration. The parameters passed should be the same values passed when the event was registered. See Table 13-4 on page 110.

Parameter0: Event Code
 Parameter1: Dependent on the event
 Parameter2: Dependent on the event

SYS_GET_NEXT_MSG

The SYS_GET_NEXT_MSG macro retrieves the next message from a VSM's message queue. The macro returns as a value an *unsigned short*, which is the message code. The macro takes one argument, a buffer to receive the message parameters. The parameter buffer address must be relative to the DS segment. Therefore, the buffer must not be a local variable, unless the VSM's memory model is such that DS = SS.

If the message queue is empty, the macro returns control to the System Manager. The VSM regains control again when there is a new message in its message queue.

No Parameters

Example of usage:

```
unsigned shortMessage;
unsigned longParams[6];
Message = SYS_GET_NEXT_MSG(&Params);
```

System Management Mode Software (Continued)**SYS_BROADCAST_MSG**

The SYS_BROADCAST_MSG macro inserts a message into one or more VSM message queue(s). Parameter2 specifies the type of VSM into whose message queue the message is inserted. If there are multiple VSMs of that type, the highest priority VSM receives the message. If that VSM chooses not to handle the message, it may invoke SYS_PASS_EVENT to notify the next highest priority VSM. A special type VSM, VSM_ANY, specifies that the message is to be sent to all VSMs. This system call is typically used to notify VSMs of a change in the system state, such as power management level or an imminent warm/cold boot. The other main usage is to instruct VSMs to save state in preparation for a Suspend-To-Disk.

Parameter0: Message Code
 Parameter1: Offset of parameter buffer
 Parameter2: VSM type

Example of usage:

```
// Tell all VSMs to save the state of their associated hardware.
SYS_BROADCAST_MSG(MSG_SAVE_STATE, &ParamsBuffer, VSM_ANY);
```

SYS_PASS_EVENT

If an event retrieved from the message queue is not handled by a VSM, this macro instructs the System Manager to pass the event along to the next VSM that has registered for this event. This scenario typically occurs when multiple VSMs register for the same event. When that event occurs, the VSM polls a status register and determines if the event is for that VSM. If not, the VSM should pass the event in order to give other VSMs an opportunity to handle the event. If a VSM fails to do so, the event is effectively ignored, which may result in unintended behavior.

Parameter0: Event Code
 Parameter1: Original Parameter1 from message
 Parameter2: Original Parameter2 from message
 Parameter3: Original Parameter3 from message

Example of usage:

```
SYS_PASS_EVENT(MSG_IO_TRAP, Param1, Param2, Param3);
```

SYS_SW_INTERRUPT

The SYS_SW_INTERRUPT system call performs a software interrupt. Some interrupts may not be supported. Typical interrupts are INT 13h and INT 15h. The interrupt vector to which control is transferred depends on when the system call is executed. The System Manager snapshots the interrupt vector once when the SMM software is installed (early POST), and again at the end of POST. This implies that caution must be used if this system call is used in response to a MSG_INITIALIZE with the parameter indicating “early POST”. Some vectors may not exist since the BIOS may not have initialized the interrupt vector yet. Also, some interrupts perform INT instructions, so those vectors must be initialized too.

Parameter0: INT number
 Parameter1: Offset of a register structure

Examples of usage:

```
Regs Registers;
Registers.R_AH = 0x88;
SYS_SW_INTERRUPT(0x15, &Registers); // Perform INT 15h function 0x88
```

System Management Mode Software (Continued)**SYS_YIELD_CONTROL**

The SYS_YIELD_CONTROL system call returns control to the System Manager in order to give other VSMs with lower priority events pending a chance to execute. This is typically used in cases where a VSM is performing a lengthy procedure, such as waiting for an IDE drive to become ready. Rather than continually poll, other useful work can be done by scheduling another VSM.

Parameter0: Maximum number of milliseconds to yield

Note: If Parameter0 = 0x00000000, then no particular preemption interval is guaranteed.

Example of usage:

```
SYS_YIELD(500); // Yield control for up to 500 ms
```

SYS_GET_SYSTEM_INFO

The SYS_GET_SYSTEM_INFO system call is used to get information about the system on which the VSM is currently executing. This is useful in that it allows a VSM to tailor its behavior, allowing a VSM to execute on a variety of platforms.

Parameter0: Offset to a hardware structure

```
typedef struct {
    ULONG Chipset_Base;           // Southbridge's PCI Address (e.g. 0x80009000)
    USHORT Chipset_ID;           // Southbridge's PCI Device ID (see DEVICE_ID_55x0 below)
    USHORT Chipset_Rev;          // Southbridge's Revision
    USHORT CPU_ID;               // CPU's PCI Device ID
    USHORT CPU_Revision;         // CPU's version DIR0::DIR1
    USHORT CPU_MHz;              // Units = MHz
    ULONG SystemMemory;          // Units = bytes
} Hardware;

#define DEVICE_ID_MEDIAGX      0x0001
#define DEVICE_ID_5510        0x0000
#define DEVICE_ID_5520        0x0002
#define DEVICE_ID_5530        0x0100
#define DEVICE_ID_5540        0x0301
```

Example of usage:

```
Hardware SysInfo;
SYS_GET_SYS_INFO(&SysInfo);
```

13.11 MESSAGES

Messages are retrieved from a VSM's message queue with the GET_NEXT_MSG macro. The GET_NEXT_MSG macro takes one parameter, the address of an array of *unsigned longs*. Messages may have zero to MAX_MSG_PARAM parameters associated with them, depending on the message. The GET_NEXT_MSG macro fills the array with the parameters, if any, associated with the retrieved message. The macro returns an *unsigned short*, which is the next highest priority message pending for the VSM.

Note that a VSM receives most of the following messages at some time during its execution. The only message that a VSM has control over is MSG_EVENT. A VSM receives MSG_EVENT only for events for which it registers. A VSM is free to ignore some messages. For example, a VSM may not have hardware state to save or restore, so it would ignore the MSG_SAVE_STATE and MSG_RESTORE_STATE messages.

System Management Mode Software (Continued)**Table 13-6. Messages**

Message	Meaning
MSG_INITIALIZE	Performs initialization (called at SMM initialization & end of POST)
MSG_SHUTDOWN	Prepares for system shutdown
MSG_SAVE_STATE	Saves entire state of device(s) controlled by VSM
MSG_RESTORE_STATE	Restores saved state of device(s) controlled by VSM
MSG_SET_POWER_STATE	Sets device(s) to specified power state
MSG_EVENT	A registered event has occurred
MSG_QUEUE_OVERFLOW	The message queue is full
MSG_WARM_BOOT	Prepares for a warm boot

MSG_INITIALIZE

This message is sent to each VSM twice. The first message (with Parameter0 = 0) is sent when the SMM software is initialized early when VSA is first installed. Examples of the types of initialization performed in early POST are:

- Registration of events
- Hardware initialization
- Initialization of virtual register values

The System BIOS signals the end of POST immediately before booting the operating system. In response to this signal, the System Manager sends a second MSG_INITIALIZE message (with Parameter0 = 1) to each VSM. This allows each VSM to perform initialization that may not have been appropriate at the earlier initialization.

Parameter0: 0 if "early POST" initialization
 1 if "end of POST" initialization

MSG_SHUTDOWN

This message is sent to each VSM immediately before the system is reset. It provides the opportunity to shut down the system hardware gracefully by disabling hardware or waiting for certain activity to complete. In particular, all SMI sources should be disabled. Processing for this event is typically the same as that for MSG_WARM_BOOT.

No Parameters

MSG_SAVE_STATE

This message is sent to each VSM in preparation for Save-To-Disk or Save-To-RAM. All information that is required for restoring the device(s) controlled by the VSM must be saved to the VSM's data segment. This message is sent at most one time by the Save-To-Disk or Save-To-RAM VSM.

No Parameters

MSG_RESTORE_STATE

This message is sent to each VSM during a Resume-From-Disk or Resume-From-RAM. All devices handled by the VSM should be restored using the state information that was saved during the handling of the MSG_SAVE_STATE message. This message is sent at most one time by the Save-To-Disk or Save-To-RAM VSM, and is only sent if a prior MSG_SAVE_STATE has been issued.

No Parameters

MSG_SET_POWER_STATE

This message is sent to each VSM when the power level of the device(s) controlled by the VSM is to be changed. For example, before the system enters Standby, all VSMs are sent the MSG_SET_POWER_STATE message with Parameter0 set to PM_STANDBY. Valid power states are shown in Table 13-7.

Parameter0: Power State

System Management Mode Software (Continued)**Table 13-7. Power States**

Value	ACPI	APM/Legacy	Description
0000	S0/C0	Full on	Full ON
0001	S0/C1		HLTed with interrupts enabled; cache(s) maintained
0002	S0/C2		Same as S0/C1
0003	S0/C3		Same as S0/C2 but cache(s) don't snoop
0100	S1	Standby	CPU & RAM context preserved
0200	S2	Save-To-RAM	RAM context preserved; CPU clocks stopped
0300	S3		RAM context preserved; All clocks stopped
0400	S4		RAM & CPU context <i>not</i> preserved
0401	S4BIOS	Save-To-Disk	RAM & CPU context <i>not</i> preserved
0500	S5		Soft OFF

MSG_EVENT

MSG_EVENT is the most common message. When a registered event occurs, this message is sent to the VSM that has registered for that event with the highest priority. If a VSM handles the event, the VSM does not have to do anything explicitly to notify the System Manager. However, if a VSM chooses not to handle the event, it **MUST** invoke the SYS_PASS_EVENT system call to give other VSMs that have registered for the same event an opportunity to handle the event. If no VSM handles the event (all registered VSMs invoked the SYS_PASS_EVENT system call), an error is signaled. See Section 13.4 “Events” on page 95.

Parameter0: Event code
 Parameter1: Dependent on event
 Parameter2: Dependent on event
 Parameter3: Dependent on event
 Parameter4: Dependent on event

MSG_QUEUE_OVERFLOW

The System Manager always reserves one message queue entry for this message. If an event occurs that would overflow the message queue, a MSG_QUEUE_OVERFLOW message is sent instead of the original message. The original message code is passed in Parameter0. If any more events occur before the MSG_QUEUE_OVERFLOW message is retrieved, those events are discarded and an error is signaled.

This message is primarily designed to notify a VSM of event overrun. In many cases, nothing can be done about the situation, i.e., it is an error condition. However, some VSMs may be able to change their behavior to alleviate the condition. For example, an audio VSM may choose to reduce the audio sampling rate.

Parameter0: Event code

MSG_WARM_BOOT

This message is sent to each VSM immediately before the system is warm booted. VSMs should respond to this message by preparing the related hardware. This typically means ensuring that no more SMIs are generated by the hardware. For example, the audio VSM should disable the audio hardware and wait for the output FIFO to empty. The video VSM should turn off vertical retrace SMI.

No Parameters

System Management Mode Software (Continued)**13.12 EVENTS**

The main routine of all VSMs is a message handling loop. When an event occurs, the VSM(s) that have registered for that event are sent a MSG_EVENT message with the event code in Parameter0. Other parameters may also be passed, depending on the event. A VSM is only sent a MSG_EVENT message for the events that it registered as a handler.

