Application Note 129
Cyrix III CPU SMM Design Guide

Cyrix Processors

REVISION HISTORY

Date	Version	Revision
5/19/99	1.0	Updated for Cyrix III bus protocol
4/9/99	0.21	Typos
4/8/99	0.2	Removed SMAC, MMAC and SMINT
4/6/99	0.1	Initial Version C:\documentation\joshua\appnotes\cIII_smm.fm

Table of Contents

APPLICATION NOTE 129

Cyrix III SMM Design Guide

1 Introduction

1.1 Scope

This Programmer's Guide is provided to assist programmers in the creation of software that uses the Cyrix™ System Management Mode (SMM) for the Cyrix III processor. Unless stated otherwise, all information in this manual pertains to the Cyrix III CPUs.

SMM provides the system designer with another operating mode for the CPU. Within this document, the standard x86 operating modes (real, V86, and protected) are referred to as normal mode. Normal-mode operation can be interrupted by an SMI interrupt that places the processor in System Management Mode (SMM).

SMM can be used to enhance the functionality of the system by providing power management, register shadowing, peripheral emulation and other system-level functions. SMM can be totally transparent to all software, including protected-mode operating systems.

1.2 Cyrix SMM Features

The Cyrix microprocessors provide a register that allows programming of the location and size of the SMM memory region. The CPUs automatically saves minimal register information, reducing the time needed for SMM entry and exit. The SMM implementation by Cyrix provides unique instructions that save additional segment registers. The x86 MOV instruction can be used to save the general purpose registers.

The Cyrix III CPU simplifies I/O trapping by providing I/O type identification and instruction restarting. This CPU also makes available to the SMM routine information that can simplify peripheral register shadowing.

Cyrix provides a method (setting the SMI_LOCK bit) that prevents the SMM configuration registers from being accessed. Locking the SMM configuration registers enhances system security from programming errors and viruses, but at the expense of making debugging more difficult.

1.3 Typical SMM Routines

Upon entry to SMM, the CPU registers that will be used by the SMM routine must be saved. The SMM environment is initialized by setting up an Interrupt Descriptor Table, initializing segment limits, and setting up a stack. If entry to SMM results from an I/O bus cycle, the SMM routine can monitor peripheral activity, shadow read-only ports, and emulate peripherals in software. If a peripheral is powered down, the SMM routine can power it up and reissue the I/O instruction. If the SMM routine is not the result of an I/O bus cycle, non-trap SMI functions can be serviced. If an HLT instruction is interrupted by an SMI then the HLT instruction should be restarted when the SMM routine is completed. Before normal operation is resumed, any CPU registers modified during the SMM routine must be restored to their previous state.

2 SMM Implementation

This chapter describes the Cyrix SMM System interface. SMM operations for Cyrix microprocessors are similar to related operations performed by other x86 microprocessors. The Cyrix III supports two SMM modes—Cyrix SMM mode and SL SMM mode.

The CPU defaults to SL SMM mode. Setting SMM_MODE bit will cause the CPU to operate in Cyrix SMM mode. SMM_MODE is controlled by CCR6 bit 0.

2.1 SMM Pins

In either SMM mode, there are three functions that need to occur:

1. Signaling when an SMI interrupt should occur,
2. Informing the chipset that the CPU is in SMM mode,
3. Informing the chipset whether the bus cycle is intended for SMM memory or system memory.

The SMI# pin and the SMM acknowledge bus cycle are used to implement SMM and cover all of these functions.

2.2 Cyrix SMM Mode and SL SMM Mode.

The CPU defaults to SL SMM mode. The only difference between SL mode and Cyrix mode is that in Cyrix Mode, SMM memory accesses can be cached. This support is not built into SL mode, however Cyrix III can run in SL mode for compatibility.

2.2.1 SMM Enter and Exit

An SMM routine is commenced by asserting the SMI# input pin. When the cpu recognizes the SMI# input, an SMM acknowledge cycle is generated on the bus.

To enter Cyrix SMM mode, the SMI# pin must be asserted during an interruptible point in program execution. Once the CPU recognizes the active SMI# input, an SMI acknowledge transaction is generated on the system bus to inform the chipset that the processor is now in SMM mode.

The SMM routine is terminated with an SMM-specific resume instruction (RSM). When the RSM instruction is executed, an SMM acknowledge cycle is again generated on the system bus, to inform the chipset that the processor is no longer in SMM mode.

SMI# is an edge-triggered input pin sampled by two rising edges of CLK. SMI# must meet certain setup and hold times to be detected on a specific clock edge. To accomplish I/O trapping, the SMI# signal should be asserted two clocks before the RESPONSE phase for that I/O cycle. Once the CPU recognizes the active SMI# input, the CPU drives the SMM_MEM bit (EX4#) active for the duration of the SMM routine. The SMM routine is terminated with an SMM-specific resume instruction (RSM).

When the RSM instruction is executed, the CPU negates the SMM_MEM bit after the last bus cycle to SMM memory.

2.2.2 SMM_MEM - SMM Memory Access Signal

To signal to the chipset that memory accesses are SMM space accesses, the cpu asserts the EX4# bit in request packet B. This bit is called SMM_MEM. This is done in the request phase of the SMI acknowledge special cycle. EX4# is the multiplexed signal on address bit seven in the second packet of the request phase. This bit is the only way the chipset can know if SMM memory is being accessed, and main memory cannot be accessed while SMM_MEM is set.

2.3 Configuration Control Registers and SMM

This section describes fields in the Configuration Registers that configure SMM operations. Fields not related to SMM are not described in this manual and are shown as blank fields in the configuration register tables. For a complete description of the configuration registers, refer to the appropriate data book.

All configuration-register bits related to SMM and power management are cleared to 0 when RESET is asserted. Asserting INIT# does not affect the configuration registers.

These registers are accessed by writing the register index to I/O port 22h. I/O port 23h is used for data transfer. Each data transfer to I/O port 23h must be preceded by an I/O port 22h register-index selection, otherwise the port 23h access will be directed off chip.

Before accessing these registers, all interrupts must be disabled. A problem could occur if an interrupt occurs after writing to port 22h but before accessing port 23h. The interrupt service routine might access port 22h or 23h. After returning from the interrupt, the access to port 23h would be redirected to another index or possibly off chip.

An SMI interrupt cannot interrupt accesses to the configuration registers. After writing an index to port 22h in the CPU configuration space, SMI interrupts are disabled until the corresponding access to port 23h is complete.

The portions of the configuration registers that apply to SMM and power management are described in the following pages.

Undefined bits in the configuration registers are reserved.

2.3.3 I/O Port 22h and 23h Access

Access to internal registers (except PCI) is accomplished via an 8-bit index/data I/O pair. Each data transfer, I/O Port 23h, must be preceded by a valid index write, I/O Port 22h. All reads from I/O Port 22h produce external I/O cycles; therefore, the index cannot be read. Accesses that hit within the on-chip configuration registers do not generate external I/O cycles.

To access the above registers, the programmer uses the Port 22h (index) and Port 23h (data). The access is atomic when an SMI is involved, but is not atomic when any of the following three conditions are presented: 1) INT, 2) NMI 3) INIT#. Proper steps must be taken to inhibit these three conditions if a pure atomic operation is to be achieved. An example of this to prevent an INT sequence would be to use a PUSHF, CLI instruction pair before doing the Port 22h/23h access with a POPF after the access has been done.

After reset, only configuration registers with indices C0-CFh and FC-FFh are accessible. This prevents potential conflicts with other devices that use Ports 22h and 23h to access their registers.

2.3.4 Map Enable (MAPEN)

The purpose of MAPEN is to increase the number of registers that can be accessed via Ports 22h/23h. The MAPEN fields can be thought of as a paging mechanism to the complete register set allowing multiple registers to share the same index value but providing different functionality. MAPEN must be set correctly to gain access to the register that is to be modified. MAPEN is located in CCR3[7:4]. It is strongly recommended that the programmer use the following sequence:

1) Read CCR3.
2) Save CCR3.
3) Modify MAPEN at CCR3[7-4].
4) Access Register via Port 22h/23h access.
5) Restore CCR3 to control the MAPEN field.

