Advanced x86:
BIOS and System Management Mode Internals
System Management Mode (SMM)

Xeno Kovah && Corey Kallenberg

LegbaCore, LLC

All materials are licensed under a Creative Commons “Share Alike” license.

http://creativecommons.org/licenses/by-sa/3.0/

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

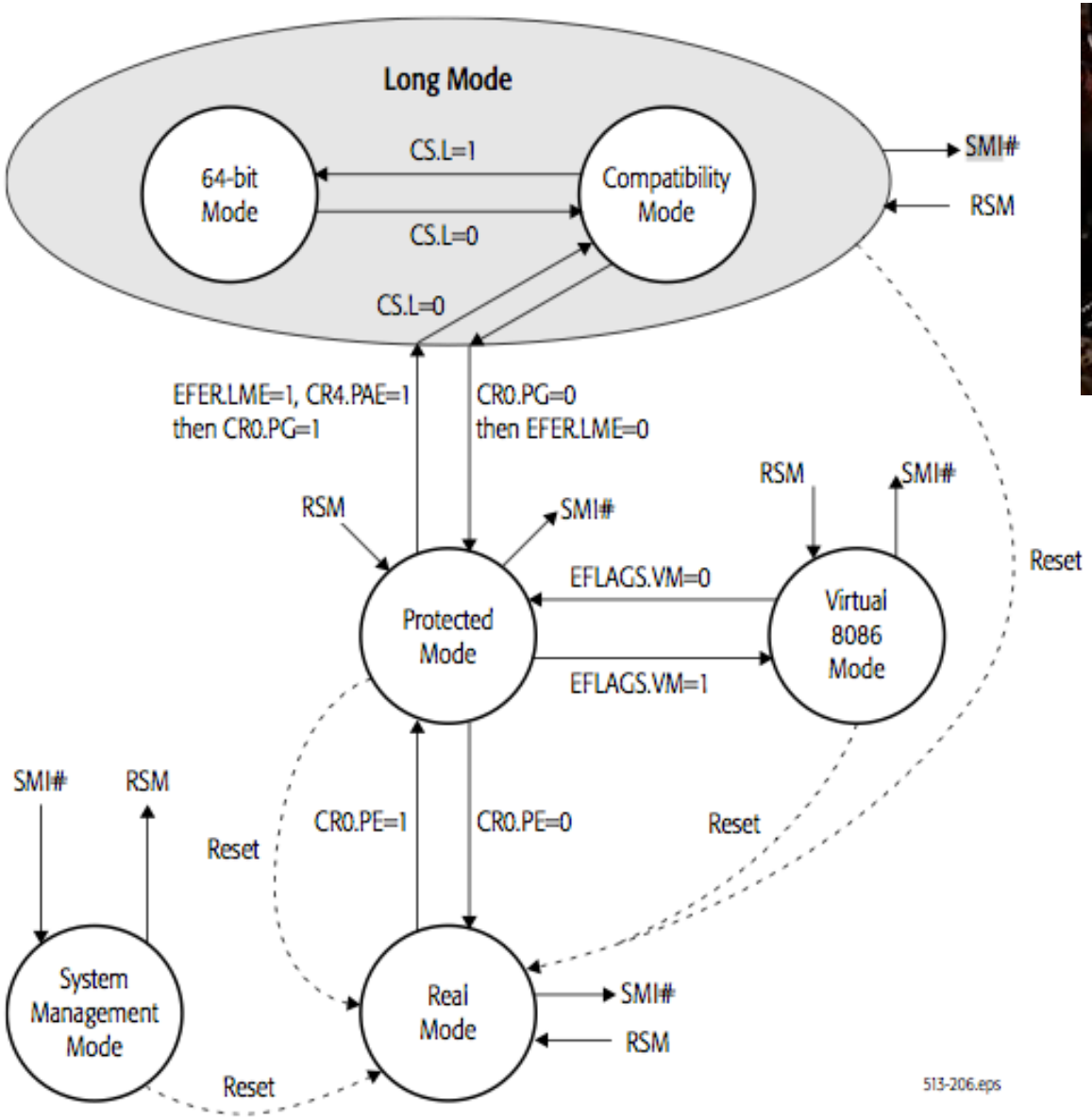
Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work
"Is derived from John Butterworth & Xeno Kovah’s ’Advanced Intel x86: BIOS and SMM’ class posted at http://opensecuritytraining.info/IntroBIOS.html”

System Management Mode (SMM)

God Mode Activate



Batteries Not Included!