System Management Mode Software (Continued)**Table 13-8. Events**

Event	Meaning
EVENT_GRAPHICS	VGA I/O access, text buffer access, or vertical retrace
EVENT_VBLANK	Vertical retrace SMI
EVENT_AUDIO	Audio event
EVENT_USB	USB event
EVENT_ACPI	The ACPI timer expired
EVENT_ACPI_TIMER	The ACPI timer expired
EVENT_IO_TRAP	I/O trap
EVENT_IO_TIMEOUT	I/O timeout
EVENT_MEMORY_TRAP	Memory trap
EVENT_MEMORY_TIMEOUT	Memory trap timeout
EVENT_VIDEO_INACTIVITY	Video inactivity timer expired
EVENT_GPIO	GPIO transition
EVENT_SOFTWARE_SMI	Software SMI
EVENT_PCI_TRAP	PCI trap
EVENT_TIMER	A millisecond timer expired
EVENT_VIRTUAL_REGISTER	Virtual register access
EVENT_A20	A change in the A20 mask has occurred
EVENT_NMI	An NMI has occurred.
EVENT_GT1	General purpose timer 1 has expired

EVENT_GRAPHICS

This event is sent when there is an SMI from the graphics unit. The parameters are only valid for trapped I/O, not asynchronous graphics events.

Parameter1: Flags field (from SMM header)
Parameter2: Bits Description
 15:0 I/O address from SMM header
 1:16 Data Size field from SMM header
Parameter3: Data (if I/O write)
Parameter4: Memory Address if string I/O

EVENT_VBLANK

A vertical blank interrupt has occurred.

No Parameters

System Management Mode Software (Continued)**EVENT_AUDIO**

An audio event has occurred. The audio VSM reads the audio SMI source registers to determine the exact type of audio event. The parameters are only valid for trapped I/O, not asynchronous audio events such as audio FIFO empty.

Parameter1: Flags field (from SMM header)
 Parameter2: Bits Description
 15:0 I/O address from SMM header
 31:16 Data Size field from SMM header
 Parameter3: Data (if I/O write)
 Parameter4: Memory Address if string I/O

EVENT_USB

A USB event has occurred.

No Parameters

EVENT_ACPI

An access to an ACPI register has occurred.

Parameter1: ACPI register index
 Parameter2: 01h = BYTE access
 03h = WORD access
 0Fh = DWORD access
 bit 7 set if access is an I/O write
 Parameter3: data (valid only if access is a write)

EVENT_ACPI_TIMER

This event is sent each time the MSB of the ACPI timer register toggles. The 24-bit ACPI counter increments at 14.31818/4 MHz. Therefore, when enabled, the ACPI timer generates an SMI every 2.3435 seconds.

No Parameters

EVENT_IO_TRAP

An access to a registered I/O location has occurred.

Parameter1: I/O address
 Parameter2: 01h = BYTE access
 03h = WORD access
 0Fh = DWORD access
 bit 7 set if access is an I/O write
 Parameter3: Data (valid only if access is a write)

EVENT_IO_TIMEOUT

An access to a trapped I/O location has not occurred for the interval specified when this event was registered.

Parameter1: I/O address

EVENT_MEMORY_TRAP

An access to a registered memory location has occurred.

Parameter1: Memory address (physical)
 Parameter2: 01h = BYTE access
 03h = WORD access
 0Fh = DWORD access
 bit 7 is set if access is an memory write
 Parameter3: Data (valid only if access is a write)

EVENT_MEMORY_TIMEOUT

An access to a trapped memory location has not occurred for the interval specified when this event was registered.

Parameter1: Memory address

System Management Mode Software (Continued)**EVENT_VIDEO_INACTIVITY**

This event is sent if the video has been inactive for the interval specified when this event was registered.

No Parameters

EVENT_GPIO

A transition has occurred on a GPIO pin. Event GPIO symbols are defined in VSA2.H for the programmer's convenience:

Parameter1: GPIO number
 Parameter2: 0 = falling edge
 1 = rising edge

EVENT_SOFTWARE_SMI

The receipt of this event signals that a software SMI has occurred. A VSM is only sent this message for software SMIs of the value for which it registered. The SMI code is the value in AX at the time the SMI occurred. This event is provided primarily for compatibility with BIOSs that are not VSAll technology aware. Newer BIOSs should use virtual registers for communication with VSA.

Parameter1: SMI code

EVENT_PCI_TRAP

An access to a PCI configuration space has occurred. The VSM is only sent this message for PCI functions that were registered.

Parameter1: PCI Config Address
 Parameter2: 01h = BYTE access
 03h = WORD access
 0Fh = DWORD access
 bit 7 set if access is an I/O write
 Parameter3: Data (valid only if access is a write)

EVENT_TIMER

A VSM may request to be notified every n milliseconds. This event is generated when the requested time interval has expired. The timer request remains active until the VSM explicitly cancels the request, via SYS_UNREGISTER_EVENT. Since a VSM may request more than one timer to run concurrently, a handle may be specified when each timer is registered. When a timer elapses, the corresponding handle is passed as a parameter to distinguish among multiple timers.

Parameter1: Interval (bits 30:0)
 Bit 31: Reserved
 Parameter2: Handle

EVENT_VIRTUAL_REGISTER

An access to a virtual register has occurred. A VSM is only sent this message for virtual register accesses for class(es) for which it registered.

Parameter1: Virtual register (Bits 15:8 = VR class; Bits 7:0 = class index)
 Parameter2: 0 = WORD read
 1 = WORD write
 Parameter3: Data (valid only if access is a write)

EVENT_A20

A change in the A20 mask has occurred. This may be either by bit 1 of I/O port 92h, or by issuing a keyboard controller command.

No Parameters

EVENT_NMI

An NMI has occurred.

No Parameters

System Management Mode Software (Continued)

EVENT_GT1

The general purpose timer 1 has expired.

Parameter1: Timeout in seconds
Parameter2: Mask of devices that can reset GT1
 Bit 0 = Primary Hard Disk
 Bit 1 = Floppy Disk
 Bit 2 = Parallel/Serial
 Bit 3 = Keyboard/Mouse
 Bit 4 = Secondary Hard Disk

13.13 MACROS

In addition to the system call macros documented in Section 13.6 “System Calls” on page 95, there are other macros available to the VSM programmer. These are designed to achieve certain required functionality without the burden of understanding the implementation details. For example, if a VSM requires information from the SMM header, the method of reading the SMM header location is CPU dependent.

System Management Mode Software (Continued)**Table 13-9. Macros**

Macro	Usage
ENTER_CRITICAL_SECTION	Disable nested SMIs
EXIT_CRITICAL_SECTION	Enable nested SMIs
SET_VIRTUAL_REGISTER	Write a virtual register
GET_VIRTUAL_REGISTER	Read a virtual register
SET_REGISTER	Store a value to an interrupted general purpose register
GET_REGISTER	Get a trapped general purpose register
SET_HEADER_DATA	Set a field in the SMM header
GET_HEADER_DATA	Get a field from the SMM header
READ_PCI_BYTE	Read a byte from PCI configuration space
READ_PCI_WORD	Read a WORD from PCI configuration space
READ_PCI_DWORD	Read a DWORD from PCI configuration space
READ_PCI_DWORD_NO_TRAP	Disable trapping on the PCI address, perform a PCI configuration space read, then re-enable PCI trapping on the address.
READ_PCI_WORD_NO_TRAP	
READ_PCI_BYTE_NO_TRAP	
WRITE_PCI_BYTE	Write a byte to PCI configuration space
WRITE_PCI_WORD	Write a WORD to PCI configuration space
WRITE_PCI_DWORD	Write a DWORD to PCI configuration space
WRITE_PCI_DWORD_NO_TRAP	Disable trapping on the PCI address, perform a PCI configuration space write, then re-enable PCI trapping on the address.
WRITE_PCI_WORD_NO_TRAP	
WRITE_PCI_BYTE_NO_TRAP	

System Management Mode Software (Continued)**13.13.1 ENTER_CRITICAL_SECTION & EXIT_CRITICAL_SECTION**

The critical section macros are used to mark the beginning and end of VSM code that should not be interrupted. Typically, this is brief sections of code that are very timing sensitive. For example, if a VSM creates a software-generated pulse of a precise duration, the VSM must not be interrupted. If the VSM is interrupted, the pulse would be longer than desired. Surrounding such code with critical section macros prevents any interruption.

Implementing critical sections ensures that structures are updated atomically.

Example of usage:

```
ENTER_CRITICAL_SECTION();
// timing sensitive code goes here
EXIT_CRITICAL_SECTION();
```

SET_VIRTUAL_REGISTER

The SET_VIRTUAL_REGISTER macro is used to write to a virtual register from within a VSM. While it is possible to perform I/O operations to accomplish the same effect, using this macro allows the System Manager to perform the operation more efficiently.

Example of usage:

```
SET_VIRTUAL_REGISTER(Virtual_Class, Virtual_Index, data);
```

GET_VIRTUAL_REGISTER

The GET_VIRTUAL_REGISTER macro is used to read to a virtual register from within a VSM. While it is possible to perform I/O operations to accomplish the same effect, using this macro allows the System Manager to perform the operation more efficiently.

Example of usage:

```
Data = GET_VIRTUAL_REGISTER(Virtual_Class, Virtual_Index);
```

13.13.2 General Purpose Register Access**SET_REGISTER**

The SET_REGISTER macro is used to modify the interrupted (non-SMM) task's registers.

Parameter0: Register name
Parameter1: Data value

Note: This system call is only valid and meaningful when servicing a *synchronous* event (EVENT_SOFTWARE_SMI, EVENT_IO_TRAP, EVENT_MEMORY_TRAP). An attempt to use it when servicing an asynchronous event is a programming error.

Valid register names are:

```
R_EAX, R_AX, R_AH, R_AL
R_EBX, R_BX, R_BH, R_BL
R_ECX, R_CX, R_CH, R_CL
R_EDX, R_DX, R_DH, R_DL
R_ESI, R_SI
R_EDI, R_DI
R_EBP, R_BP
R_ESP, R_SP
R_DS
R_SS
R_ES
R_FS
R_GS
```

Example of usage:

```
SET_REGISTER (R_EAX, 0x123456578); // Set caller's EAX to 0x12345678
```

System Management Mode Software (Continued)**GET_REGISTER**

The GET_REGISTER macro is used to retrieve a general-purpose register from the interrupted (non-SMM) task.

Parameter0: Register name

Note: This system call is only valid and meaningful when servicing a *synchronous* event (EVENT_SOFTWARE_SMI, EVENT_IO_TRAP, EVENT_MEMORY_TRAP). An attempt to use it when servicing an asynchronous event is a programming error.

Example of usage:

```
Register_EDX = GET_REGISTER (R_EDX); // Get caller's EDX
```

SET_HEADER_DATA

The SET_HEADER_DATA macro is used to write data to the SMI header from the interrupted (non-SMM) task. The size of the field written is determined by the field.

Parameter0: Name of SMM header field

Parameter1: Data

Note: This system call is only valid and meaningful when servicing a *synchronous* event (EVENT_SOFTWARE_SMI, EVENT_IO_TRAP, EVENT_MEMORY_TRAP). An attempt to use it when servicing an asynchronous event signals an error.

Valid SMM header field names are defined in Section “GET_HEADER_DATA”.

Example of usage:

```
SET_HEADER_DATA(CURRENT_EIP, GET_HEADER_DATA (NEXT_EIP));
```

GET_HEADER_DATA

The GET_HEADER_DATA macro is used to read data from the SMI header from the interrupted (non-SMM) task. The size of the field retrieved is determined by the field.

Parameter0: Name of SMM header field

Note: This system call is only valid and meaningful when servicing a *synchronous* event (EVENT_SOFTWARE_SMI, EVENT_IO_TRAP, EVENT_MEMORY_TRAP). An attempt to use it when servicing an asynchronous event signals an error.