There are 256 possible registers available for each MAPEN setting, for a total of 4096.

Note: Registers and MAPEN values not mentioned in this section are either reserved, or not used for SMM. See Cyrix III databook for complete register information.

2.3.5 Cyrix Configuration Control Registers (CCR0-CCR7)

The Configuration Control Registers (CCR0-CCR7) are used to assign non-cached memory areas, set up SMM, provide CPU identification information and control various features such as cache write policy and bus locking control.

CCR1, CCR3, and CCR6 may be written at any time unless the SMI_LOCK (CCR3[0]) is set or an SMI is active.

The following register bits must be set accordingly for SMM operation.

1. Configuration Control Register 1 (CCR1) Bit 3: SM3 = 1
2. Configuration Control Register 3 (CCR3) Bit 0: SMI_LOCK = 0 or 1
3. Configuration Control Register 6 (CCR6) Bit 6: Nested SMI_Enable = 1 or 0
4. Configuration Control Register 6 (CCR6) Bit 0: SMM_MODE = 1 or 0
5. Address Region Register 3 (ARR3) All Bits: SMM Memory Base Address and Size
6. Region Control Register 3 (RCR3) All Bits: SMM Memory Properties

Following are the detailed instructions to program each register bit.

2.3.6 Configuration Control Register 1 (CCR1)

7	6	5	4	3	2	1	0
SM3	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*

Index: C1h
Default Value: 20h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
7	SM3	SMM Address Space Address Region 3: 1 = Address Region 3 is designated as SMM address space. 0 = Address Region 3 is system memory.

2.3.7 Configuration Control Register 3 (CCR3)

7	6	5	4	3	2	1	0
MAPEN[3-0]				*Reserved*	*Reserved*	NMI_EN	SMI_LOCK

Index: C3h
Default Value: 00h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
7:4	MAPEN[3-0]	**Map Enable Bits** **These four bits enable different combinations of configuration registers.** 0001 = All configuration registers are accessible except L2 and BIU. 0000 = Only configuration register with indexes: C0 - CFh, FEh, and FFh are accessible
1	NMI_EN	NMI Enable: 1 = NMI interrupt is recognized while servicing an SMI interrupt. NMI_EN should be set only while in SMM after the appropriate SMI interrupt service routine has been set up. 0 = NMI disabled while servicing SMI.
0	SMI_LOCK	SMI Lock: 1 = The following SMM configuration bits can only be modified while in an SMI service routine: CCR1: SM3 CCR3: NMI_EN CCR6: SMM_MODE ARR3: Starting address and block size. Once set, the features locked by SMI_LOCK cannot be unlocked until the RESET pin is asserted.

2.3.8 Configuration Control Register 5 (CCR5)

7	6	5	4	3	2	1	0
Reserved		ARREN	*Reserved*				

Index: E9h
Default Value: 00h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
5	ARREN	**Address Region Registers Enable:** Enables address decoding of ARR0-ARRD. 1 = Enables all ARR registers. 0 = Disables all ARR registers. If SM3 is set ARR3 is enabled regardless of the setting of ARREN. (SM3 is bit 7 in CCR1.)

2.3.9 Configuration Control Register 6 (CCR6)

7	6	5	4	3	2	1	0
Reserved	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*	SMM_MODE

Index: EAh:
Default Value: 40h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
0	SMM_MODE	**SMM Mode:** 1 = Enable Cyrix-enhanced SMM mode. 0 = Disable Cyrix-enhanced SMM mode.

2.4 Address Regions in Memory

Selected regions of main memory space can be assigned different attributes. These regions are called address regions. Each address region is defined by a pair of registers—an Address Region Register (ARRn) and a Region Control Register (RCRn).

The ARRn registers are used to specify the location and size for these regions.

The RCRn registers are used to specify the attributes for these regions.

The number (n) is a hexadecimal number that designates the region number.

2.4.10 Address Region Register 3 (ARR3)

23 16	15 8	7 4	3 0
MAIN MEMORY BASE ADDRESS			SIZE

Index: C4-C6h
Default Value: 00h
Access: Read/Write
MAPEN: xxxx.

Bit	Name	Description
23-16	MAIN MEMORY BASE ADDRESS	Starting address for the particular address region. Memory address bits A[31-24] defined by ARRn bits 23 - 16 Memory address bits A[23-16] defined by ARRn bits 15 - 8 Memory address bits A[15-12] defined by ARRn bits 7 - 4
3-0	SIZE	Size of the particular address region as defined by Table1, ". ARRn Address Region Size Field Definitions," on page16.

Address region 7 defines total system memory unless superseded by other address region definitions. The SIZE field is defined in two different way as listed in Table 2-1. If the address region size field is zero, the address region is disabled. After a reset, all ARR Registers are initialized to 00h. The base address of the ARR address region, selected by the Base Address field, must be on a block size boundary. For example, a 128KB block is allowed to have a starting address of 0KB, 128KB, 256KB, and so on. A 512KB block is allowed to have a starting address of 0KB, 512KB, 1024KB, and so on. Address region 3 defines SMM space when enabled by CCR1 bit 7.

Table 2-1. ARRn Address Region Size Field Definitions

Size	Block Size		
	ARR0-ARR6	ARR7-ARRB	ARRC-ARRD
00h	Disable	Disable	Disable
01h	4 KB	256 KB	256KB
02h	8 KB	512 KB	Reserved
03h	16 KB	1 MB	
04h	32 KB	2 MB	
05h	64 KB	4 MB	
06h	128 KB	8 MB	
07h	256 KB	16 MB	
08h	512 KB	32 MB	
09h	1 MB	64 MB	
0Ah	2 MB	128 MB	
0Bh	4 MB	256 MB	
0Ch	8 MB	512 MB	
0Dh	16 MB	1 GB	
0Eh	32 MB	2 GB	
0Fh	4 GB	4 GB	

2.4.11 Region Control Registers

Region Control Register 3 defines SMM space.

7	6	5	4	3	2	1	0
Reserved	INV_RGN	WP	WT	WG	*Reserved*		RCD

Index: DFh
Default Value: 00h
Access: Read/Write
MAPEN: x00x

BIT POSITION	NAME	DESCRIPTION
6	INV_RGN	ARR0 Invert Region 1 = applies the controls specified in RCRn to all memory addresses outside the region specified in ARR0.
5	WP	Write-Protect 1 = enables write protect for address region n.
4	WT	Write-Through 1 = defines the address region as write through instead of write-back. Any system ROM that is allowed to be cached by the processor should be defined as write through.
3	WG	Write-Gathering 1 = enables write gathering for address region n. With WG enabled, multiple byte, word, dword, or quad-word writes to sequential addresses that would normally occur as individual cycles on the bus are collapsed, or "gathered" within the processor and then completed as a single write cycle. WG improves bus utilization and should be used on memory regions that are not sensitive to gathering.
0	RCD	Cache Disable (RCR0 - RCR6 only) 1 = defines the address region n as non-cacheable.

Only one bit of WP, WT, WG, or RCD may be set.

3 System Management Mode

System Management Mode (SMM) is a distinct CPU mode that differs from normal CPU x86 operating modes (real mode, V86 mode, and protected mode) and is most often used to perform power management.

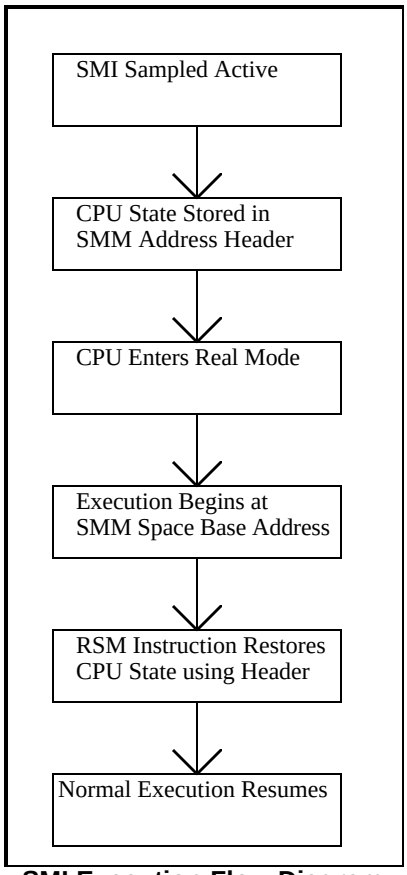
The Cyrix III CPU is backward compatible with the SL-compatible SMM found on previous Cyrix microprocessors. The Cyrix III cpu does not support nesting of SMI's found in previous Cyrix cpus. The Cyrix mode SMM in the Cyrix III cpu does provide for cacheing of SMM memory.