Figure 1-6. Operating Modes of the AMD64 Architecture

From http://support.amd.com/us/Processor_TechDocs/24593.pdf

System Management Mode (SMM) Overview

- Most privileged x86 processor operating mode
- Runs transparent to the operating system
- When the processor enters SMM, all other running tasks are suspended
- SMM can be invoked *only* by a System Management Interrupt (SMI) and exited *only* by the RSM (resume) instruction
- Intended use is to provide an isolated operating environment for
  - Power/Battery management
  - Controlling system hardware
  - Running proprietary OEM code
  - etc. (anything that should run privileged and uninterrupted)

System Management Mode (SMM) Overview

- The code that executes in SMM (called the SMI handler) is instantiated from the BIOS flash
- Protecting SMM is a matter of protecting both the active (running) SMRAM address space but also protecting the flash chip from which it is derived
  - Protect itself (SMBASE (location), SMRAM Permissions)
  - Write-Protect Flash
- So far in our research, only about 5% of SMRAM configurations were directly unlocked and vulnerable to overwrite
- However, since > 50% of the BIOS flash chips we've seen are vulnerable, that means > 50% of SMRAM will follow suit

System Management Interrupt (SMI)

- SMM can only be invoked by signaling a System Management Interrupt (SMI)
- SMI's can be received via the SMI# pin on the processor or through the APIC bus
- SMI's cannot be masked like normal interrupts (e.g. with the "cli" instruction, or clearing the IF bit in EFLAGS)
- SMI's are independent from the normal processor interrupt and exception-handling mechanisms
- SMI's take precedence over non-maskable and maskable interrupts
  - Including debug exceptions and external interrupts
- If a SMI and NMI occur at the same time, only the SMI will be handled

Causes of SMI#

Just an example, your ICH/PCH will list them for your system

Causes of SMI# and SCI (Sheet 1 of 2)

Cause	SCI	SMI	Additional Enables	Where Reported
PME#	Yes	Yes	PME_EN=1	PME_STS
PME_B0 (Internal, Bus 0, PME-Capable Agents)	Yes	Yes	PME_B0_EN=1	PME_B0_STS
PCI Express* PME Messages	Yes	Yes	PCI_EXP_EN=1 (Not enabled for SMI)	PCI_EXP_STS
PCI Express Hot Plug Message	Yes	Yes	HOT_PLUG_EN=1 (Not enabled for SMI)	HOT_PLUG_STS
Power Button Press	Yes	Yes	PWRBTN_EN=1	PWRBTN_STS
Power Button Override (Note 7)	Yes	No	None	PRBTNOR_STS
RTC Alarm	Yes	Yes	RTC_EN=1	RTC_STS
Ring Indicate	Yes	Yes	RI_EN=1	RI_STS
USB#1 wakes	Yes	Yes	USB1_EN=1	USB1_STS
USB#2 wakes	Yes	Yes	USB2_EN=1	USB2_STS
USB#3 wakes	Yes	Yes	USB3_EN=1	USB3_STS
USB#4 wakes	Yes	Yes	USB4_EN=1	USB4_STS
USB#5 wakes	Yes	Yes	USB5_EN=1	USB5_STS
USB#6 wakes	Yes	Yes	USB6_EN=1	USB6_STS
THRM# pin active	Yes	Yes	THRM_EN=1	THRM_STS
ACPI Timer overflow (2.34 sec.)	Yes	Yes	TMROF_EN=1	TMROF_STS
Any GPI	Yes	Yes	GPI[x]_Route=10 (SCI) GPI[x]_Route=01 (SMI) GPE0[x]_EN=1	GPI[x]_STS GPE0_STS
TCO SCI Logic	Yes	No	TCOSCI_EN=1	TCOSCI_STS
TCO SCI message from (G)MCH	Yes	No	none	MCHSCI_STS
TCO SMI Logic	No	Yes	TCO_EN=1	TCO_STS
TCO SMI — Year 2000 Rollover	No	Yes	none	NEWCENTURY_STS
TCO SMI — TCO TIMEROUT	No	Yes	none	TIMEOUT
TCO SMI — OS writes to TCO_DAT_IN register	No	Yes	none	OS_TCO_SMI
TCO SMI — Message from (G)MCH	No	Yes	none	MCHSMI_STS
TCO SMI — NMI occurred (and NMIs mapped to SMI)	No	Yes	NMI2SMI_EN=1	NMI2SMI_STS
TCO SMI — INTRUDER# signal goes active	No	Yes	INTRD_SEL=10	INTRD_DET
TCO SMI[8] — Change of the BIOSWP bit from 0 to 1	No	Yes	BC.LE=1	BIOSWR_STS