Valid SMM header field names are:

```
NEXT_EIP
CURRENT_EIP
CS_LIMIT
CS_BASE
CS_SELECTOR
CS_ATTRIBUTE
R_CR0
R_DR7
EFLAGS
DATA_SIZE
IO_ADDRESS
WRITE_DATA
```

Example of usage:

```
Address = GET_HEADER_DATA (NEXT_EIP); // Get address of trapped instruction
```

System Management Mode Software (Continued)

13.13.3 Shorthand Macros

Since the AL, AX and EAX registers are the most commonly accessed registers, specific macros are defined to facilitate readability of the code. These macros and their equivalent definitions are:

System Management Mode Software (Continued)**Table 13-10. Shorthand Macros**

Shorthand Macro	Equivalent Macro
GET_AL()	GET_REGISTER(R_AL)
GET_AH()	GET_REGISTER (R_AH)
GET_AX()	GET_REGISTER(R_AX)
GET_EAX()	GET_REGISTER(R_EAX)
SET_AL(data)	SET_REGISTER(R_AL, (unsigned char) data)
SET_AH (data)	SET_REGISTER (R_AH, (unsigned char) data)
SET_AX(data)	SET_REGISTER(R_AX, (unsigned short) data)
SET_EAX(data)	SET_REGISTER(R_EAX, (unsigned long) data)
SET EFLAGS (data)	SET_HEADER_DATA (EFLAGS, (unsigned long) data)

13.13.4 PCI Register Access

All PCI accesses must be naturally aligned. Otherwise, unpredictable results will occur.

READ_PCI_BYTE

Parameter0: PCI Configuration Address

Example of usage:

```
HeaderType = READ_PCI_BYTE(0x8000930E);
```

READ_PCI_WORD

Parameter0: PCI Configuration Address

Example of usage:

```
DeviceID = READ_PCI_WORD(0x80009302);
```

READ_PCI_DWORD

Parameter0: PCI Configuration Address

Example of usage:

```
BAR0 = READ_PCI_DWORD(0x80009310);
```

System Management Mode Software (Continued)**READ_PCI_DWORD_NO_TRAP****READ_PCI_WORD_NO_TRAP****READ_PCI_BYTE_NO_TRAP**

These macros disable trapping on the PCI address, perform a PCI configuration space read, then re-enable PCI trapping on the address. This is useful for accessing hardware directly and avoiding any trapping that may be in place.

Parameter0: PCI Configuration Address

Example of usage:

```
BAR1 = READ_PCI_DWORD_NO_TRAP (0x800093144);
```

WRITE_PCI_BYTE

Parameter0: PCI Configuration Address

Parameter1: Data to write

Example of usage:

```
WRITE_PCI_BYTE(0x8000933D, InterruptPin);
```

WRITE_PCI_WORD

Parameter0: PCI Configuration Address

Parameter1: Data to write

Example of usage:

```
WRITE_PCI_WORD(0x80009304, Command);
```

WRITE_PCI_DWORD

Parameter0: PCI Configuration Address

Parameter1: Data to write

Example of usage:

```
WRITE_PCI_DWORD(0x80009314, BAR1);
```

WRITE_PCI_DWORD_NO_TRAP**WRITE_PCI_WORD_NO_TRAP****WRITE_PCI_BYTE_NO_TRAP**

This macro disables trapping on the PCI address, performs a PCI DWORD write, then re-enables PCI trapping on the address. This is useful for forwarding trapped PCI writes to the hardware.

Parameter0: PCI Configuration Address

Parameter1: Data to write

Example of usage:

```
WRITE_PCI_DWORD_NO_TRAP (0x80009314, BAR1);
```

System Management Mode Software (Continued)

14.0 Virtual Hardware

The Geode family of devices and the XpressROM SMM software make it possible to emulate hardware functionality. The firmware responds to SMIs generated by specialized hardware and provides virtual devices such as XpressAUDIO, XpressGRAPHICS, RTC/CMOS devices, and UARTs.

14.1 VIRTUAL AUDIO

The XpressROM SMM software optionally includes support for XpressAUDIO. XpressAUDIO provides the support for industry standard 16-bit stereo synthesis and OPL3 emulation. Support is also included for an AC97 Specification Revision 1.3, 2.0, and 2.1 compatible codec. XpressAUDIO support is selected during XpressROM configuration.

The audio support is virtualized using the standard audio legacy register interface and virtual registers that define the initialization and the environment in which the XpressAUDIO will operate. (Refer to Section 13.9 “Virtual Registers” on page 101 for further information on virtual registers.) These registers listed in Table 14-1 are intended for use during initialization. Writing them at other times may have unpredictable results. After initialization, the preferred audio programming interface is the standard audio legacy register interface.

Table 14-1. Virtual Audio Registers

Register Index	Register Name or Use	Access	Description
0x0100	Audio Version	Read Only	Returns audio version in AX.
0x0101	Codec Type	Read Only	Returns codec version.
0x0102	Audio PM State	Read	Returns current audio power management state.
		Write	Sets the audio power management state. 0 = On, 5 = Off. Default is on (0). The audio PM states are similar to, but do not map one to one with the ACPI PM states. This register is written during initialization and used by the Windows driver to provide ACPI compliance. Writing to this register can cause unpredictable behavior and is not generally recommended.
0x0103	16-Bit Sound I/O Base Address	Read	Returns current 16-bit sound base address.
		Write	Sets 16-bit sound base address to desired value from set {0, 220h, 240h, 260h, 280h}. Default is 220h.
0x0104	MIDI Base Address	Read	Returns current MIDI base address. 0 means MIDI is off.
		Write	Sets MIDI base address to desired value from set {0, 300h, 330h}. Default is off (0).
0x0105	Maximum Sampling Rate	Read	Returns current maximum sampling rate. 0 means no limit.
		Write	Sets maximum audio sampling rate. 0 means no limit. Default is 0.
0x0106	Reserved	Read	Returns 0.
		Write	Not supported.
0x0107	Reserved	Read	Returns 0.
		Write	Not supported.
0x0108	Audio IRQ ¹	Read	Returns current audio IRQ value.
		Write	Sets audio IRQ to desired value from set {5,7,9,10}. Default IRQ is 5.
0x0109	Audio 8-bit DMA Channel ¹	Read	Returns current 8-bit DMA channel value.
		Write	Sets 8-bit DMA channel to desired value from set {0,1,3}. Default 8-bit DMA is 1.
0x010a	Audio 16-bit DMA Channel ¹	Read	Returns current 16-bit DMA channel value.
		Write	Sets 16-bit DMA channel to desired value from set {5,6,7}. Default 16-bit DMA is 5.
0x0700	External Amplifier Power (Optional hardware)	Read	Returns current state of external amplifier power-down signal.
		Write	Sets the external amplifier power-down signal. Used for audio mute and un-mute.

1. After initialization this interface should not be used. Instead, use the standard audio legacy register interface.

Virtual Hardware (Continued)**Example 1:**

; Read the audio version

```

mov    dx, 0AC1Ch    ; The index register address
mov    ax, 0100h     ; The index of the version register
out    dx, ax        ; Set the index
mov    dx, 0AC1Eh    ; The data register address
in     ax, dx        ; Read the indexed register data

```

Example 2:

; Set the audio IRQ

```

mov    dx, 0AC1Ch    ; The index register address
mov    ax, 0103h     ; The index of the audio IRQ register
out    dx, ax        ; Set the index
mov    dx, 0AC1Eh    ; The data register address
mov    ax, 5         ; The desired IRQ number
out    dx, ax        ; Write the audio IRQ value to the audio IRQ register

```

14.2 VIRTUAL VIDEO

Using Virtual System Architecture technology, the XpressROM firmware optionally includes support for SoftVGA. SoftVGA takes advantage of embedded hardware capability to provide a VGA graphics subsystem that is compatible with the IBM VGA with additional functionality to support the VESA BIOS extensions.

The SoftVGA virtualizes the video interface by supporting the standard VGA interface with the VESA BIOS extensions and a single indexed register. This register operates as an indexed pair such that the register index is written to 0AC1Ch, and the data is then read from or written to 0AC1Eh. This register is used to define three initialization parameters: the size of the video memory in 128 KB segments, whether SoftVGA is to be enabled, and if enabled, whether it is to be initially active. This register should be written only at initialization. Writing this register after initialization creates unpredictable results. After initialization, the preferred video interface is the standard VGA interface.

The initialization parameters fit within one byte and have the format as shown in Table 14-2.

Table 14-2. Virtual Video Register

Register Index	Register Name or Use	Access	Description
0x0200	Video initialization	Write Only	Set the video initialization parameters. Default is SoftVGA installed and on, using the memory size specified during the XpressROM configuration. The bit formats are: Bit 7: 0 = SoftVGA is initially active. 1 = SoftVGA is initially inactive (used for dual-monitor support). Bits [6:1]: Amount of video memory in 128 KB blocks. Bit 0: 0 = SoftVGA is not to be installed. 1 = Install SoftVGA.

Example 1:

; Set the SoftVGA initialization parameters

```

mov    dx, 0AC1Ch    ; The index register address
mov    ax, 0200h     ; The index of the initialization register
out    dx, ax        ; Set the index
mov    dx, 0AC1Eh    ; The data register address
mov    ax, 0029h     ; SoftVGA installed and initially active, 2.5 MB video memory

```

Virtual Hardware (Continued)

14.3 VIRTUAL RTC/CMOS

The XpressROM SMM software can optionally provide functionality, which emulates an RTC/CMOS device. This is done by registering for two types of events at initialization; a timer to be notified every second (1000 ms) and I/O access to 70h and 71h. During the operation, two other timers are used for the following events; periodic interrupt and update in progress.

During POST, the date and time in the V-RTC is initialized to midnight, January 1, 1990. Other initial values are:

- Status_A = 26h
- Status_B = 02h
- Status_C = 00h
- Status_D = 80h
- All other V-CMOS locations = 00h

Note that POST will initialize various “legacy” locations as well.

15.0 Power Management

There are multiple industry-standard power management (PM) methods to be supported. Each of these methods share common PM concepts such as device timeouts, power states, and power state transition diagrams. Power management methods typically differ primarily in the API and in some of the details, such as the number of power states and the types of power state transitions that are supported.

The XpressROM power management is modularized relative to SMM in a carefully defined manner. For example, the API for Advanced Power Management (APM) is implemented in an APM-specific VSM. The setting of a particular power state (e.g. Standby), however, is common among several PM methods. Rather than duplicate power state logic in each VSM, such common functionality is implemented in a generic PM VSM. Other PM VSMs leverage this common code by communicating its requirements to the generic PM VSM via virtual registers. This distributed architecture results in smaller overall code size when multiple PM schemes are supported. In addition, when a change (e.g., a bug fix or feature enhancement) is made to the generic PM VSM, *all* of the PM VSMs benefit from such changes, usually *without recompilation* of the API-specific VSMs.

The generic PM VSM handles the low-level details of power management. It controls the CPU power state directly. It also coordinates power management among other VSMs. Specifically, when the power state changes, it broadcasts a MSG_POWER_STATE message to *all* other VSMs. The new power state is passed as a parameter. This gives other VSMs a chance to put their associated hardware, if any, into the new power state. VSMs that do not control actual hardware would typically ignore these messages. However, VSMs such as XpressGRAPHICS, XpressAUDIO, a SuperI/O or battery VSM would take appropriate action. As stated earlier, this distributed design provides the ability for custom VSMs to be written to handle platform-specific power management issues without the need to modify the basic power management software.