3.1 Overall Operation

The overall operation of a SMM operation is shown on the next page. SMM is entered using the System Management Interrupt (SMI) pin. SMI interrupts have higher priority than any other interrupt, including NMI interrupts.

Upon entering SMM mode, portions of the CPU state are automatically saved in the SMM address memory space header. The CPU enters real mode and begins executing the SMI service routine in SMM address space.

Execution of a SMM routine starts at the base address in SMM memory address space. Since the SMM routines reside in SMM memory space, SMM routines can be made totally transparent to all software, including protected-mode operating systems.



SMI Execution Flow Diagram

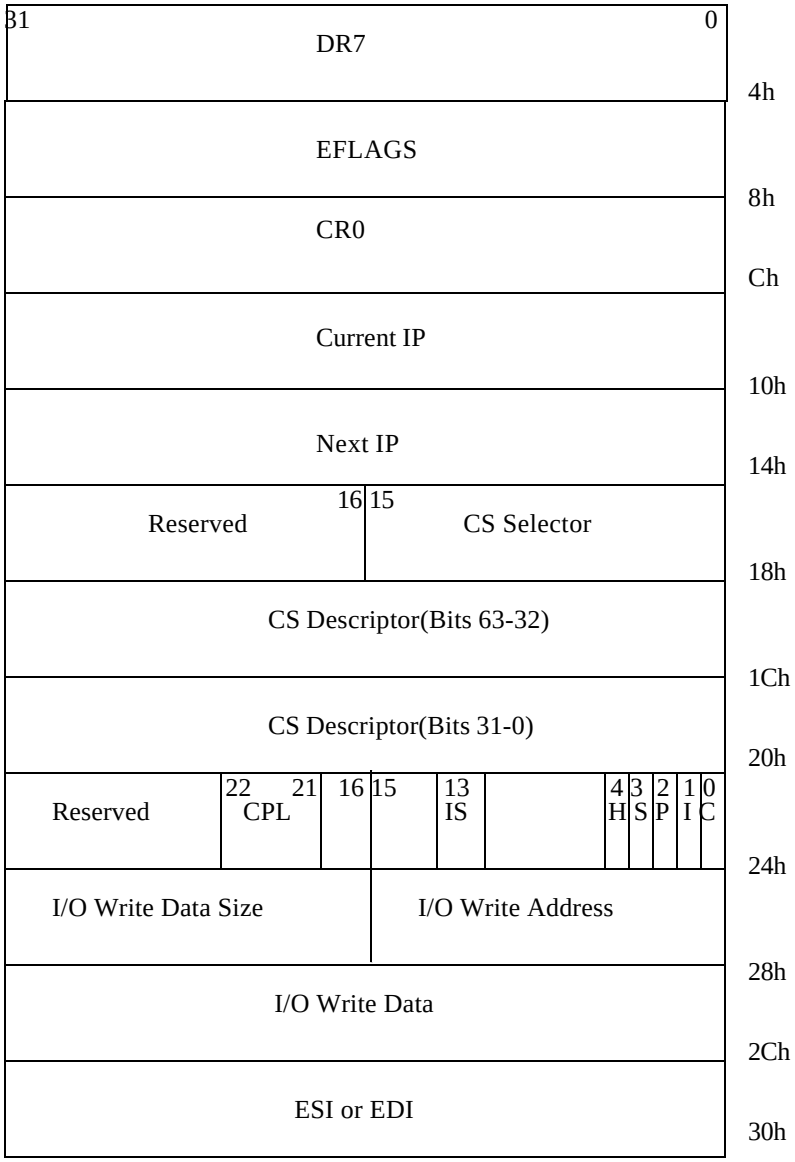
3.2 SMM Memory Space

SMM memory must reside within the bounds of physical memory and not overlap with system memory. SMM memory space, as illustrated on the next page, is defined by setting the SM3 bit in CCR1 and specifying the base address and size of the SMM memory space in the ARR3 register.

The base address must be a multiple of the SMM memory space size. For example, a 32 KByte SMM memory space must be located on a 32KByte address boundary. The memory space size can range from 4 KBytes to 4GBytes. SMM accesses ignore the state of the A20M# input pin and drive the A20 address bit to the unmasked value.

3.3 SMM Memory Space Header

The SMM Memory Space Header (shown in the figure below) is used to store the CPU state prior to starting an SMM routine. The fields in this header are described on the next page. After the SMM routine has completed, the header information is used to restore the original CPU state. The location of the SMM header is determined by the SMM Header Address Register (SMHR)**.**



SMM Memory Space Header

NAME	DESCRIPTION	SIZE
DR7	The contents of Debug Register 7.	4 Bytes
EFLAGS	The contents of Extended Flags Register.	4 Bytes
CR0	The contents of Control Register 0.	4 Bytes
Current IP	The address of the instruction executed prior to servicing SMI interrupt.	4 Bytes
Next IP	The address of the next instruction that will be executed after exiting SMM mode.	4 Bytes
CS Selector	Code segment register selector for the current code segment.	2 Bytes
CS Descriptor	Code segment register descriptor for the current code segment.	8 Bytes
CPL	Current privilege level for current code segment.	2 Bits
IS	Internal SMI Indicator If IS =1: current SMM is the result of an internal SMI event. If IS =0: current SMM is the result of an external SMI event.	1 Bit
H	SMI during CPU HALT state indicator If H = 1: the processor was in a halt or shutdown prior to servicing the SMM interrupt.	1 Bit
P	REP INSx/OUTSx Indicator If P = 1: current instruction has a REP prefix. If P = 0: current instruction does not have a REP prefix.	1 Bit
I	IN, INSx, OUT, or OUTSx Indicator If I = 1: if current instruction performed is an I/O WRITE. If I = 0: if current instruction performed is an I/O READ.	1 Bit
C	Code Segment writable Indicator If C = 1: the current code segment is writable. If C = 0: the current code segment is not writable.	1 Bit
I/O Data Size	Indicates size of data for the trapped I/O write: 01h = byte 03h = word 0Fh = dword	2 Bytes
I/O Write Address	I/O Write Address Processor port used for the trapped I/O write.	2 Bytes
I/O Write Data	I/O Write Data Data associated with the trapped I/O write.	4 Bytes
ESI or EDI	Restored ESI or EDI value. Used when it is necessary to repeat a REP OUTSx or REP INSx instruction when one of the I/O cycles caused an SMI# trap.	4 Bytes

3.3.1 Current and Next IP Pointers

Included in the header information are the Current and Next IP pointers. The Current IP points to the instruction executing when the SMI was detected and the Next IP points to the instruction that will be executed after exiting SMM.

Normally after an SMM routine is completed, the instruction flow begins at the Next IP address. However, if an I/O trap has occurred, instruction flow should return to the Current IP to complete the I/O instruction.

If SMM has been entered due to an I/O trap for a REP INSx or REP OUTSx instruction, the Current IP and Next IP fields contain the same address.

If an entry into SMM mode was caused by an I/O trap, the port address, data size and data value associated with that I/O operation are stored in the SMM header. Note that these values are only valid for I/O operations. The I/O data is not restored within the CPU when executing a RSM instruction.

Under these circumstances the I and P bits, as well as ESI/EDI field, contain valid information.

Also saved are the contents of debug register 7 (DR7), the extended flags register (EFLAGS), and control register 0 (CR0).