Causes of SMI# and SCI (Sheet 2 of 2)

Cause	SCI	SMI	Additional Enables	Where Reported
TCO SMI — Write attempted to BIOS	No	Yes	BIOSWP=1	BIOSWR_STS
BIOS_RLS written to	Yes	No	GBL_EN=1	GBL_STS
GBL_RLS written to	No	Yes	BIOS_EN=1	BIOS_STS
Write to B2h register	No	Yes	APMC_EN = 1	APM_STS
Periodic timer expires	No	Yes	PERIODIC_EN=1	PERIODIC_STS
64 ms timer expires	No	Yes	SWSMI_TMR_EN=1	SWSMI_TMR_STS
Enhanced USB Legacy Support Event	No	Yes	LEGACY_USB2_EN = 1	LEGACY_USB2_STS
Enhanced USB Intel Specific Event	No	Yes	INTEL_USB2_EN = 1	INTEL_USB2_STS
UHCI USB Legacy logic	No	Yes	LEGACY_USB_EN=1	LEGACY_USB_STS
Serial IRQ SMI reported	No	Yes	none	SERIRQ_SMI_STS
Device monitors match address in its range	No	Yes	none	DEVTRAP_STS
SMBus Host Controller	No	Yes	SMB_SMI_EN Host Controller Enabled	SMBus host status reg.
SMBus Slave SMI message	No	Yes	none	SMBUS_SMI_STS
SMBus SMBALERT# signal active	No	Yes	none	SMBUS_SMI_STS
SMBus Host Notify message received	No	Yes	HOST_NOTIFY_INTREN	SMBUS_SMI_STS HOST_NOTIFY_STS
Access microcontroller 62h/66h	No	Yes	MCSMI_EN	MCSMI_STS
SLP_EN bit written to 1	No	Yes	SMI_ON_SLP_EN=1	SMI_ON_SLP_EN_STS
USB Per-Port Registers Write Enable bit changes to 1.	No	Yes	USB2_EN=1, Write_Enable_SMI_Enable=1	USB2_STS, Write Enable Status
Write attempted to BIOS	No	Yes	BIOSWPD = 0	BIOSWR_STS
GPIO Lockdown Enable bit changes from '1' to '0'.	No	Yes	GPIO_UNLOCK_SMI_EN=1	GPIO_UNLOCK_SMI_STS

Generating SMI: APM

Fixed I/O Ranges Decoded by Intel® ICH9 (Sheet 2 of 2)

I/O Address	Read Target	Write Target	Internal Unit
84h–86h	DMA Controller	DMA Controller and LPC or PCI	DMA
87h	DMA Controller	DMA Controller	DMA
B2h–B3h	Power Management	Power Management	Power Management

- This is applicable to systems that support Advanced Power Management (most do these days)
- Fixed I/O range, so it cannot be relocated
- Check your I/O Controller Hub datasheet to verify its supported
- Writes to 0xB3 do not trigger an SMI#, only the write to 0xB2
- 0xB3 can be used to pass information

Advanced Power Management (APM)

13.8.2 APM I/O Decode

Table 13-10 shows the I/O registers associated with APM support. This register space is enabled in the PCI Device 31: Function 0 space (APMDEC_EN), and cannot be moved (fixed I/O location).

Table 13-10. APM Register Map

Address	Mnemonic	Register Name	Default	Type
B2h	APM_CNT	Advanced Power Management Control Port	00h	R/W
B3h	APM_STS	Advanced Power Management Status Port	00h	R/W

- APM_CNT (0xB2) is the control register
- APM_STS (0xB3) is the status register
- Located in I/O Address space
- Registers are R/W
- Note: APM != ACPI, but even on other systems which use ACPI 2.0, this still triggers an SMI#
  - The PCH datasheets (up to 8-series) also still list these under the fixed IO address

APM_CNT & APM_STS

APM_CNT—Advanced Power Management Control Port Register

I/O Address:	B2h	Attribute:	R/W
Default Value:	00h	Size:	8-bit
Lockable:	No	Usage:	Legacy Only
Power Well:	Core		

Bit	Description
7:0	Used to pass an APM command between the OS and the SMI handler. Writes to this port not only store data in the APMC register, but also generates an SMI# when the APMC_EN bit is set.