15.1 POWER STATES

Different power management methods define the various power states differently. However, they typically have enough in common that they can be mapped onto one of the following:

- **FULL ON** - The processor is running at full speed with all peripherals at a ready state.
- **DOZE** - Peripherals are at a ready state. The processor is in a clock-throttled mode. Any user input (keyboard or mouse activity) returns the system to FULL ON in a negligible amount of time. SMIs and IRQs also return the system to FULL ON for a short duration, then the system is clock-throttled again. This ensures quick interrupt and video servicing. Transitions in and out of this state usually cannot be noticed by the user.
- **STANDBY** - Peripherals may be put in a lower power state (minimal recovery time). The processor is in its lowest power state. The state of DRAM is preserved. The video is blanked and the hard drive is spun down. Any SMI or any IRQ defined to be a wakeup event returns the system to FULL ON.
- **SUSPEND** - This state preserves the entire state of the system, either in a Suspend-to-RAM or Suspend-to-Disk mode. The processor is off, but the power to the DRAM controller remains on in the case of Suspend-to-RAM.
- **OFF** - Power to the system is turned off. The system is typically set to this state after a Suspend-to-Disk or after the operating system has made the system ready for shutdown.

15.2 PM INITIALIZATION

Initialization requirements of power management consist primarily of writing various virtual registers within the PM virtual register class. The sample code that follows demonstrates how to write a single virtual register:

```
MOV    DX, 0AC1Ch          ; virtual register index
MOV    AH, 04h             ; PM class
MOV    AL, 02h             ; Doze timeout register
OUT    DX, AX
ADD    DX, 2               ; virtual register data
MOV    AX, <doze_timeout>  ; units = seconds
OUT    DX, AX
```

Table 15-1. PM Virtual Registers

Register	Defined As	Description
0401h	PM_MODE	Defines which PM API (e.g. APM, Legacy, etc.) is currently in use. This register is written by a PM VSM. It is not initialized by the boot ROM.
01h	POWER_STATE	Defines the desired power state. This register is written by a PM VSM. It is not initialized by the boot ROM.

Power Management (Continued)**Table 15-1. PM Virtual Registers (Continued)**

Register	Defined As	Description
02h	DOZE_TIMEOUT ¹	Defines the duration of system idleness (in seconds), defined as lack of user input, after which the system is transitioned to the DOZE power state.
03h	STANDBY_TIMEOUT ¹	Defines the duration of system idleness (in seconds), defined as lack of user input, after which the system is transitioned to the STANDBY power state.
04h	SUSPEND_TIMEOUT ¹	Defines the duration of system idleness (in seconds), defined as lack of user input, after which the system is transitioned to the SUSPEND power state.
05h	KEYBOARD_TIMEOUT	Defines duration inactivity of keyboard before system is considered idle.
06h	MOUSE_TIMEOUT	Defines duration inactivity of mouse before system is considered idle.
07h	VIDEO_TIMEOUT	Defines duration inactivity of video before system is considered idle.
08h	DISK_TIMEOUT ¹	Defines the duration of hard drive idleness (in seconds), defined as lack of I/O to the IDE interface, after which the disk drive(s) are spun down.
09h	FLOPPY_TIMEOUT ¹	Defines the duration of floppy drive idleness (in seconds), defined as lack of floppy I/O, after which the floppy subsystem is put in a low power state. This type of power control is typically located in the SuperI/O chipset.
0Ah	SERIAL_TIMEOUT ¹	Defines the duration of serial port idleness (in seconds), defined as lack of COM I/O, after which the serial port subsystem is put in a low power state. This type of power control is typically located in the SuperI/O chipset.
0Bh	PARALLEL_TIMEOUT ¹	Defines the duration of parallel port idleness (in seconds), defined as lack of LPT I/O, after which the parallel port subsystem is put in a low power state. This type of power control is typically located in the SuperI/O chipset.
0Ch	IRQ_WAKEUP_MASK	A mask of IRQs (LSB = IRQ0; MSB = IRQ15) that are to be used as wakeup events from STANDBY mode. Typically these would include IRQ1 (keyboard), IRQ12 (PS/2 mouse), as well as others such as IRQ3/4 (modem) or network IRQs.
0Dh	SUSPEND_MODULATION	Defines the effective clock frequency to be used during DOZE mode. The upper byte defines the OFF time while the lower byte defines the ON time. For example, 0x0901 would yield a 10% effective clock speed.
0Eh	VIDEO_SPEEDUP	Defines the number of ms that video activity will temporarily disable clock throttling (DOZE power state). After the elapsed time, clock throttling resumes.
0Fh	IRQ_SPEEDUP	Defines the number of ms that IRQ activity will temporarily disable clock throttling (DOZE power state). After the elapsed time, clock throttling resumes.
10h	WAKEUP_SMI_MASK	
11h	INACTIVITY_CONTROL	
12h	MOUSE_CONTROL	
13h	RESUME_ON_RING	
14h	SCI_CONTROL	
15h	SCI_ROUTING	
16h	VECTOR_SEGMENT	
17h	VECTOR_OFFSET	

1. These timeout values are only applicable to legacy power management. Other power management interfaces change power states under program control rather than with inactivity timers.

16.0 Configuring/Customizing

16.1 QUICK START

- 1) Make sure you have ml.exe (masm), link.exe, and nmake in your path.
- 2) Change directory (cd) to examples\newproj (this is a user dir).
- 3) Type “nmake all”.
- 4) Answer the questions.

16.2 BUILD PROCESS:

The build process starts in the user directory by running nmake. The first time nmake runs, the configurator asks a series of questions about the platform. nmake then attempts to complete the build according to the answers. When the build successfully completes, the following is displayed:

```
#####
#_____XpressRom build complete_____#
#####
```

16.3 ENVIRONMENT VARIABLE(S)

- Path – Should include the bin and binr directories from masm 6.11 and msvc 1.52. as well as the normal files for DOS / Windows
- Include – Should the include directories for masm 6.11 and msvc.
- Lib – Should include the lib directories from masm6.11 and msvc.

16.4 THE CONFIGURATOR

The configurator is a perl script which is used to configure the XpressROM build. The build process includes a series of questions in which the system designer provides information about the board wiring, options to install, and generally how XpressROM should behave. The tool runs automatically the first time the build process takes place and on subsequent builds uses information from a saved option file:

- Debug/Test related – Debug questions are accessed by running nmake debug. Otherwise, they are not asked. These include cache defaulting to OFF instead of ON, halting on unknown interrupts, conservative memory timings, etc.
- SMM Software related questions – These include XpressAUDIO and XpressGRAPHICS.
- Hardware related – Chipset, SIO, and LCD are included in this section.
- Boot screen related – Splash screen support, summary screen support.
- General configuration – Hard drive support, floppy support, video memory used, user option ROMs, etc.

The configurator outputs the files xromcfg.opt, rules.mak, and options.inc into the user directory.

The configurator also decodes and recreates an xromcfg.opt file from an xpress.rom file. This is useful both for recreating old ROM files and for debugging problems with the ROMs.

Configuring/Customizing (Continued)

16.5 BUILD FILES

These are non-source files which are used by the build process.

- Xpress\makefile – The master makefile for the XpressROM tree - build options (targets) are discussed in Section 16.6 “BUILD OPTIONS”.
- Xpress\Xromcfg.pl – "THE CONFIGURATOR" - This file sets up the build options for XpressROM. See the configuration section for specifics.
- User\makefile – The makefile in the user directory simply sets up the user path and calls the xpress\makefile to do the build. The common targets in this makefile are the same as for the xpress\makefile.
- User\Xromcfg.opt – The configuration file used by xromcfg.pl to set up the build options. Generated by xromcfg.pl.
- User\Rules.mak – Generated by xromcfg.pl, this file gets included by the xpress\makefile when building. DO NOT EDIT THIS FILE. Any changes made in this file will be lost when xromcfg.pl is run again.
- User\Options.inc – Generated by xromcfg.pl, this file gets included by the asm files to control various options. DO NOT EDIT THIS FILE. Any changes made in this file will be lost when xromcfg.pl is run again.
- User\Userrule.mak – This is a makefile that targets may be added to. This file gets included by the user\rules.mak file if it exists.

16.6 BUILD OPTIONS

- Nmake all – builds xpress.rom. Will do a nmake configuration if needed. This is the default if no target is specified for the build. Nmake help – Shows list of targets.
- Nmake clean – Removes object files, ROM files, etc.
- Nmake cleanall – Removes all output files of build, including xromcfg.opt.
- Nmake config – Runs the configurator to set up build options.
- Nmake refresh – Uses the configurator's output file (xromcfg.opt) to rebuild the rules.mak and options.inc files.
- Nmake debug – Runs the configurator to set up build options and debugging options.
- Nmake checktools – Generates toollist.txt - contains a list of various things to help debug build problems.
- Nmake checkbuild – Generates toollist.txt the same as checktools, but also attempts to do a build at the end.
- Nmake decode – Re-creates an xromcfg.opt file which was used to build a ROM. This also generates a list of filenames which were overrides when the ROM was built.

16.7 SPLASH SCREEN

The splash screen is used to provide a vendor or platform specific graphic display at the end of POST and this includes a selectable display time before the system proceeds to operating system load.

Considerations when using:

- The XpressROM splash screen feature only supports 320x200 resolution for external video cards. The only limitation if the Geode internal graphics is used instead is the size of the image at higher resolutions, e.g. 800x600 etc.
- Option ROM or setup entry prompts are displayed during the display of the splash screen. Both Option ROM and setup entry prompts use palette 0 for the text background and palette 7 for the text foreground. These can be customized to highlight the text entries or to allow them to blend into the splash graphics display.
- RLE compression is used for all XpressROM compression and this limits the size and complexity of the splash screen chosen.

Configuring/Customizing (Continued)**16.8 SUMMARY SCREEN**

The summary screen reports the basic configuration of XpressROM with regard to a target platform.

Figure 16-1 is an example of a Summary Screen for the Dorado platform.

```

Revision: V0.14 +                               Built: 12/19/2000 12:54:34
CPU: GXlv 5.1 @ 333MHz      PCI: 33MHz      Multiplier: 10x      SC2200 Rev: B
Memory: 61440k @ 93MHz      CAS: 2      SDRAM Divisor: 2.5      Shift SDCLK: 1.5
Floppy A: 1.44M  Drive 80: 4311MB  COM1: 03F8  LPT1: 0378  XpressROM: V3.1.3
RTC: Present      Drive 81: 4311MB  COM2: 02F8  LPT2: 0378  VSA: 0200
USB: Legacy      Drive 82: 4311MB  COM3: 03F8  LPT3: 0378  VBIOS: 040B
PM: Legacy / APM Drive 83: 4311MB  COM4: 02F8  MTEST: ON   TMCLK: 27MHz
                                           MEM: DEBUG  Dowser: ON
(c) 1999-2000 Copyright National Semiconductor - Not for distribution.
For distribution rights, contact Insyde Software at NSCPM@insydesw.com

```

Figure 16-1. Sample Summary Screen

Configuring/Customizing (Continued)**16.9 CMOS SETUP****16.9.1 SETUP MENUS**

To enter Setup: Press the F1 key while booting the system.

16.9.1.1 General Menu Navigation

The following table describes the keys used to navigate within menus.

Key	Function
Up Arrow	Highlight the item above the currently highlighted item. If on the top item will highlight the last item.
Down Arrow or Space	Highlight the item below the currently highlighted item. If on the bottom item will highlight the top item.
Left arrow	Highlight the item to the left of the currently highlighted item. If there is no item directly to the left, the rightmost item on the previous line is highlighted.
Right Arrow	Highlight the item to the right of the currently highlighted item. If there is no item directly to the right, the leftmost item on the next line will be highlighted.
Use Enter or a Press the chrachter that matches the highlighted 1 st character of an item	Select the currently highlighted item. This will cycle through the Choices for an item or bring up an entry box for direct entry of a value depending on how the item is configured.
F1	Will display a Long Help for the currently highlighted item. It there is no Long Help for the item the Short Help will be displayed.
Escape	Will close the current menu and return to the previous menu. If there is no previous menu, this key will do nothing

Figure 16-2. Setup Navigation Keys

Configuring/Customizing (Continued)**16.9.1.2 Setup Main Menu**

When entering Setup this is the first menu displayed.