3.3.2 SMM Header Address Pointer

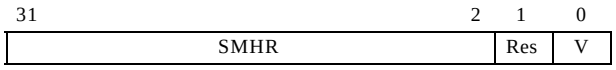
The SMM Header Address Pointer Register (SMHR) (shown on the next page) contains the 32-bit SMM Header pointer. The SMHR address is dword aligned, so the two least significant bits are ignored.

The SMHR valid bit (bit 0) is cleared with every write to ARR3 and during a hardware RESET. Upon entry to SMM, the SMHR valid bit is examined before the CPU state is saved into the SMM memory space header. When the valid bit is reset, the SMM header pointer will be calculated (ARR3 base field + ARR3 size field) and loaded into the SMHR and the valid bit will be set.

If the desired SMM header location is different than the top of SMM memory space then the SMHR register must be loaded with a new value and valid bit from within the SMI routine before nesting is enabled.

The SMM memory space header can be relocated using the new RDSHR and WRSHR instructions.

SMHR Register



SMHR Register Bits

BIT POSITION	DESCRPTION
31 - 2	SMHR header pointer address.
1	Reserved
0	Valid Bit

3.4 SMM Instructions

After entering the SMI service routine, the MOV, SVDC, SVLDT and SVTS instructions, shown in the table below, can be used to save the complete CPU state information. If the SMI service routine modifies more than what is automatically saved or forces the CPU to power down, the complete CPU state information must be saved. Since the CPU is a static device, its internal state is retained when the input clock is stopped. Therefore, an entire CPU state save is not necessary prior to stopping the input clock.

SMM Instruction Set

INSTRUCTION	OPCODE	FORMAT	DESCRIPTION
SVDC	0F 78 [mod sreg3 r/m]	SVDC mem80, sreg3	*Save Segment Register and Descriptor* Saves reg (DS, ES, FS, GS, or SS) to mem80.
RSDC	0F 79 [mod sreg3 r/m]	RSDC sreg3, mem80	*Restore Segment Register and Descriptor* Restores reg (DS, ES, FS, GS, or SS) from mem80. Use RSM to restore CS. Note: Processing "RSDC CS, Mem80" will produce an exception.
SVLDT	0F 7A [mod 000 r/m]	SVLDT mem80	*Save LDTR and Descriptor* Saves Local Descriptor Table (LDTR) to mem80.
RSLDT	0F 7B [mod 000 r/m]	RSLDT mem80	*Restore LDTR and Descriptor* Restores Local Descriptor Table (LDTR) from mem80.
SVTS	0F 7C [mod 000 r/m]	SVTS mem80	*Save TSR and Descriptor* Saves Task State Register (TSR) to mem80.
RSTS	0F 7D [mod 000 r/m]	RSTS mem80	*Restore TSR and Descriptor* Restores Task State Register (TSR) from mem80.
RSM	0F AA	RSM	*Resume Normal Mode* Exits SMM mode. The CPU state is restored using the SMM memory space header and execution resumes at interrupted point.
RDSHR	0F 36	RDSHR ereg/mem32	*Read SMM Header Pointer Register* Saves SMM header pointer to extended register or memory.
WRSHR	0F 37	WRSHR ereg/mem32	*Write SMM Header Pointer Register* Load SMM header pointer register from extended register or memory.

Note: mem32 = 32-bit memory location
mem80 = 80-bit memory location

The SMM instructions can be executed only if:

1. ARR3 Size > 0
2. Current Privilege Level =0
3. SM3 (CCR1-bit 7) = 1

If the above conditions are not met and an attempt is made to execute an SVDC, RSDC, SVLDT, RSLDT, SVTS, RSTS, RSM, RDSHR, or WDSHR instruction, an invalid opcode exception is generated. These instructions can be executed outside of defined SMM space provided the above conditions are met.

The SVDC, RSDC, SVLDT, RSLDT, SVTS and RSTS instructions save or restore 80 bits of data, allowing the saved values to include the hidden portion of the register contents.

The WRSHR instruction loads the contents of either a 32-bit memory operand or a 32-bit register operand into the SMHR pointer register based on the value of the mod r/m instruction byte. Likewise the RDSHR instruction stores the contents of the SMHR pointer register to either a 32 bit memory operand or a 32 bit register operand based on the value of the mod r/m instruction byte.

3.5 SMM Operation

This section describes SMM operations. Detailed information and programming follow in later sections.

3.5.1 Entering SMM

Entering SMM requires the assertion of the SMI# pin. SMI interrupts have higher priority than any interrupt including NMI interrupts.

For the SMI# instruction to be recognized, the configuration register bits must be set as shown in the table below.

Requirements for Recognizing SMI#

REGISTER (Bit)		SMI#
ARR3	SIZE (3-0)	> 0
SM3	CCR1 (7)	1

Upon entry into SMM, after the SMM header has been saved, the CR0, EFLAGS, and DR7 registers are set to their reset values. The Code Segment (CS) register is loaded with the base, as defined by the ARR3 register, and a limit of 4 GBytes. The SMI service routine then begins execution at the SMM base address in real mode.

3.5.2 Saving the CPU State

The programmer must save the value of any registers that may be changed by the SMI service routine. For data accesses immediately after entering the SMI service routine, the programmer must use CS as a segment override. I/O port access is possible during the routine but care must be taken to save registers modified by the I/O instructions. Before using a segment register, the register and the register's descriptor cache contents should be saved using the SVDC instruction. While executing in the SMM space, execution flow can transfer to normal memory locations.

3.5.2.1 Program Execution

Hardware interrupts, (INTRs and NMIs), may be serviced during a SMI service routine. If interrupts are to be serviced while executing in the SMM memory space, the SMM memory space must be within the 0 to 1 MByte address range to guarantee proper return to the SMI service routine after handling the interrupt.

INTRs are automatically disabled when entering SMM since the IF flag is set to its reset value. Once in SMM, the INTR can be enabled by setting the IF flag. NMI is also automatically disabled when entering SMM. Once in SMM, NMI can be enabled by setting NMI_EN in CCR3. If NMI is not enabled, the CPU latches one NMI event and services the interrupt after NMI has been enabled or after exiting SMM through the RSM instruction.

Within the SMI service routine, protected mode may be entered and exited as required, and real or protected mode device drivers may be called.

3.5.2.2 Exiting SMM

To exit the SMI service routine, a Resume (RSM) instruction, rather than an IRET, is executed. The RSM instruction causes the Cyrix III processor to restore the CPU state using the SMM header information and resume execution at the interrupted point. If the full CPU state was saved by the programmer, the stored values should be reloaded prior to executing the RSM instruction using the MOV, RSDC, RSLDT and RSTS instructions.

When the RSM instruction is executed at the end of the SMI handler, the EIP instruction pointer is automatically read from the NEXT IP field in the SMM header.

When restarting I/O instructions, the value of NEXTIP may need modification. Before executing the RSM instruction, use a MOV instruction to move the CURRENTIP value to the NEXTIP location as the CURRENT IP value is valid if an I/O instruction was executing when the SMI interrupt occurred. Execution is then returned to the I/O instruction rather than to the instruction after the I/O instruction.

A set H bit in the SMM header indicates that a HLT instruction was being executed when the SMI occurred. To resume execution of the HLT instruction, the NEXTIP field in the SMM header should be decremented by one before executing RSM instruction.

3.6 SL and Cyrix SMM Operating Modes

There are two SMM modes, SL-compatible mode (default) and Cyrix SMM mode.

3.6.1 SL-Compatible SMM Mode

While in SL-compatible mode, SMM memory space accesses can only occur during an SMI service routine. While executing an SMI service routine SMM_MEM remains asserted regardless of the address being accessed. This includes the time when the SMI service routine accesses memory outside the defined SMM memory space.

SMM memory caching is not supported in SL-compatible SMM mode. If a cache inquiry cycle occurs while SMM_MEM bit is active, any resulting write-back cycle is issued with SMM_MEM asserted. This occurs even though the write-back cycle is intended for normal memory rather than SMM memory. To avoid this problem it is recommended that the internal caches be flushed prior to servicing an SMI event. Of course in write-back mode this could add an indeterminate delay to servicing of SMI.