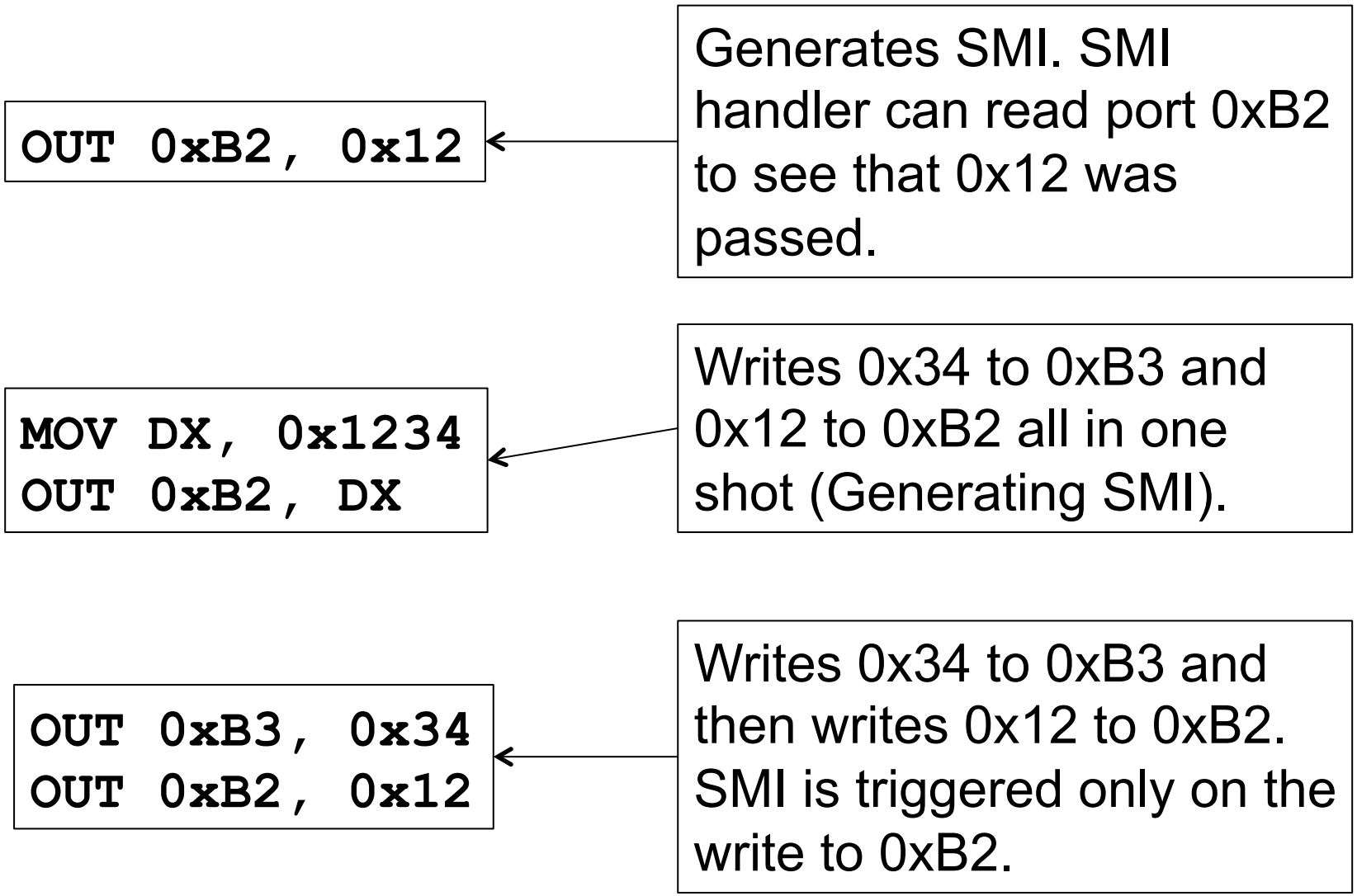
APM_STS—Advanced Power Management Status Port Register

I/O Address:	B3h	Attribute:	R/W
Default Value:	00h	Size:	8-bit
Lockable:	No	Usage:	Legacy Only
Power Well:	Core		

Bit	Description
7:0	Used to pass data between the OS and the SMI handler. Basically, this is a scratchpad register and is not affected by any other register or function (other than a PCI reset).