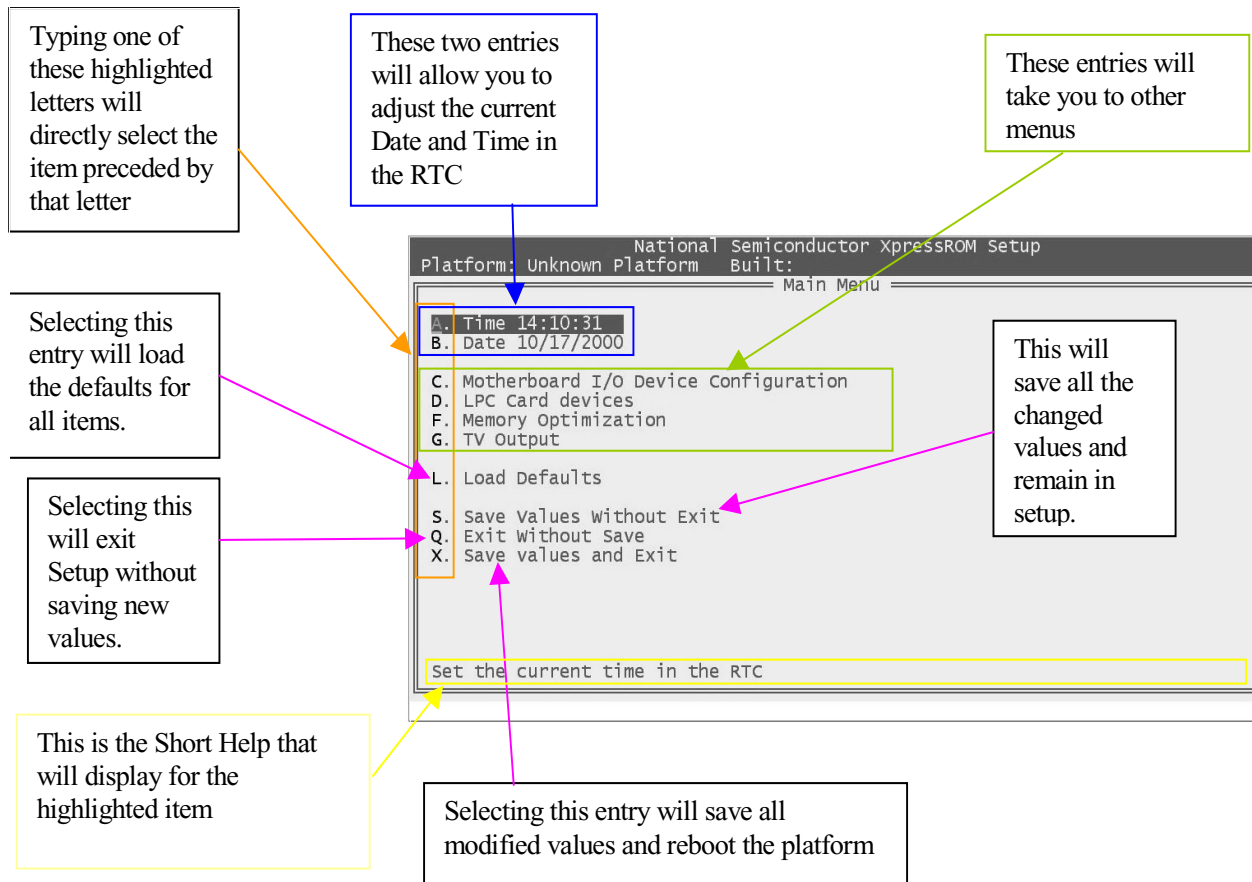


Figure 16-3. Setup Main Menu Display

Configuring/Customizing (Continued)

16.9.1.3 Motherboard I/O Device Configuration Menu

This is the menu for motherboard I/O devices. Note that some entries may not display depending on the platform and Processor chip configured. For instance, Serial Port C will be displayed only for National® Geode™ SCxxx chips.

The values to be used may be changed by highlighting the entry and pressing the enter key to cycle through the choices.

When done press the escape key to exit this menu.

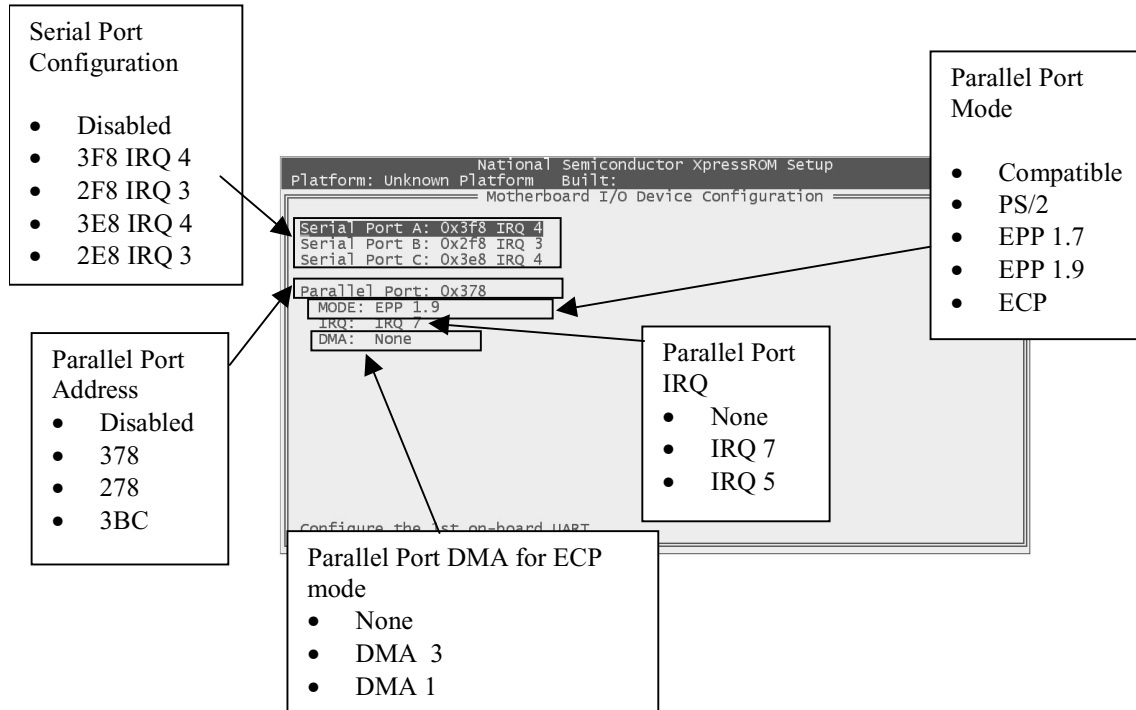


Figure 16-4. I/O Device Configuration Menu

Configuring/Customizing (Continued)

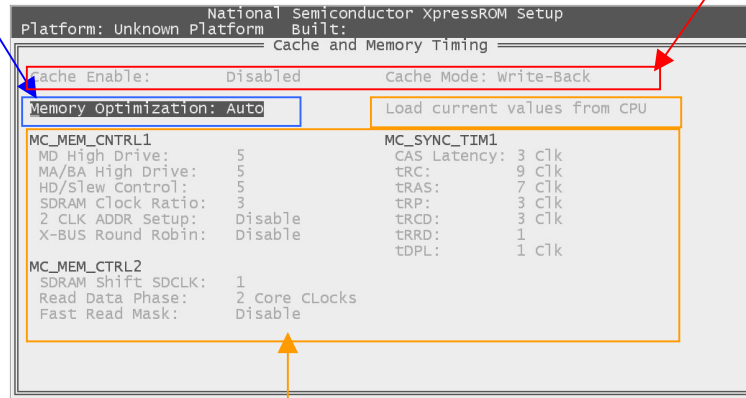
16.9.1.4 Memory Optimization Menu

Standard Menu

This is the standard Memory Optimization menu.

Memory Optimization Type

- Auto
- Aggressive
- Conservative
- Manual



Unimplemented entries are grayed out indicating they are not active at this time.

Only Active in Manual mode.
See next section for Manual Mode settings.

Figure 16-5. Standard Memory Optimization Menu

Manual Settings Menu

When Manual is selected for Memory Optimization the rest of the entries on the menu become active. The values in these entries will (if saved) will become active when Setup is exited and the system re-boots.

It is highly recommended that the user select **Load Current Values From CPU** when first switching to manual optimization in order to obtain the correct base values that the system is using.

WARNING: Changing Memory Optimization values can result in the system becoming non-operational. Change these values only upon understanding the proper values and use of each field from the appropriate CPU's manual.

If the system becomes non-operational, resetting the platform 3 times (when it fails to boot) will result in all default values being restored.

Configuring/Customizing (Continued)

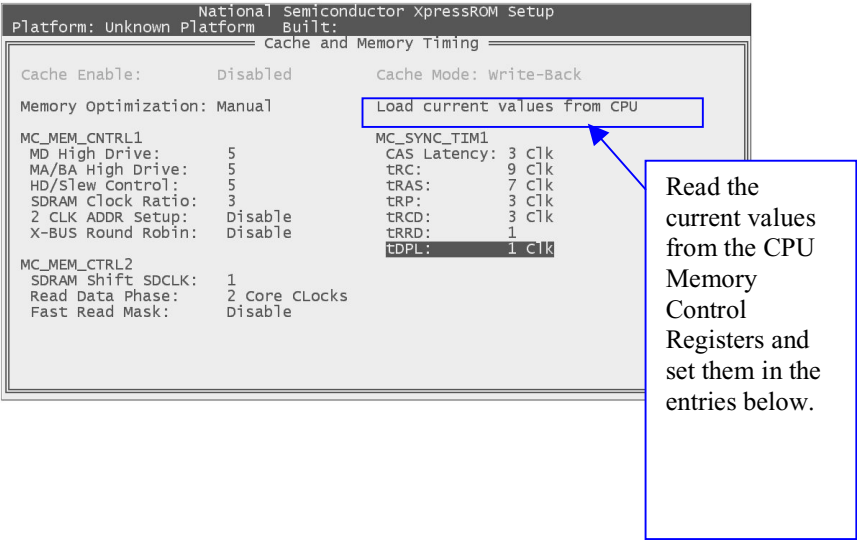


Figure 16-6. Manual Memory Configuration Menu Display

Configuring/Customizing (Continued)**16.9.1.5 TV Output Menu**

This menu is present only with Setup that was compiled for Geode™ SC1200 Chips.