An interrupt on the SMI# input pin has higher priority than the NMI input. The SMI# input pin is falling edge sensitive and is sampled on every rising edge of the processor input clock.

Asserting SMI# forces the processor to save the CPU state to memory defined by SMHR register and to begin execution of the SMI service routine at the beginning of the defined SMM memory space. After the processor

internally acknowledges the SMI# interrupt, the SMM_MEM output is asserted for the duration of the interrupt service routine.

When the RSM instruction is executed, the CPU negates the SMM_MEM bit after the last bus cycle to SMM memory. While executing the SMM service routine, one additional SMI# can be latched for service after resuming from the first SMI.

3.6.2 Cyrix Enhanced SMM Mode

The Cyrix SMM Mode is enabled when bit0 in the CCR6 (SMM_MODE) is set. Only in Cyrix enhanced SMM mode can SMM memory be chached.

3.6.2.1 Pin Interface

The SMI# pin and SMM acknowledge special cycle behave the same in Cyrix Enhanced SMM mode.

3.6.2.2 Cacheability of SMM Space

In SL-compatible SMM mode, caching is not available, but in Cyrix SMM mode, both code and data caching is supported. In order to cache SMM data and avoid coherency issues the processor assumes no overlap of main memory with SMM memory. This implies that a section of main memory must be dedicated for SMM.

4 SMM Programming Details

This section provides detailed SMM information and programming examples.

4.1 Initializing SMM

Many systems have memory controllers that aid in the initialization of SMM memory. Cyrix SMM features allow the initialization of SMM memory without external hardware memory remapping.

When loading SMM memory with an SMM interrupt handler it is important that the SMI# does not occur before the handler is loaded.

To load SMM memory with a program it is first necessary to enable SMM memory without enabling the SMI pins. The SMM region is physically mapped to allow memory access within the SMM region. A REP MOV instruction can then be used to transfer the program to SMM memory.

SMM space can be located anywhere in the 4-GByte address range. However, if the location of SMM space is above 1 MByte, the value in CS will truncate the segment above 16bits when stored from the stack. This would prohibit calls or interrupts from real mode without restoring the 32-bit features of the cpu because of the incorrect return address on the stack.

```
; load SMM memory from system memory (Cyrix SMM mode only)


include SMIMAC.INC
SMMBASE = 68000h
SMMSIZE = 4000h    ;SMM SIZE is 16K
SMI    = 1 shl 1

;interrupts should be disabled here
       mov             al, 0cdh     ;SMM base<A31-A24>
       out             22h, al      ;select
       mov             al, 00h      ;set high SMM address to 00
       out             23h, al      ;write value
       mov             al, 0ceh     ;index SMM base<A23-A16>
       out             22h, al      ;select
       mov             al, 06h      ;set mid SMM address to 06h
       out             23h, al      ;write value
       mov             al, 0cfh     ;SMM base<A15-A12> & SIZE
       out             22h, al      ;select
       mov             al, 083h     ;set SMM lower addr. 80h, 16K
       out             23h, al      ;write value
       mov             ax, SMMBASE shr 4
       mov             es, ax
       mov             edi, 0       ;es:di = start of the SMM area
       mov             esi, offset SMI_ROUTINE  ;start of copy of SMM
       mov             ax, seg SMI_ROUTINE     ;routine in main memory
       mov             ds, ax
       mov             ecx, (SMI_ROUTINE_LENGTH+3)/4  ;calc. length

; this line copies the SMM routine from DS:ESI to ES:EDI
       rep
       movs dword ptr es:[edi],dword ptr ds:[esi]
```

4.2 SMM Handler Entry State

Before entering an SMM routine, certain portions of the CPU state are saved at the top of SMM memory. To optimize the speed of SMM entry and exit, the CPU saves the minimum CPU state information necessary for an SMI interrupt handler to execute and return to the interrupted context.

The information is saved to the relocatable SMM header at the top of the defined SMM region (starting at SMM base + size - 30h) as shown in the Figure (page 22). Only the CS, EIP, EFLAGS, CR0, and DR7 are saved upon entry to SMM. Data accesses must use a CS segment override to save other registers and access data in SMM memory. To use any other segment register, the SMM programmer must first save the contents using the SVDC instruction for segment registers or MOV operations for general purpose registers (See Cyrix SMM instruction description on Page 25). It is possible to save all the CPU registers as needed. See Section 4.3 (Page 2-34) for an example of saving and restoring the entire CPU state.

Upon execution of a RSM instruction, control is returned to NEXT_IP. The value of NEXT_IP may need to be modified for restarting I/O instructions. This modification is a simple move of the CURRENT_IP value to the NEXT_IP location. Execution is then returned to the I/O instruction, rather than to the instruction after the I/O instruction.

This CURRENT_IP value is valid only if the instruction executing when the SMI occurred was an I/O instruction. The table below lists the SMM header information needed to restart an I/O instruction. The restarting of I/O instructions may also require modifications to the ESI, ECX and EDI depending on the instruction.

I/O Trap Information

BIT	DESCRIPTION	SIZE
H	HALT Indicator If = 1: The CPU was in a halt or shut down prior to serving the SMM interrupt. If = 0: The CPU was not in a halt or shut down prior to serving the SMM interrupt.	1 bit
S	Software SMM Entry Indicator S=1, if current SMM is the result of an SMINT instruction. (Impossible) S=0, if current SMM is not the result of an SMINT instruction.	1 bit
P	REP INSx/OUTSx Indicator If = 1: Current instruction does not have a REP prefix If = 0: Current instruction has a REP prefix	1 bit
I	IN, INSx, OUT, or OUTSx Indicator If = 1: Current instruction performed an I/O WRITE If = 0: Current instruction performed an I/O READ	1 bit
I/O Data Size	Indicates size of data for the trapped I/O write: 01h = byte 03h = word 0Fh = dword	2 Bytes
I/O Write Address	I/O Write Address Processor port used for the trapped I/O write.	2 Bytes
I/O Write Data	I/O Write Data Data associated with the trapped I/O write.	4 Bytes
ESI or EDI	Restored ESI or EDI value. Used when it is necessary to repeat a REP OUTSx or REP INSx instruction when one of the I/O cycles caused an SMI# trap.	4 Bytes

The EFLAGS, CR0 and DR7 registers are set to their reset values upon entry to the SMI handler. Resetting these registers has implications for setting breakpoints using the debug registers. Breakpoints in SMM address space cannot be set prior to the SMI interrupt using debug registers. A debugger will only be able to set a code break-

point using INT 3 outside of the SMM handler. See Section 4.9 (Page 2-42) for restrictions on debugging SMM code. Once the SMI has occurred and the debugger has control in SMM space, the debug registers can be used for the remainder of the SMI handler execution.

Upon SMM entry, I/O trap information is stored in the SMM memory space header. This information allows restarting of I/O instructions, as well as the easy emulation of I/O functions by the SMM handler. This data is valid only if the instruction executing when the SMI occurred was an I/O instruction. In the Cyrix III cpu, both I/O reads and I/O write traps result in valid I/O fields and current P and I field values.

If the H bit in the SMM header is set, a HLT instruction was being executed when the SMI occurred. To resume execution of the HLT instruction, the field NEXT-IP in the SMM header should be decremented by one before executing RSM instruction.

The values found in the I/O trap information fields are specified below for all cases.