- Writing a byte to port 0xB2 will trigger an SMI#
- Writing to 0xB3 does NOT trigger SMI#
- Can be used to pass information to the SMI handler.
- Control flow through the SMI handler can be determined by the values in ports B2 & B3h
- Usage: Could tell the SMI handler to measure Hypervisor memory, or initiate a BIOS update

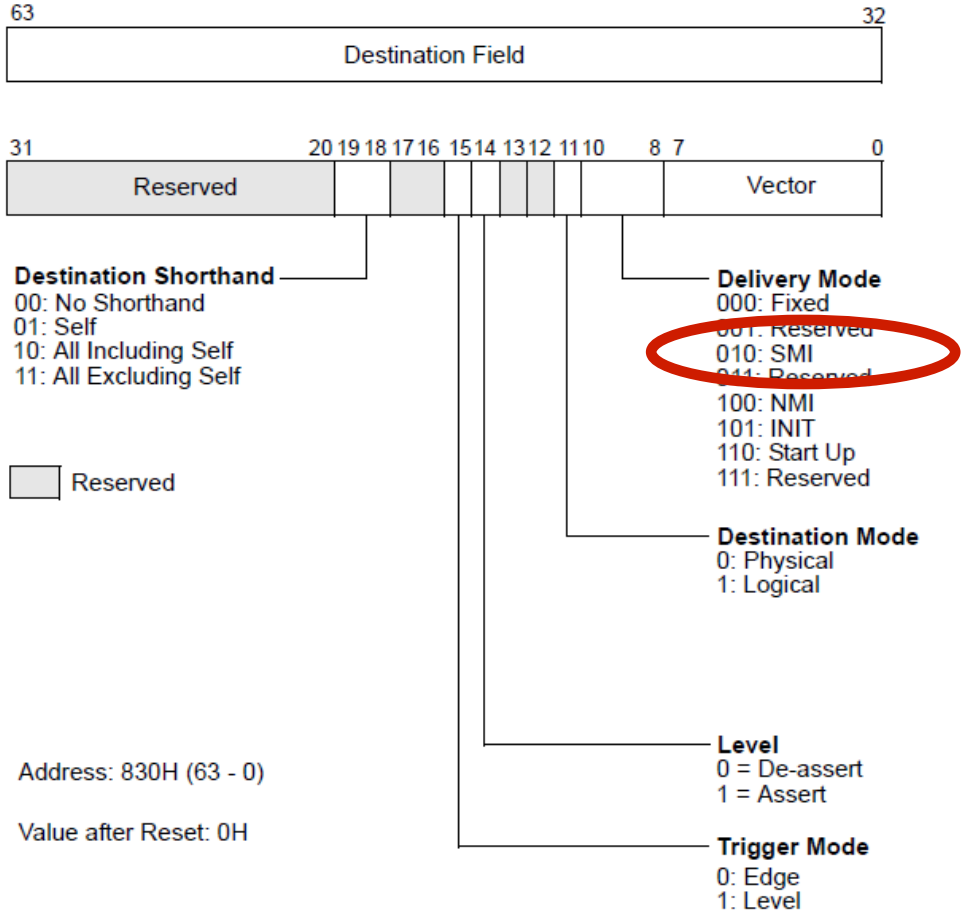
Examples


OUT 0xB2, 0x12
Generates SMI. SMI handler can read port 0xB2 to see that 0x12 was passed.
MOV DX, 0x1234
OUT 0xB2, DX
Writes 0x34 to 0xB3 and 0x12 to 0xB2 all in one shot (Generating SMI).
OUT 0xB3, 0x34
OUT 0xB2, 0x12
Writes 0x34 to 0xB3 and then writes 0x12 to 0xB2. SMI is triggered only on the write to 0xB2.

Generating SMI via APIC

Table 10-6. Local APIC Register Address Map Supported by x2APIC (Contd.)

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
830H[4]	300H and 310H	Interrupt Command Register (ICR)	Read/write	See Figure 10-28 for reserved bits



- We've not yet had time to play with this
- Should be able to generate SMI by programming the Interrupt Command Register in the APIC
- Architecture (and APIC type) dependent
- There is also a Self IPI register