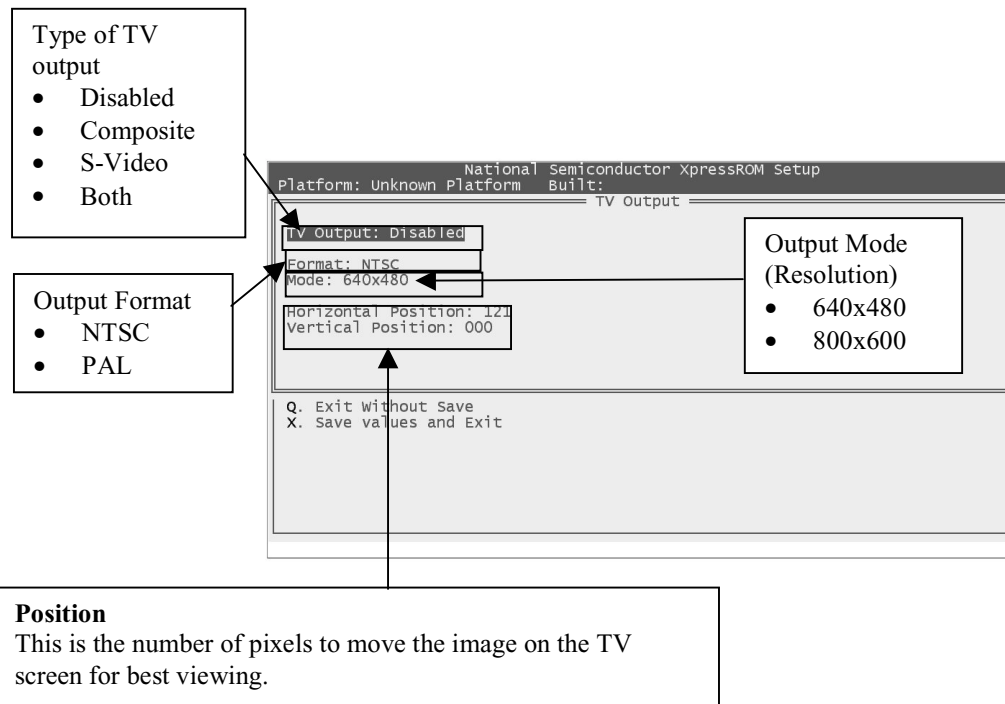


Figure 16-7. TV Output and Configuration Menu

Configuring/Customizing (Continued)

16.9.1.6 LPC Card Devices Menu

This menu is present only with Setup that was compiled for Geode™ SCx200 Chips. Designs using the SCx200 may not include a SIO. In certain designs the LPC bus may be present and available through a connector where a small card that includes an SIO may be inserted. This menu is used to configure such a card for operation in concert with other SC devices, such as serial ports etc.



Figure 16-8. Low Pin Count (LPC) Bus Setup Menu

Configuring/Customizing (Continued)**16.9.2 Programming Reference****16.9.2.1 Sample SMENU.ASM file:**

INIT_MENU

MENUm_menu,"Main Menu",3,1,24,80,TOP_MENU

ITEM "A. Time "

SPECIAL_RTC_TIME "Time"

ITEM_HELP "Set the current time in the RTC"

ITEM "B. Date "

SPECIAL_RTC_DATE "Date"

ITEM_HELP "Set the current date in the RTC"

SKIP_LINE

ITEM "C. Motherboard I/O Device Configuration"

GOTO_MENU io_menu

ITEM_HELP "Set addresses for motherboard devices"

SKIP_LINE

ITEM "L. Load Defaults"

DO_FUNCTION load_defaults

ITEM_HELP "Load Default Values"

SKIP_LINE

ITEM "S. Save Values Without Exit"

LABEL_ITEM no_exit_save

DO_FUNCTION save_changed_values

ITEM_HELP "Saved the changed values (if any) to CMOS"

ITEM "Q. Exit Without Save"

DO_FUNCTION exit_no_save

ITEM_HELP "Exit without saving changes."

LONG_HELP "Exit without saving any changes except for Date And Time. "

ITEM "X. Save values and Exit"

DO_FUNCTION exit_with_save

ITEM_HELP "Save all changes and Exit"

LONG_HELP "Save all changed values and then re-boot the system "

LONG_HELP "in order that the changed values will take effect."

MENUio_menu,"Motherboard I/O Device Configuration",3,1,25,80,NO_DIRECT

ITEM "Serial Port A: "

ITEM_HELP "Configure the 1st on-board UART"

LABEL_ITEM uart1_port

Configuring/Customizing (Continued)

```

FIELD_DEF TOKEN,TOKEN_UART1_CONFIG,UART_CONFIG_WIDTH,1 or ZERO_OK
CHOICE"Disabled",TVALUE_UART_CONFIG_DIS
CHOICE"0x3f8 IRQ 4",TVALUE_UART_CONFIG_3F8
CHOICE"0x2f8 IRQ 3",TVALUE_UART_CONFIG_2F8
CHOICE"0x3e8 IRQ 4",TVALUE_UART_CONFIG_3E8
CHOICE"0x2e8 IRQ 3",TVALUE_UART_CONFIG_2E8

```

ITEM "Serial Port B: "

ITEM_HELP "Configure the 2nd on-board UART"

LABEL_ITEMuart2_port

```

FIELD_DEF TOKEN,TOKEN_UART2_CONFIG,UART_CONFIG_WIDTH,1 or ZERO_OK

```

SAME_CHOICES

SKIP_LINE

ITEM "TV Horizontal Position: "

LABEL_ITEM tv_hpos

```

FIELD_DEF TOKEN,TOKEN_TV_HPOS,TV_HPOS_WIDTH

```

RANGE"TV Horizontal Position (0-255) ",DECIMAL,0,255

END_MENU

16.9.2.2 MACRO REFERENCE**INIT_MENU**

This must be the first entry. This will initialize the internal variables needed for the menu system.

MENU name,title,ul,uc,lr,lc,flags

This entry defines a menu screen.

Parameters:

- ² Name – This is the name of the menu.
- ² Title – This is the title of the menu that will be displayed on the top border line of the menu. It is a character string.
- ² UR – This defines the top row on the screen that this menu will occupy.
- ² UC – This defines the leftmost column on the screen that this menu will occupy.
- ² LR – This defines the bottom row on the screen that this menu will occupy.
- ² LC – This defines the rightmost row on the screen that this menu will occupy.
- ² Flags [optional] – These are any special flags required for this menu. Valid Flags are:
 - q NO_DIRECT – This flag, if present, will suppress the direct selection of a menu item by typing the 1st character of the item name.
 - q TOP_MENU – This flag, if present, declares this menu to be the “Top Menu”. A menu so tagged cannot be exited by hitting the ESCAPE key. This is normally used for the main menu to force a user to select an action such as “SAVE” or “QUIT”.

When this entry is processed the menu is assigned an ordinal number. The first menu is ordinal 0 and each subsequent menu is assigned the next number in sequence. This ordinal is made PUBLIC as `ORDINAL_name`. This may be used by user functions to determine which menu is being processed.

Configuring/Customizing (Continued)

The address of the menu definition entry is also made PUBLIC in the form LABEL_ *name*. This label may be used by user functions to alter various parts of the menu definition. **Use with extreme care** as there is no error checking of altered entries!

SKIP_LINE

This will place a blank line on the screen and force the next item to appear one line down from where it would normally be placed. This item will only have an effect if the **ROW** entry in the **ITEM** entry is omitted.

ITEM *name,row,col,flags*

This entry defines an item.

Parameters:

- ² *name* – This is the text that will be placed on the menu window at the given row and column.
- ² *row* – [optional] This is the row in the menu window where the item will be placed. If omitted, the item will be placed on the line after the last defined item, or if this is the 1st item, line 2 of the menu screen will be used. In addition, row can be “LAST_ITEM_ROW” which will place the item on the same row as the previous item.
- ² *col* – [optional] This is the column of the menu screen where the item will be placed. If col is omitted the item will be placed in the starting in the same column as the previous entry. If this is the 1st entry in a menu, the default starting column is column 2. If col is entered, it will become the default starting column for all following entries in the current menu.
- ² *flags* – This parameter contains flags that modify the default behavior of this item.
 - q NO_SELECT – Do not allow this item to be selected.
 - q NO_DISPLAY – Do not display this item. The item will appear as a blank line on the Menu.
 - q GRAY_OUT – Display this item in low intensity. Normally combined with NO_SELECT.
 - q BOLD – Display this item in high intensity.

LABEL_ITEM *user_label*

This entry labels the current item with the user label and makes PUBLIC the following information about the item:

- ² **LABEL_ITEM_ *user_label*** – The label of the item entry definition for the current item.
- ² **ITEM_MENU_ORDINAL_ *user_label*** – The ordinal of the menu containing the item.
- ² **ITEM_ORDINAL_ *user_label*** – The ordinal of the item within the menu.

ITEM_HELP *help_text*

This defines the help text that will appear on the bottom line of the menu screen when this item is highlighted. Help text must consist of only one line and be no wider than the menu screen. The help text will start in column 2 of the menu screen. There may be only one **ITEM_HELP** line for each item.

LONG_HELP *long_help_text*

This defines one line of a long help that will be displayed when the **F1** key is pressed while an item is highlighted. There may be more than one **LONG_HELP** line for the item. Each will display as a separate line in the long help screen. If there are more lines than will fit on the long help screen the user may scroll up and down with the PgUp and PgDown keys.

GOTO_MENU *menu_name*

This entry tells the system, that when this item is selected; that the menu specified by **menu_name** will be displayed. **Menu_name** must be the same as a **name** parameter of a **MENU** entry. In addition

Configuring/Customizing (Continued)

menu_name may be one of the following special entries:

- ² QUIT_TO_TOP – Exit out to the menu that had the **TOP_MENU** flag. If no menu in the current tree had the **TOP_MENU** flag set then exit the menu system.
- ² PREV_MENU – Go to the menu that called this menu.
- ² SAME_MENU – Stay on this menu.

DO_FUNCTION function_name

This entry tells the system, that when this item is selected; that the function specified by **function_name** will be executed. The function is a 'c' function external to the smenu.asm file. The prototype of the function is:

```
int function_name(int menu, int item);
```

Where:

Menu is the ordinal number of the menu that contains the item.

Item is the ordinal number of the item within the menu.

The function must return on of the following:

- ² Zero – The function had an error. An error beep will sound. No **GOTO_MENU** (if any was defined) will be done.
- ² One – The function was successful. Any **GOTO_MENU** define will be done.
- ² Negative number – Exit up n levels of menu. A -1 will go up one menu.

FIELD_DEF field_type,field_name,length,conflict

This defines a field that will be operated upon when the item is selected. There may be only one **FIELD_DEF** for an item.

Parameters:

- ² field_type – Always the literal "TOKEN"
- ² field_name – This is the NVRAM TOKEN that this field will act upon.
- ² length – This is the length in **BITS** of this field.
- ² conflict – This entry is used for conflict checking. All fields with the same conflict number will be compared for identical values. If identical values are found " **Conflict nn**" will be displayed next to all items with the same conflict number. (nn is the conflict number.) In addition the flag "**ZERO_OK**" may be or'ed in with the conflict number to indicate that multiple items with the value zero are NOT a conflict.

NOTE: The following entries CHOICE, SAME_CHOICE and RANGE are mutually exclusive!

CHOICE prompt,value

This entry defines one of several choices for this field. Each time the item is selected the system will cycle to the next choice. There may be multiple **CHOICE** entries for a **FIELD_DEFINITION**.

Parameters:

- ² prompt – This is what will be displayed next to the item when the value matches the current selected value.
- ² Value – This is the value for this choice.

SAME_CHOICES

This entry indicates that this item has the same choices as the previous item.

Configuring/Customizing (Continued)**RANGE prompt,type,lowvalue,highvalue**

This entry indicates that, when this item is selected, a data entry box will be displayed asking for the value to assign to this field. The value entered will be checked to see if it is greater than or equal to the **lowvalue** and less than or equal to the **highvalue**. If the value is valid it will be stored in the field. If the value is outside the range, it will not be stored and an error beep will sound.