Valid I/O Trap Cases

VALID CASES	P	I	I/O WRITE DATA SIZE	I/O WRITE ADDRESS	I/O WRITE DATA	ESI OR EDI
Not an I/O instruction	x	x	x	x	x	x
IN al	0	0	01h	I/O Address	xxxxxxxx	EDI
IN ax	0	0	03h	I/O Address	xxxxxxxx	EDI
IN eax	0	0	0Fh	I/O Address	xxxxxxxx	EDI
INSB	0	0	01h	I/O Address	xxxxxxxx	EDI
INSW	0	0	03h	I/O Address	xxxxxxxx	EDI
INSD	0	0	0Fh	I/O Address	xxxxxxxx	EDI
REP INSB	1	0	01h	I/O Address	xxxxxxxx	EDI
REP INSW	1	0	03h	I/O Address	xxxxxxxx	EDI
REP INSD	1	0	0Fh	I/O Address	xxxxxxxx	EDI
OUT al	0	1	01h	I/O Address	xxxxxxdd	ESI
OUT ax	0	1	03h	I/O Address	xxxxdddd	ESI
OUT eax	0	1	0Fh	I/O Address	dddddddd	ESI
OUTSB	0	1	01h	I/O Address	xxxxxxdd	ESI
OUTSW	0	1	03h	I/O Address	xxxxdddd	ESI
OUTSD	0	1	0Fh	I/O Address	dddddddd	ESI
REP OUTSB	1	1	01h	I/O Address	xxxxxxdd	ESI
REP OUTSW	1	1	03h	I/O Address	xxxxdddd	ESI
REP OUTSD	1	1	0Fh	I/O Address	dddddddd	ESI

Upon SMM entry, the CPU enters the state described in table below.

SMM Entry State

REGISTER	REGISTER CONTENT	COMMENTS
CS	SMM base	CS limit is set to 4 GBytes
EIP	0000 0000h	Begins execution at the base of SMM memory
EFLAGS	0000 0002h	Reset State
DR7	0000 0400h	Traps disabled

4.3 Maintaining the CPU State

The following registers are not automatically saved on SMM entry or restored on SMM exit.

General Purpose Registers:	EAX, EBX, ECX, EDX
Pointer and Index Registers:	EBP, ESI, EDI, ESP
Selector/Segment Registers:	DS, ES, SS, FS, GS
Descriptor Table Registers:	GDTR, IDTR, LDTR, TR
Control Registers:	CR2, CR3
Debug Registers:	DR0, DR1, DR2, DR3, DR6
Configuration Registers:	all valid configuration registers
FPU Registers:	Entire FPU state.

If the SMM routine will use any of these registers, their contents must be saved after entry into the SMM routine and then restored prior to exit from SMM. Additionally, if power is to be removed from the CPU and the system is required to return to the same system state after power is reapplied, then the entire CPU state must be saved to a non-volatile memory subsystem such as a hard disk.

4.3.1 Maintaining Common CPU Registers

The following is an example of the instructions needed to save the entire CPU state and restore it. This code sequence will work from real mode if the conditions needed to execute Cyrix SMM instructions are met (see Section 2.3). Configuration registers would also need to be saved if power is to be removed.

```
; Save and Restore the common CPU registers.
; The information automatically saved in the
; header on entry to SMM is not saved again.
include SMIMAC.INC

      .386P                     ;required for SMIMAC.INC macro
      mov     cs:save_eax,eax
      mov     cs:save_ebx,ebx
      mov     cs:save_ecx,ecx
      mov     cs:save_edx,edx
      mov     cs:save_esi,esi
      mov     cs:save_edi,edi
      mov     cs:save_ebp,ebp
      mov     cs:save_esp,esp
      svdc    cs:,save_ds,ds
      svdc    cs:,save_es,es
      svdc    cs:,save_fs,fs
      svdc    cs:,save_gs,gs
      svdc    cs:,save_ss,ss
      svldt   cs:,save_ldt      ;sldt is not valid in real mode
      svts    cs:,save_tsr      ;str is not valid in real mode
      db      66h               ;32bit version saves everything
      sgdt    fword ptr cs:[save_gdt]
      db      66h               ;32bit version saves everything
      sidt    fword ptr cs:[save_idt]

; at the end of the SMM routine the following code
; sequence will reload the entire CPU state
      mov     eax,cs:save_eax
      mov     ebx,cs:save_ebx
      mov     ecx,cs:save_ecx
      mov     edx,cs:save_edx
      mov     esi,cs:save_esi
      mov     edi,cs:save_edi
      mov     ebp,cs:save_ebp
      mov     esp,cs:save_esp
      rsdc    ds,cs:,save_ds
      rsdc    es,cs:,save_es
      rsdc    fs,cs:,save_fs
      rsdc    gs,cs:,save_gs
      rsdc    ss,cs:,save_ss
      rsldt   cs:,save_ldt
      rsts    cs:,save_tsr
```

```
        db      66h
        lgdt    fword ptr cs:[save_gdt]
        db      66h
        lidt    fword ptr cs:[save_idt]

; the data space so save the CPU state is in
; the Code Segment for this example
save_ds   dt      ?
save_es   dt      ?
save_fs   dt      ?
save_gs   dt      ?
save_ss   dt      ?
save_ldt  dt      ?
save_tsr  dt      ?
save_eax  dd      ?
save_ebx  dd      ?
save_ecx  dd      ?
save_edx  dd      ?
save_esi  dd      ?
save_edi  dd      ?
save_ebp  dd      ?
save_esp  dd      ?
save_gdt  df      ?
save_idt  df      ?
```

4.3.2 Maintaining Control Registers

CR0 is maintained in the SMM header. CR2 and CR3 should be saved if the SMM routine will be entering protected mode and enabling paging. Most SMM routines will not need to enable paging. However, if the CPU will be powered off, these registers should be saved.

4.3.3 Maintaining Debug Registers

DR7 is maintained in the SMM Header. Since DR7 is automatically initialized to the reset state on entry to SMM, the Global Disable bit (DR7 bit 13) will be cleared. This allows the SMM routine to access all of the Debug Registers. Returning from the SMM handler will reload DR7 with its previous value. In most cases, SMM routines will not make use of the Debug Registers and they will need to be saved only if the CPU needs to be powered down.

4.3.4 Maintaining Configuration Control Registers

The SMM routine should be written so that it maintains the Configuration Control Registers in the same state as they were initialized by the BIOS at power-up.

4.3.5 Maintaining FPU State

If power will be removed from the CPU or if the SMM routine will execute FPU instructions, then the FPU state should be maintained for the application running before SMM was entered. If the FPU state is to be saved and restored from within SMM, there are certain guidelines that must be followed to make SMM completely transparent to the application program.

The complete state of the FPU can be saved and restored with the FNSAVE and FNRSTOR instructions. FNSAVE is used instead of the FSAVE because FSAVE will wait for the FPU to check for existing error conditions before storing the FPU state. If there is an unmasked FPU exception condition pending, the FSAVE instruction will wait until the exception condition is serviced. To maintain transparency for the application program, the SMM routine should not service this exception. If the FPU state is restored with the FNRSTOR instruction before returning to normal mode, the application program can correctly service the exception. Any FPU instructions can be executed within SMM once the FPU state has been saved.

The information saved with the FSAVE instruction varies depending on the operating mode of the CPU. To save and restore all FPU information, the 32-bit protected mode version of the FPU save and restore instruction should be used. This can be accomplished by using the following code example:

```
; Save the FPU state
      mov    eax,CR0
      or     eax,00000001h
      mov    CR0,eax            ;set the PE bit in CR0
      jmp    $+2                ;clear the prefetch que
      db     66h                ;do 32bit version of fnsave
      fnsave [save_fpu]         ;saves fpu state to
                                ;the address DS:[save_fpu]
      mov    eax,CR0
      and    eax, 0FFFFFFFEh    ;clear PE bit in CR0
      mov    CR0,eax            ;return to real mode

;now the SMM routine can do any FPU instruction.
;Restore the FPU state before executing a RSM
      FNINIT                    ;initialize the FPU to a valid state
      mov    eax,CR0
      or     eax,00000001h
      mov    CR0,eax            ;set the PE bit in CR0
      jmp    $+2                ;clear the prefetch que
      db     66h                ;do 32bit version of fnsave
      frstor [save_fpu]         ;restore the FPU state
                                ;Some assemblers may require
                                ;use of the fnrstor instruction
      mov    eax,CR0
      and    eax, 0FFFFFFFEh    ;clear PE bit in CR0
      mov    CR0,eax            ;return to real mode
```

Be sure that all interrupts are disabled before using this method for entering protected mode. Any attempt to load a selector register while in protected mode will shutdown the CPU since no GDT is set up. Setting up a GDT and doing a long jump to enter protected mode will also work correctly.