“Corollary” SMI# generation

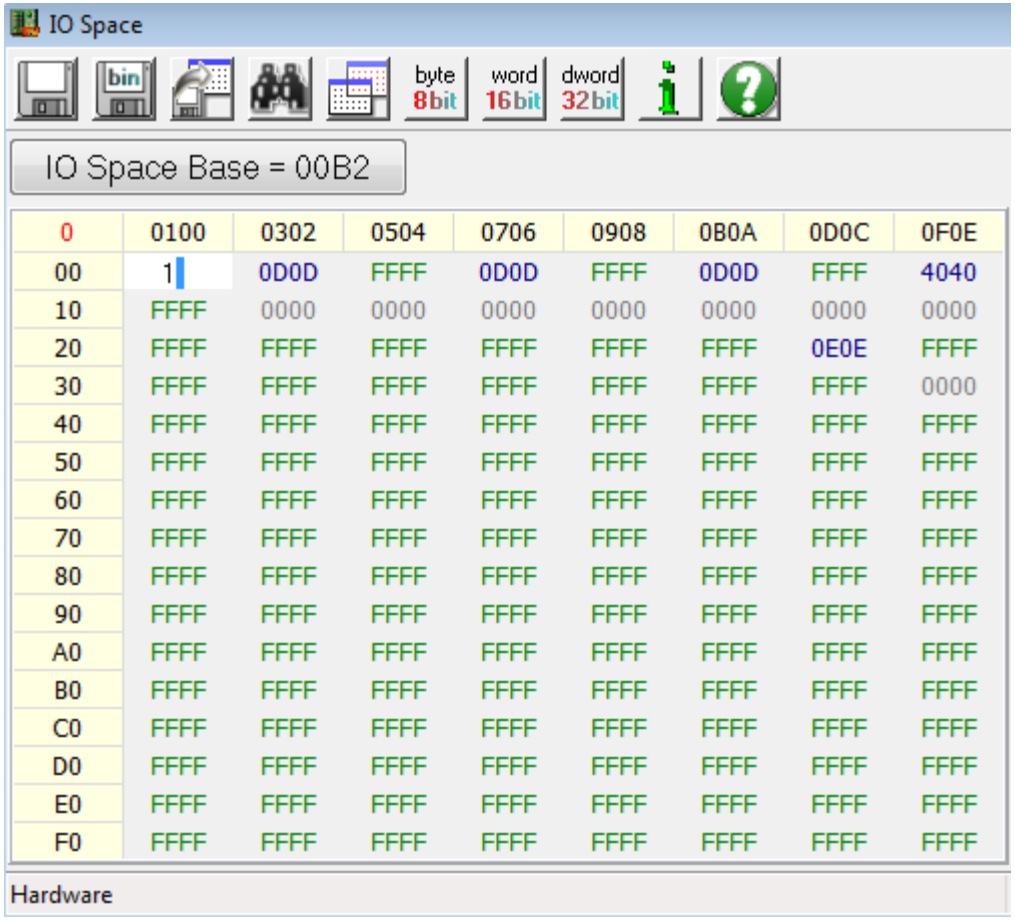
9.1.34 SWSMI—Software SMI

B/D/F/Type: 0/2/0/PCI
Address Offset: E0-E1h
Default Value: 0000h
Access: R/W
Size: 16 bits

Bit	Access	Default Value	RST/PWR	Description
15:8	R/W	00h	Core	Software Scratch Bits (SWSB):
7:1	R/W	00h	Core	**Software Flag (SWF):** This field is used to indicate caller and SMI function desired, as well as return result.
0	R/W	0b	Core	**GMCH Software SMI Event (GSSMIE):** When set, this bit will trigger an SMI. Software must write a 0 to clear this bit.

- Looking through the datasheets there are various (too many to show) register/bit-combinations that will also generate an SMI
- John called this a “corollary” SMI#. The SMI# is correlated with software is setting a register bit

SMI invocation example



0	0100	0302	0504	0706	0908	0B0A	0D0C	0F0E
00	1	0D0D	FFFF	0D0D	FFFF	0D0D	FFFF	4040
10	FFFF	0000	0000	0000	0000	0000	0000	0000
20	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	0E0E	FFFF
30	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	0000
40	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
50	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
60	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
70	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
80	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
90	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
A0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
B0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
C0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
D0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
E0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
F0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF

- Open a port IO window in RW-E at address 0xB2
- Type in the number 1 and hit enter
  - I choose 1 because I know it to be a ‘safe’ value to enter
- Notice anything?
  - (You’re not supposed to)
- The system just transitioned into SMM, executed code, and then exited SMM

SMI invocation counter

- So we've seen that there are a lot of events that can generate SMI and we've generated some on our own as well
- One logical question is: how frequently are these generated?
- As always, the answer is "it depends"
- On a system like a laptop, SMM will likely be called frequently to check the battery/power status
  - Should be an SMI# in that case
- But on a desktop it may be called much less frequently

SMI counter (MSR)

34H	52	MSR_SMI_COUNT	Core	**SMI Counter (R/O)**
		31:0		**SMI Count (R/O)** Running count of SMI events since last RESET.
		63:32		Reserved.