Parameters:

- ² Prompt – This is the prompt that will be displayed for this item in the entry box.
- ² Type – This is the entry/display format desired. Values are:
 - q **DECIMAL** – Entry is an unsigned decimal number
 - q **HEX** – Entry is an unsigned number in Hexadecimal format
- ² Lowvalue – This is the lowest value allowed for this entry.
- ² Highvalue – This is the highest value allowed for this entry.

SPECIAL_RTC_TIME prompt

This is a special entry indicating that the current item is the current time from the RTC. This item will be continuously updated when it is on the current menu and that menu is the current displayed window. When this item is selected an entry box will open requesting the new time to be entered (24-hour format).

This entry may be only used once in the smenu.asm file.

Parameters:

- ² Prompt – This is the prompt that will be displayed for this item in the entry box.

SPECIAL_RTC_DATE prompt

This is a special entry indicating that the current item is the current date from the RTC. This item will be continuously updated when it is on the current menu and that menu is the current displayed window. When this item is selected an entry box will open requesting the new date to be entered (mm/dd/yyyy format).

This entry may be only used once in the smenu.asm file.

Parameters:

- ² Prompt – This is the prompt that will be displayed for this item in the entry box.

COMMENT_ITEM iname,row,col

This defines a non-selectable item on the menu. This may be used for headings within a menu. It is the same as defining an item with the “**NO_SELECT**” flag.

Parameters:

- ² name – This is the text that will be placed on the menu window at the given row and column.
- ² row – [optional] This is the row in the menu window where the item will be placed. If omitted, the item will be placed on the line after the last defined item, or if this is the 1st item, line 2 of the menu screen will be used. In addition, row can be “**LAST_ITEM_ROW**” which will place the item on the same row as the previous item.
- ² col – [optional] This is the column of the menu screen where the item will be placed. If col is omitted the item will be placed in the starting in the same column as the previous entry. If this is the 1st entry in a menu, the default starting column is column 2. If col is entered, it will become the default starting column for all following entries in the current menu.

Configuring/Customizing (Continued)**SKIP_IF_NOT xx and END_SKIP xx**

These are special macros used to remove code from the menu system for features that are not needed for a particular platform. They are used due to the fact that predefined code in SETMAIN.C may use certain external made public by SMENU.ASM. These functions will set a flag that will preserve the PUBLICs in SMENU.ASM without generating any code. Any LABEL_ will be made public as a NULL. Any ORDINAL_ will be made public as -1. The code in SETMAIN.C will recognize these and not perform any operations on these entries.

xx is compared to 0 and, if so, turns on the skip flag. If xx is not zero the skip flag is unchanged.

SKIP_IF_NOT and END_SKIP may be nested, but must not overlap. Example:

GOOD:

```
SKIP_IF_NOT A
:
:
SKIP_IF NOT B
:
:
END_SKIP B
:
END_SKIP A
```

WRONG!:

```
SKIP_IF_NOT A
:
:
SKIP_IF NOT B
:
END_SKIP A
:
:
END_SKIP B
```

END_MENU

This item closes out the menu definitions and performs any final cleanup as needed. This must be the last item after all menu definitions and must be present..

User Functions

The functions called by **DO_FUNCTION** are normally contained in the SETMAIN.C file which may be modified by the user. The function is a 'c' function with a 'c' calling sequence. The prototype of the function is:

```
int function_name(int menu, int item);
```

Where:

Menu is the ordinal number of the menu that contains the item.

Item is the ordinal number of the item within the menu.

The function must return on of the following:

- ² Zero – The function had an error. An error beep will sound. No **GOTO_MENU** (if any was defined) will be done.
- ² One – The function was successful. Any **GOTO_MENU** define will be done.
- ² Negative number – Exit up n levels of menu. A -1 will go up one menu.

Configuring/Customizing (Continued)**Predefined Special User Functions**

The following functions are pre-written and contained in the SETMAIN.C file and may be called with the **DO_FUNCTION** macro.

load_current

This function will load all fields with the current values from NVRAM for each token defined in a **FIELD_DEF** macro. Any value stored in a field will be replaced. Unsaved values will be lost.
This function always returns a 1.

Function prototype: `int load_current(int menu,int item);`

load_defaults

This function will load all fields with the default values for NVRAM for each token defined in a **FIELD_DEF** macro. Any value stored in a field will be replaced.
This function always returns a 1.

Function prototype: `int load_defaults(int menu,int item);`

save_changed_values

This function will load all fields with the default values for NVRAM for each token. In addition NVRAM is check-summed and the Failed boot flag in NVRAM is cleared.
This function always returns a 1.

Function prototype: `int save_changed_values(int menu,int item);`

exit_no_save

This function will bring up a question box asking if the user really wants to exit without save. If the user answers no, the function will exit with a return code 1 keeping the same menu and item. If the user responds with a yes, the function will exit with a -99, forcing setup to terminate. The system will then re-boot.

Function prototype: `int exit_no_save(int menu,int item);`

exit_with_save

This function will save all changed fields to NVRAM. If a save to NVRAM fails, the function will exit with returning a zero, keeping the user in setup. If all saves are good, the function will return with a QUIT_TO_TOP, which will exit setup and re-boot the system.

Function prototype: `int exit_with_save(int menu,int item);`

get_mem_stuff

This function will read the MC_MEM_CTRL1, MC_MEM_CTRL2, and the MC_SYNC_TIM1 registers, split out the sub-fields, and set the appropriate fields for MANUAL memory optimization.
This function always returns a 1.

Function prototype: `int get_mem_stuff(void);`

Configuring/Customizing (Continued)

17.0 Debugging

17.1 MTEST PINS:

MTEST is a feature of the Geode processor that generates special debug information used by hardware engineers to trace program execution. Cache is DISABLED when MTEST is used, and this feature will DISABLE DIMM1.

DO NOT enable this feature if you do not plan on using it.

17.2 GPCS#

The General Purpose Chip Select (GPCS) may be set to activate on I/O reads and writes to a 16-bit I/O location with a width of 1 to 32 bytes. This may be used for example to monitor Port 80 writes on a system which has no other way to see these. Full documentation can be found in Section 4.3.1 in the CS5530 I/O companion data book.

18.0 Utilities

This chapter describes the usage of custom and off-the-shelf build utilities, each of which is required for building and debugging the XpressROM firmware:

18.1 CUSTOM BUILD UTILITIES BY NATIONAL SEMICONDUCTOR

- X2ROM.EXE – Convert a .EXE file to a zero-based binary image.
- FLASHROM.COM – Update the contents of a memory-mapped flash ROM device.
- RLE.EXE – Compress a file using a Run Length Encoding algorithm.
- BMPFLIP.EXE – Flips the order of the bytes that appear in the OEM “splash” screen bit-map file.
- CPRPAD.EXE – Pads out the concatenated rle files to 192 KB.
 - which.pl - a perl implementation of which. Used to locate where files are being executed from. Downloaded from: <http://language.perl.com/ppt/what.html>.
 - buildtool.pl - a perl script used by the build. It generates the date & time of the build to be used in the display screen, and maintains the file userdir.txt which makes sure all files are re-built correctly when building from different user directories.
- ROMPAD.EXE – Adjusts a ROM image to be 64 KB in size, padding the file with 0xFF's at the beginning.
- BINCOPY.EXE – Concatenates files together from a list of files.

18.1.1 X2ROM.EXE – .EXE to .ROM Conversion Utility Version 1.0

X2ROM.EXE, converts a .EXE file to its zero-based, .ROM equivalent. As part of this conversion process, X2ROM performs the following sequence of operations on the .EXE file:

- 1) Calculates the binary image size based on information within the .EXE header.
- 2) Strips off .EXE relocation header.
- 3) Pads the file with trailing FFh bytes according to the user-specified size argument.
- 4) Prepends the file with leading FFh bytes if the user has so specified.
- 5) Checksums the file image and place the checksum in the last location.
- 6) Writes the resulting binary file image to the user-specified filename.

Additionally, X2ROM.EXE provides a mode that is compatible with DOS.

Example of usage:

X2ROM filename /F:xxxx /O:xxxx /S:xxxx /V /X /Y

Where: filename = .EXE file to convert (No Extension)

/F:xxxx	= Hex fixup segment	– Default F000h
/O:xxxx	= Hex code origin in .ROM	– Default 0
/S:xxxx	= Hex size .ROM to make	– Hex size .ROM to make
/V	= Verbose	– Default OFF
/X	= EXE2BIN mode	– Default OFF
/Y	= EXE2BIN mode w/padding	– Default OFF

Utilities (Continued)

The usage for each command line parameter is as follows:

X2ROM.EXE – .EXE to .ROM file conversion utility version 1.0

Magic number:	5a4d
Bytes on last page:	01e4
Pages in file:	0008
Relocations:	0000
Paragraphs in header:	0020
Extra paragraphs needed:	0000
Extra paragraphs wanted:	ffff
Initial stack location:	0000:0000
Word checksum:	0000
Entry point:	0000:0000
Relocation table address:	001e
Memory Needed:	4k

X2ROM.EXE – The “/F:xxxx” (Fixup Segment) Command Line Argument

The “/F” argument is used to “fix” far (segment:offset) links within the .EXE file with the hexadecimal value xxxxh supplied by the developer. Normally, segment fixups are supplied by DOS’ .EXE loader, however, this loader is unavailable to ROM modules. Therefore, it must be supplied during the build process by the developer. Also, the fact that a ROM module requires a fixup is normally (but not always) indicative of a problem with the code’s structure.

X2ROM.EXE – The “/O:xxxx” (Code Origin) Command Line Argument

The “/O” argument is used to move the code origin in the resulting .ROM output file from offset 0 to the caller-specified offset xxxxh. Normally, this option is used to “pad” the beginning of the .ROM output file with FFh bytes if so desired. It saves the developer from having to create an additional utility to prepend “empty” space to the end of the .ROM output file.

X2ROM.EXE – The “/S:xxxx” (Code Size) Command Line Argument

The “/S” argument is used to specify the size, in bytes, of the resulting .ROM output file. Unless executing in EXE2BIN mode (described below) X2ROM.EXE will calculate the 2’s complement of the binary ROM image and place the checksum value at offset xxxxh – 1 within the resulting .ROM output file. It is the responsibility of the developer to ensure that the placement of this checksum byte does not overwrite, or in any other way affect the operating of the code within the ROM image.

X2ROM.EXE – The “/V” (Verbose Output) Command Line Argument

The “/V” argument is used to display information contained in the .EXE header. The format of this information is similar to that output by the Microsoft EXEHDR.EXE utility. The following section displays the verbose output information for a ROM-based keyboard handler. For an explanation of each field, consult a PC technical reference book such as Raymond Duncan’s, “Advanced MS-DOS Programming”, available from Microsoft Press.

X2ROM.EXE – The “/X” (EXE2BIN Compatible) Command Line Argument

The “/X” option is used primarily to validate the X2ROM.EXE utility. When this argument is specified, X2ROM outputs a file with no checksum, nor any leading or trailing “pad” bytes. This is the same format as that output by the MS-DOS EXE2BIN utility. Therefore, X2ROM.EXE functionality can be checked by running X2ROM.EXE (with the “/X” option specified) and EXE2BIN.EXE on the same .EXE file and comparing the resulting output files. Note that the “/X” option cannot be used in conjunction with the “/O” or “/S” command line switches, nor can it be used in conjunction with the “/Y” option, which is described in the next section.

X2ROM.EXE – The “/Y” (EXE2BIN Mode w/Padding) Command Line Argument

The “/Y” option is identical in nature to the “/X” option described in the previous section, except it causes X2ROM to “round up” the file size to the next paragraph boundary and pads the trailing paragraph of the output file with FFh bytes.