4.4 Initializing the SMM Environment

After entering SMM and saving the CPU registers that will be used by the SMM routine, a few registers need to be initialized.

Segment registers need to be initialized if the CPU was operating in protected mode when the SMI interrupt occurred. Segment registers that will be used by the SMM routine should be loaded with known limits before they are used. The protected mode application could have set a segment limit to less than 64K. To avoid a protec-

tion error, all segment registers can be given limits of 4 GBytes. This can be done with the Cyrix RSDC instruction and will allow access to the full 4GBytes of possible system memory without entering protected mode. Once the limits of a segment register are set, the base can be changed by use of the MOV instruction.

If necessary, an Interrupt Descriptor Table (IDT) should be set up in SMM memory before any interrupts or exceptions occur. The Descriptor Table Register can be loaded with an LIDT instruction to point to a small IDT in SMM memory that can handle the possible interrupts and exceptions that might occur while in the SMM routine.

A stack should always be set up in SMM memory so that stack operations done within SMM do not affect the system memory.

```
; SMM environment initialization example
include SMIMAC.INC              ; see Appendix A
      rsdc   ds,cs:,seg4G       ;DS is a 4GByte segment, base=0
      rsdc   es,cs:,seg4G       ;ES is a 4GByte segment, base=0
      rsdc   fs,cs:,seg4G       ;FS is a 4GByte segment, base=0
      rsdc   gs,cs:,seg4G       ;GS is a 4GByte segment, base=0
      rsdc   ss,cs:,seg4G       ;SS is a 4GByte segment, base=0
      lidt   cs:smm_idt         ;load IDT base and limit for
                                ;SMM's IDT
      mov    esp, smm_stack
      jmp    continue_smm_code
;
;descriptor of 4GByte data segment for use by rsdc
seg4G     dw        0ffffh        ; limit 4G
          dw        0             ; base = 0
          db        0             ; base = 0
          db        10010011B     ; data segment, DPL=0,P=1
          db        8fh           ; limit = 4G,
          db        0h            ; base = 0
          dw        0             ; segment register = 0
smm_idt   dw        smm_idt_limit
          dd        smm_idt_base
```

4.5 I/O Restart

Often when implementing a power management design, peripherals are required to be powered down by the system when not in use. When an I/O instruction is issued to a powered down device, the SMM routine is called to power up the peripheral and then reissue the I/O instruction. Cyrix CPUs make it easy to restart the I/O instruction that has generated an SMI interrupt.

The system will generate an SMI interrupt when an I/O bus cycle to a powered-down peripheral is detected. The SMM routine should interrogate the system hardware to find out if the SMI was caused by an I/O trap. By checking the SMM header information, the SMM routine can determine the type of I/O instruction that was trapped. If the I/O instruction has a REP prefix, the ECX register needs to be incremented before restarting the instruction. If the I/O trap was on a string I/O instruction, the ESI or EDI registers must be restored to their previous value before restarting the instruction.
The following code example shows how easy I/O restart is with the Cyrix CPU.

```
include SMIMAC.INC              ;see Appendix A
;Restart the interrupted instruction
      mov    eax,dword ptr cs:[SMI_CURRENTIP]
      mov    dword ptr cs:[SMI_NEXTIP],eax
      mov    al,byte ptr cs:[SMI_BITS]

;test for REP instruction
      bt     ax,2               ;rep instruction?
                                ;(result to Carry)
      adc    ecx,0              ;if so, increment ecx
      test   al,1 shl 1         ;test bit 1 to see
                                ;if an OUTS or INS
      jnz    out_instr
; A port read (INS or IN) instruction caused the
; chipset to generate an SMI instruction.
; Restore EDI from SMM header.
      mov    edi, dword ptr cs:[SMI_ESIEDI]
      jmp    common1

; A port write (OUTS or OUT) instruction caused the
; chipset to generate an SMI instruction.
; Restore ESI from SMM header.
out_instr:
      mov    esi, dword ptr cs:[SMI_ESIEDI]
common1:
```

4.6 I/O Port Shadowing and Emulation

Some system peripherals contain write-only ports. In a system that performs power management, these peripherals need to be powered off and then reinitialized when their functions are needed later. The Cyrix SMM implementation makes it very easy to monitor the last value written to specific I/O ports. This process is known as

shadowing. If the system can generate an SMI whenever specific I/O addresses get accessed, the SMM routine can, transparently to the system, monitor the port activity. The SMM header contains the address of the I/O write as well as the data. In addition, information is saved which indicates whether it is a byte, word or dword write. With this information, shadowing system write-only ports becomes trivial.

Some peripheral components contain registers that must be programmed in a specific order. If an SMI interrupt occurs while an application is accessing this type of peripheral, the SMI routine must be sure to reload the peripheral registers to the same stage before returning to normal mode. If the SMM routine needs to access such a peripheral, the previous normal-mode state must be restored. The previous accesses that were shadowed by previous SMM calls can be used to reload the peripheral registers back to the stage where the application was interrupted. The application can then continue where it left off accessing the peripheral.

In a similar way, the Cyrix SMM implementation allows the SMM routine to emulate the function of peripheral components in software.

4.7 Resume to HLT Instruction

To make an SMI interrupt truly transparent to the system, an SMI interrupt from a HLT instruction should return to the HLT instruction. There are known cases with DOS software where returning from an SMI handler to the instruction following the HLT will cause a system error. To determine if a HLT instruction was interrupted by the SMI, the H bit in the SMM header must be interrogated. If the H bit is set, the SMI interrupted a HLT instruction. To restart the HLT instruction simply decrement the NEXT_IP field in the SMM header.

The H bit is not available on a Cx486DX2/DX4.

```
;This is the start of specific code to check if the SMI
;occurred while in a HLT instruction.  If it did, then
;resume back to the HLT instruction when SMI is finished.

      include SMIMAC.INC                        ;see Appendix A


      mov         ax,cs:word ptr[SMI_BITS]      ;get H bit
      test        ax,0010h                      ;check if H=1
      je          not_hlt                       ;was not a HLT
      dec         cs:dword ptr[SMI_NEXTIP]      ;decrement NEXT_IP
not_hlt:
```

4.8 Exiting the SMI Handler

When the RSM instruction is executed at the end of the SMI handler, the EIP is loaded from the SMM header at the address (SMMbase + SMMsize - 14h) called NEXT_IP. This permits the instruction to be restarted if NEXT_IP was modified by the SMM program. The values of ECX, ESI, and EDI, prior to the execution of the instruction that was interrupted by SMI, can be restored from information in the header which pertains to the INx and OUTx instructions. See Section 3.6 for an example program to restart an I/O instruction. The only registers that are restored from the SMM header are CS, NEXT_IP, EFLAGS, CR0, and DR7. All other registers which were modified by the SMM program need to be restored before executing the RSM instruction.

4.9 Testing and Debugging SMM Code

An SMI routine can be debugged with standard debugging tools, such as DOS DEBUG, if the following requirements are met:

1. The debugger will only be able to set a code break point using INT 3 outside of the SMI handler. The debug control register DR7 is set to the reset value upon entry to the SMI handler. Therefore, any break conditions in DR0-3 will be disabled after entry to SMM. Debug registers can be used if they are set after entry to the SMI handler and if debug registers DR0-3 are saved.
2. The debugger must be running in real mode and the SMM routine must not enter protected mode. This insures that normal system interrupts, BIOS calls and the debugger will work correctly from SMM mode.
3. Before an INT 3 break point is executed, all segment registers should have their limits modified to 64K, or larger, within the SMM routine.

APPENDIX A

5 *Assembler Macros For Cyrix Instructions*

The include file, SMIMAC.INC, provides a complex set of macros which generate SMM opcodes along with the appropriate mod/rm bytes. In order to function, the macros require that the labels which are accessed correspond to the specified segment. Thus segment overrides must be passed to the macro as an argument.