Nehalem (Core i series & Xeon) and later architectures only!

- And as if answering my wishes for some way to track how frequently SMM is entered, I found this in Intel's Software Programming Guide (Chapter 35, MSRs)
- It's only available on Nehalem and later processor families (new stuff)
- Trying to read this will crash any system that doesn't support it
  - I hope you read this before trying ;)

From Intel Vol. 3. Ch. "Model Specific Registers (MSRs)"

Periodic SMI#

GEN_PMCON_1—General PM Configuration 1 Register (PM—D31:F0)

Offset Address:	A0h	Attribute:	R/W, RO, R/WO
Default Value:	0000h	Size:	16-bit
Lockable:	No	Usage:	ACPI, Legacy
		Power Well:	Core

1:0	**Periodic SMI# Rate Select (PER_SMI_SEL)** — R/W. Set by software to control the rate at which periodic SMI# is generated. 00 = 64 seconds 01 = 32 seconds 10 = 16 seconds 11 = 8 seconds

- More as a side note (but related to the SMI counter), SMI# can be configured to fire periodically
- This way it can be guaranteed that SMI will be generated at least once every 8, 16, 32, or 64 seconds
- This register is R/W and resides on the LPC bus (D31:F0, offset A0h, bits 1:0)

13.8.3.8 SMI_EN—SMI Control and Enable Register

I/O Address:	PMBASE + 30h	Attribute:	R/W, R/WO, WO
Default Value:	00000002h	Size:	32 bit
Lockable:	No	Usage:	ACPI or Legacy
Power Well:	Core		

Note: This register is symmetrical to the SMI status register.

Bit	Description
31:28	Reserved
27	**GPIO_UNLOCK_SMI_EN**—R/WO. Setting this bit will cause the Intel® PCH to generate an SMI# when the GPIO_UNLOCK_SMI_STS bit is set in the SMI_STS register. Once written to 1, this bit can only be cleared by PLTRST#.
26:19	Reserved
18	**INTEL_USB2_EN**—R/W. 0 = Disable 1 = Enables Intel-Specific USB2 SMI logic to cause SMI#.
17	**LEGACY_USB2_EN**—R/W. 0 = Disable 1 = Enables legacy USB2 logic to cause SMI#.
16:15	Reserved
14	**PERIODIC_EN**—R/W. 0 = Disable. 1 = Enables the PCH to generate an SMI# when the PERIODIC_STS bit (PMBASE + 34h, bit 14) is set in the SMI_STS register (PMBASE + 34h).

From Intel Vol. 2

Reversing tip: searching for SMI communication byte patterns

OUT—Output to Port

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
E6 *ib*	OUT *imm8*, AL	I	Valid	Valid	Output byte in AL to I/O port address *imm8*.
E7 *ib*	OUT *imm8*, AX	I	Valid	Valid	Output word in AX to I/O port address *imm8*.
E7 *ib*	OUT *imm8*, EAX	I	Valid	Valid	Output doubleword in EAX to I/O port address *imm8*.
EE	OUT DX, AL	NP	Valid	Valid	Output byte in AL to I/O port address in DX.
EF	OUT DX, AX	NP	Valid	Valid	Output word in AX to I/O port address in DX.
EF	OUT DX, EAX	NP	Valid	Valid	Output doubleword in EAX to I/O port address in DX.

- You can search for instances where a program is communicating with SMM via port IO by searching for byte patterns like those above
- To locate triggering of SMI via port 0xB2, you can search for the bytes “E6 B2” and “E7 B2” in IDA Pro
  - Single byte searches for EE and EF yield many false-positives so analyze the code before it to ensure that the DX register contains B2
- You can also script IDA to create IDB files for all binaries in a folder, and then search within those binaries for “out 0B2h,”

SMI byte patterns example

```
66 E7 B2                        out     0B2h, ax        ; Advanced Power Management Control Port
                                                        ; generates SMI interrupt if APMC_EN bit is set
E6 B2                           out     0B2h, al        ; Advanced Power Management Control Port
                                                        ; generates SMI interrupt if APMC_EN bit is set
```

```
Asmi                    out   0B2h, al
                        out   0B2h, al
sub_3D1AEC              out   0B2h, al
sub_3D1B99              out   0B2h, al
SMIi                    out   0B2h, al
SNDI                    out   0B2h, al
SNDI                    out   0B2h, al
SNDI                    out   0B2h, al
sub_3D6E2C              out   0B2h, al
                        db 0E6h ; µ
                        out   0B2h, al
sub_3F0310              out   0B2h, al
                        db 0E6h ; µ
SMI_0xC4                out   0B2h, al
```