Utilities (Continued)**18.1.2 FLASHROM.COM – Flash ROM Device Update Utility**

FLASHROM.COM is used to update the contents of a flash ROM device with a new version of the XpressROM firmware or equivalent boot loader.

Example of usage:

"FlashROM file_name" [Flash & clear CMOS]

The following displays the command line options supported by versions 8.00 and newer of FLASHROM.COM:

FlashROM: Version 8.11

Platform = unknown/unknown

FlashROM: Version 8.11 by National Semiconductor® Corporation

FlashROM flash updates Geode processor/I/O Companion based systems that have one of the following ROMs installed:

AMD	28F020	29F200	29F020N	29F002(N)(T)
Atmel	29C020	29C040A	49F020	29LV020
MXIC	28F2000P	28F2000TP		
SST	29F002T	29F002B		
SST	29EE020	29LE020		
Intel	28F020			
Fujitsu	29F200T			
Winbond	29C020			

Options:

FlashROM	file_name	/D"	Flash & don't clear CMOS
FlashROM	file_name	/N"	No messages
FlashROM	file_name	/R"	Read ROM and write to file
FlashROM	/C"		Clear CMOS only]
FlashROM	/I"		ID ROM]
FlashROM	/?"		[This message

FLASHROM.COM – The “/D” Command Line Argument

The “/D” command line argument prevents FLASHROM.COM from clearing the contents of the RTC CMOS RAM following the completion of the flash ROM update. This argument is required if the caller wants to preserve the current CMOS settings when updating the system flash ROM device.

FLASHROM.COM – The “/N” Command Line Argument

The “/N” option prevents FLASHROM.COM from displaying sign-on/status messages during the flash update procedure.

FLASHROM.COM – The “/R” Command Line Argument

The “/R” option instructs FLASHROM.COM to read the contents of the flash device into the file “file_name” as specified by the user. FLASHROM determines the size of the output file by first “ID”-ing the installed flash part and retrieving its size from an internal table.

FLASHROM.COM – The “/C” Command Line Argument

The “/C” option instructs FLASHROM to clear the contents of the RTC CMOS RAM. This feature is useful for enabling options that have been newly introduced to the system firmware and would otherwise not get enabled automatically. The “/C” option can be used in conjunction with any command line option other than “/D” (Don’t Clear CMOS RAM).

FLASHROM.COM – The “/I” Command Line Argument

Utilities (Continued)

The “/I” option instructs FLASHROM.COM to display the manufacturer and type of the installed flash ROM device.

18.1.3 RLE.EXE – Run Length Encoded compression Utility

RLE.EXE is used to compress code/data modules intended for inclusion in an XpressROM firmware component image.

RLE: Version 1.30 National Semiconductor® Corporation

Run Length Encoding utility

Example of usage:

RLE filename		[Run length encodes filename, finds best fit]
RLE filename	/A:xxxx	[xxxx = phys dest address (dword)]
RLE filename	/C:x	[x = forced compression mode (0,1,2)]
RLE filename	/D:x	[x = forced decompression mode(0,1,2)]
RLE filename	/B	[Make header a \$BMP not \$IMG]

The files that RLE generates are:

- RLE.CMP [Compressed version of filename]
- RLE.EXP [Decompressed version of RLE.CMP (ie. same as filename)]
- RLE.HDR [Header information plus the RLE.CMP file]

Normally, the operation of RLE.EXE is embedded in “make” files and is transparent to the developer. In any case, the following section details each command line argument that RLE.EXE supports:

RLE.EXE – The “/A” Command Line Argument

If specified, the “/A:xxxxxxxh” argument instructs RLE.EXE to insert the physical address xxxxxxxxh in the header that it prepends to the compressed image. This address specifies the location at which the companion decompression code within the boot loader will insert the decompressed code or data module.

RLE.EXE – The “/C” Command Line Argument

If specified, the “/C:n” argument instructs RLE.EXE to use the caller-specified compression mode. By default, RLE.EXE tries each compression method and uses the most optimal. The “/C:n” argument allows the caller to override the optimal method and use method n (where n=[0, 1, 2 ...] instead. This command line switch is used primarily to test RLE.EXE.

RLE.EXE – The “/D:n” Command Line Argument

If specified, the “/D:n” argument instructs RLE.EXE to use the caller-specified decompression mode. By default, RLE.EXE decompresses a file based on the compression mode indicator that RLE.EXE placed in the compressed file information header when the file was originally compressed. The “/D:n” argument allows the caller to override the method contained in the compression information header and use method n (where n=[0, 1, 2 ...] instead. This command line switch is used primarily to test RLE.EXE.

Utilities (Continued)

RLE.EXE – The “/B” Command Line Argument

If specified, the “/B” command line argument instructs RLE.EXE to embed the string “\$BMP” (rather than “\$IMG”) in the file’s compression information header. This argument is used when the source file is a bit-map rather than a code or data image. The primary use of this argument is to simplify the XpressROM device’s search for the system’s “splash” screen image during the power on sequence. Table 18-1 details the format of the information header that RLE.EXE prepends to each compressed code or data image:

Table 18-1. Information Header Format

Offset	Description
00h	‘\$IMG’ or ‘\$BMP’ image type identification string
04h	Compression mode (0 = byte count, 1 = word count, 2 = dword count)
05h	Physical address at which to decompress the image
09h	Length of code/data image file prior to compression
0Dh	Length of code/data image file following compression
11h	Checksum of compressed file
15h	Start of compressed data

18.1.4 BMPFLIP.EXE – Windows .BMP File Conditioning Utility

BMPFLIP.EXE is used during the code build process to flip the order of the bytes that appear in the OEM “splash” screen bit-map file. The resulting bit-map file contains the original image in a format that is more easily displayed by the XpressROM firmware component power on code.

Example of usage:

```
 bmpflip [in.bmp] [outfile] [width] [height]
```

Normally, BMPFLIP is executed automatically from the make-file and is never invoked directly by the developer.

18.1.5 CPRPAD.EXE – ComPressed ROM Pad Utility

CPRPAD.EXE is used during the code build process to “pad” the CPR “ComPressed ROM” image to 196 KB by adding FFh bytes to the beginning of this image. The XpressROM firmware component is structured in such a fashion that any components other than the boot loader reside in compressed form in the 196 KB region between FFFC0000h and FFFEFFFFh. The boot loader occupies the remaining 64 KB of flash ROM address space starting at FFFF0000h.

Example of usage:

```
 CPRPAD [filename.cpr]
```

Where: Filename.cpr is the name of both the input and output files. Normally, CPRPAD is executed automatically by the “make”-file and requires no intervention by the developer.

18.1.6 ROMPAD.EXE – Boot Loader Image Pad Utility

ROMPAD.EXE is used during the XpressROM firmware component build process to condition the 64 KB boot loader image that resides in the traditional system BIOS space at F000:0h. ROMPAD prepends FFh bytes to the XpressROM firmware component boot loader to ensure that the file is exactly 64 KB in length.

Example of usage:

```
 ROMPAD [filename.rom]
```

Where: Filename.rom is the name of both the input and output files.

Normally, ROMPAD is executed automatically by the “make”-file and requires no intervention by the developer.

Utilities (Continued)**18.1.7 BINCOPY.EXE**

Concatenates a group of files into one.

Example of usage:

bincopy [file.lst]

Where: File.lst contains the destination file on a single line followed by full/relative files to add to destination file, each on a separate line.

Example file.lst

- OUTPUT.BIN
- PART1.ROM
- C:\WINDOWS\PART2.ROM
- D:\TEMP\PART3.DOC

Normally, BINCOPY is executed automatically by the “make”-file and requires no intervention by the developer.

18.2 OFF-THE-SHELF BUILD UTILITIES

This section lists those widely available build utilities required for building the XpressROM firmware component. A brief description accompanies each utility program's name.

- PERL.EXE – “PERL” language interpreter for creating the platform-specific RULES.MAK script. This is distributed under the Gnu Public License, and is available from <ftp://www.perl.com/pub/perl/CPAN/ports/msdos/LMOLNAR/perl542b.zip>
- MASM 6.11c or greater. This is a commercial product which must be purchased separately.
 - Upgrade to 6.14: <ftp://ftp.microsoft.com/softlib/mslfiles/ML614.EXE>
 - ML.EXE 6.11c or greater.
 - ML.EXE is the assembler
 - H2INC.EXE – version 6.12a comes with MASM 6.11
 - H2INC translates the msvc style .h files to the masm style .inc files
- MSVC 1.52. This is a commercial product which must be purchased separately. Newer versions will not work correctly as they are 32-bit not 16-bit compilers.
 - CL.EXE – version 8.00c
 - CL is the C compiler
 - LINK.EXE - Version 5.60.339 comes with MSVC152(link 5.31.009 comes with MASM6.11). Link combines all of the object code from libraries and object files, resolves the named resources, and builds an executable file. Update to 5.60.339: <ftp://ftp.microsoft.com/softlib/MSLFILES/LNK563.EXE>
- NMAKE.EXE - 1.40 or greater (1.20 is included with MASM 6.11 – the upgrade is needed to build)
 - Nmake is used to build & rebuild the project
 - Update to 1.50: <ftp://ftp.microsoft.com/softlib/mslfiles/NMAKE15.EXE>
- LIB.EXE – Version 3.20.010 or greater (comes with both msvc 1.52 and masm 6.11)
 - LIB creates standard libraries, import libraries, and export files that you can use with link.exe

Appendix A Support Documentation

A.1 SPECIFICATION REVISION HISTORY

This document is a report of the revision/creation process of the XpressROM Firmware Functional Specification. A summary revision history and details of specific changes to the document (i.e., additions, deletions, parameter corrections, etc.) are recorded in the following table(s).

Table A-1. Revision History

Revision # (PDF Date)	Revisions / Comments
0.1 (7/28/99)	First pass draft, released to training session attendees.
0.2 (7/30/99)	Software group edited rev 0.1 and provided data on configuring/customizing and debugging sections. This version will be distributed at training session.
0.3 ()	See Table A-2 for details.

Table A-2. Edits to Previous Revision

Section	Description
Features	Added new features and modified text of existing features.
1.0 Specification Introduction	Added SoftVGA to acronym table.
3.0 Post	Added descriptions for 05h and 0Ah of checkpoint 69h and changed many of the Function names in Table 3-1.
7.0 Serial Port Support	Added new section for INT 14h, Functions 00h - 03h.
8.0 Parallel Port Support	Added new section for INT 17h, Functions 00h - 02h.
9.0 PS/2 Mouse Support	Added new section for INT 15h, Function C2h, Subfunctions 00h - 07h.
9.0 System Services	Added Function BEh and Functions 00h to 04h became subfunctions of BEh. Added 05h - 09h and 99h as subfunctions of BEh. Also added information to the existing functions. Added E8h Subfunctions 01h and 20h.
11.0 System Management Mode Software	Modified MSG_INITIALIZE message in Section 13.11 "Messages" on page 114. Removed STORE_STRING macro from Section 13.13.3 "Shorthand Macros" on page 125, does not exist.
15.0 Virtual Hardware	Added Section 14.3 "Virtual RTC/CMOS" and Section 14.4 "Virtual UART".

Table A-3. Edits to Current Revision

Section	Description
Features	Added new features and modified text of existing features.
13	Whole System Management Mode section has been updated.
16	Added CMOS Setup documentation.
All Sections	Checked and updated specific processor references to Geode for consistency.

(Continued)

Section	Description
All Sections	Updated revision to 0.4 as well as date.
11	Updated Int15 service table entries.

(Continued)

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.



National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Francais Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507