Do not specify a segment override if the default segment for an address is being used. If an address size override is used, a final argument of '1' must be passed to the macro as well. Address size overrides must be presented explicitly to prevent the assembler from generating them automatically and breaking the macros.

```
;SMM Instruction Macros - SMIMAC.INC
;Macros which generate mod/rm automatically

svdc    MACRO   segover,addr,reg,adover
        domac   segover,addr,reg,adover,78h
        ENDM
rsdc    MACRO   reg,segover,addr,adover
        domac   segover,addr,reg,adover,79h
        ENDM
svldt   MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7ah
        ENDM
rsldt   MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7bh
        ENDM
svts    MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7ch
        ENDM
rsts    MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7dh
        ENDM
rsm     MACRO
        db      0fh,0aah
        ENDM
smint   MACRO
        db      0fh,7eh
        ENDM
;Sub-Macro used by the above macro
```

```
domac   MACRO   segover,addr,reg,adover,op
        local   place1,place2,count
        count   = 0
        ifnb    <adover>
            count=count+1
        endif
        ifnb   <segover>
            count=count+1
        endif
        if      (count eq 0)
                nop             ;expanding the opcode one byte
        endif
        place1  = $
;pull off the proper prefix byte count
        mov     word ptr segover addr,reg
        org     place1+count
        mov     word ptr segover addr,reg
        place2  = $
;patch the opcode
        org     place1+(count*2)-1
        db      0Fh,op
        org     place2
ENDM

;Offset Definition for access into SMM space
SMI_SAVE STRUC
$ESIEDI         DD      ?
$IOWDATA        DD      ?
$IOWADDR        DW      ?
$IOWSIZE        DW      ?
$BITS           DD      ?
$CSSELL         DD      ?
$CSSELH         DD      ?
$CS             DW      ?
$RES1           DW      ?
$NEXTIP         DD      ?
$CURRENTIP      DD      ?
$CR0            DD      ?
$EFLAGS         DD      ?
$DR7            DD      ?
SMI_SAVE ENDS
SMI_ESIEDI      EQU  ($ESIEDI + SMMSIZE - SIZE SMI_SAVE)
SMI_IOWDATA     EQU  ($IOWDATA+ SMMSIZE - SIZE SMI_SAVE)
SMI_IOWADDR     EQU  ($IOWADDR+ SMMSIZE - SIZE SMI_SAVE)
SMI_IOWSIZE     EQU  ($IOWSIZE+ SMMSIZE - SIZE SMI_SAVE)
```

```
SMI_BITS        EQU  ($BITS   + SMMSIZE - SIZE SMI_SAVE)
SMI_CSSELL      EQU  ($CSSELL + SMMSIZE - SIZE SMI_SAVE)
SMI_CSSELH      EQU  ($CSSELH + SMMSIZE - SIZE SMI_SAVE)
SMI_CS          EQU  ($CS     + SMMSIZE - SIZE SMI_SAVE)
SMI_RES1        EQU  ($RES1   + SMMSIZE - SIZE SMI_SAVE)
SMI_NEXTIP      EQU  ($NEXTIP + SMMSIZE - SIZE SMI_SAVE)
SMI_CURRENTIP   EQU  ($CURRENTIP+ SMMSIZE -SIZE SMI_SAVE)
SMI_CR0         EQU  ($CR0    + SMMSIZE - SIZE SMI_SAVE)
SMI_EFLAGS      EQU  ($EFLAGS + SMMSIZE - SIZE SMI_SAVE)
SMI_DR7         EQU  ($DR7    + SMMSIZE - SIZE SMI_SAVE)
```

SMM Instruction macro example: TEST.ASM

```
                                .MODEL  SMALL
                                .386
                                ;SMM Macro Examples


                                include smimac.inc

0000                          .DATA
0000  0A*(??)                 there   db      10 dup (?)
000A                          .CODE
0000  2E 0F 78 1E 004E                svdc    cs:,hello,ds
0006  2E 0F 79 1E 004E                rsdc    ds,cs:,hello
000C  2E 0F 79 2E 004E                rsdc    gs,cs:,hello
0012  2E 67 2E 0F 78 9C 58 0000004E svdc    cs:,[eax+ebx*2+hello],1
001D  67| 0F 78 23                   svdc    ,[ebx],fs,1

0021  0F 78 2E 0000                  svdc    ,there,gs
0026  2E 0F 7A 06 004E               svldt   cs:,hello
002C  2E 0F 7B 06 004E               rsldt   cs:,hello

0032  2E 0F 7D 06 004E               rsts    cs:,hello
0038  2E 67 2E 0F 7C 84 58 0000004E svts    cs:,[eax+ebx*2+hello],1
0043  67| 0F 7A 03                   svldt   ,[ebx],1
0047  0F 7C 06 0000                  svts    ,there
004C  0F AA                           rsm

004E  0A*(??)                         hello   db   10 dup (?)
end
```

APPENDIX B

6 Differences in Cyrix Processors

The table below lists the major differences between the 6x86, 6x86MX M II, and Cyrix III CPUs as related to System Management Mode.

Differences Between Cyrix CPUs

FEATURE	6x86	6x86MXI	MII AND MII MOBILE	CYRIX III
SMAC CCR1 - bit 2	Available	Available	Available	Not Available
MMAC CCR1 - bit 3	Not available	Valid only. if SMM_MODE=0.	Valid only. if SMM_MODE=0.	Not Available.
SM3 CCR1 - bit 7	Must be set to define register index CDh CEh and CFh as SMAR.	Must be set to define register index CDh CEh and CFh as SMAR.	Must be set to define register index CDh CEh and CFh as SMAR.	Must be set to define register index CDh CEh and CFh as SMAR.
SMIACT CCR3 - bit3	Always in SL SMM mode.	Use SMM_MODE CCR6_Bit 0	Use SMM_MODE CCR6_Bit 0	Not Available
SMI# acknowledged when:	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & SMAC=0 & (in normal mode)	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & SMAC=0 & (in normal mode)	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & SMAC=0 & (in normal mode)	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & (in normal mode)
Cyrix Specific SMM instructions are valid when:	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & (SMAC=1 or in SMM mode)	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & (SMAC=1 or in SMM mode)	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & (SMAC=1 or in SMM mode)	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & (in SMM mode)
H bit in SMM header	Valid	Valid	Valid	Valid
I/O trap information	I/O Data Size, I/O Address and I/O Data valid for both I/O reads and writes trapped by an SMI.	I/O Data Size, I/O Address and I/O Data valid for both I/O reads and writes trapped by an SMI.	I/O Data Size, I/O Address and I/O Data valid for both I/O reads and writes trapped by an SMI.	I/O Data Size, I/O Address and I/O Data valid for both I/O reads and writes trapped by an SMI.
CS limit on entry to SMM	4 GByte limit	4 GByte limit	4 GByte limit	4 GByte limit
CR0 value on entry to SMM	6000 0010h if LOCK_NW=1 then NW is not changed	6000 0010h if LOCK_NW=1 then NW is not changed	6000 0010h if LOCK_NW=1 then NW is not changed	6000 0010h if LOCK_NW=1 then NW is not changed

©1999 Copyright Via Technologies. All rights reserved.

Printed in the United States of America

Trademark Acknowledgments:

Cyrix is a registered trademark of Via Technologies.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Order Number: 94xxx-xx

Via-Cyrix

2703 North Central Expressway

Richardson, Texas 75080-2010

United States of America

Via-Cyrix (Cyrix) reserves the right to make changes in the devices or specifications described herein without notice. Before design-in or order placement, customers are advised to verify that the information is current on which orders or design activities are based. Cyrix warrants its products to conform to current specifications in accordance with Cyrix' standard warranty. Testing is performed to the extent necessary as determined by Cyrix to support this warranty. Unless explicitly specified by customer order requirements, and agreed to in writing by Cyrix, not all device characteristics are necessarily tested. Cyrix assumes no liability, unless specifically agreed to in writing, for customers' product design or infringement of patents or copyrights of third parties arising from use of Cyrix devices. No license, either express or implied, to Cyrix patents, copyrights, or other intellectual property rights pertaining to any machine or combination of Cyrix devices is hereby granted. Cyrix products are not intended for use in any medical, life saving, or life sustaining system. Information in this document is subject to change without notice.