- Searching for instances of “E6 B2” and “E7 B2” in IDA will yield a list of examples like that on the left
- Most importantly, IDA provides an interesting clue here to a problem that we’ll be covering shortly…
- And is actually the only reason this slide is included

Entering SMM

- When receiving an SMI, the processor waits for all instructions to complete and stores to complete
- SMI interrupts are handled on an architecturally defined "interruptible" point in program execution
  - Like an instruction boundary:

- Processor saves the context in SMRAM and begins executing the SMI Handler
- In a multi-core processor, no SMI handler code is executed until all cores have performed the above and entered SMM

Entering SMM due to IO

Table 34-8. I/O Instruction Restart Field Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
00H	00H	Does not re-execute trapped I/O instruction.
00H	FFH	Re-executes trapped I/O instruction.

- IO Instruction restart field is located in the state save area in SMRAM
  - SMBASE + 8000 + 7F00h
- If an IO instruction to a device triggers an SMI, the SMI handler has the option of re-executing that instruction upon returning from SMM
  - Example: If a device is asleep, port IO to it may generate SMI#. The SMI handler can then wake up the device and re-execute the instruction that generated the original SMI#

Exiting SMM: RSM

RSM—Resume from System Management Mode

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F AA	RSM	NP	Invalid	Valid	Resume operation of interrupted program.

- The only way to exit SMM is through the RSM instruction
  - Or system reset/shut down
- Returns control to the application program or operating-system procedure that was interrupted by the SMI
- The processor's state is restored from the save state area within SMRAM. If the processor detects invalid state information during state restoration, it enters the shutdown state.
- The operating mode the processor was in at the time of the SMI is restored.
- RSM can only be executed from within SMM
  - So if you see this, you are debugging the SMRAM code
  - RSM multi-byte opcode is: 0x0F 0xAA
- Executing RSM while not in SMRAM generates an invalid opcode exception

“Performance Implications of System Management Mode”

- Here’s a good paper which pushes back against all the academic researchers who act like they can just implement all their security features in SMM
- http://web.cecs.pdx.edu/~karavan/research/SMM_IISWC_preprint.pdf

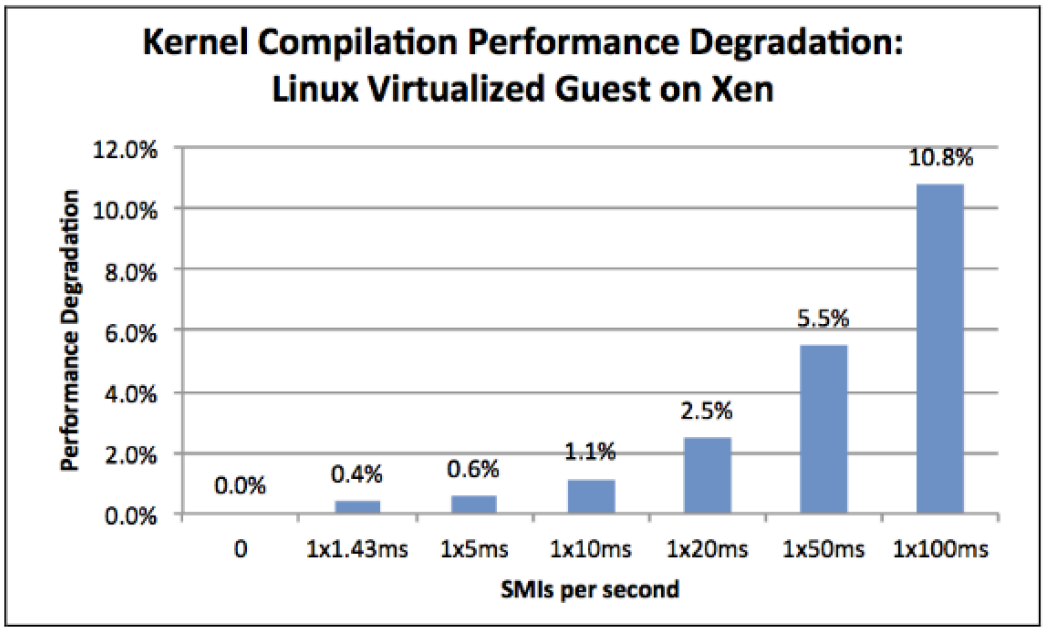


Figure 6: Kernel Compilation Performance for Linux/Xen

“Performance Implications of System Management Mode” 2

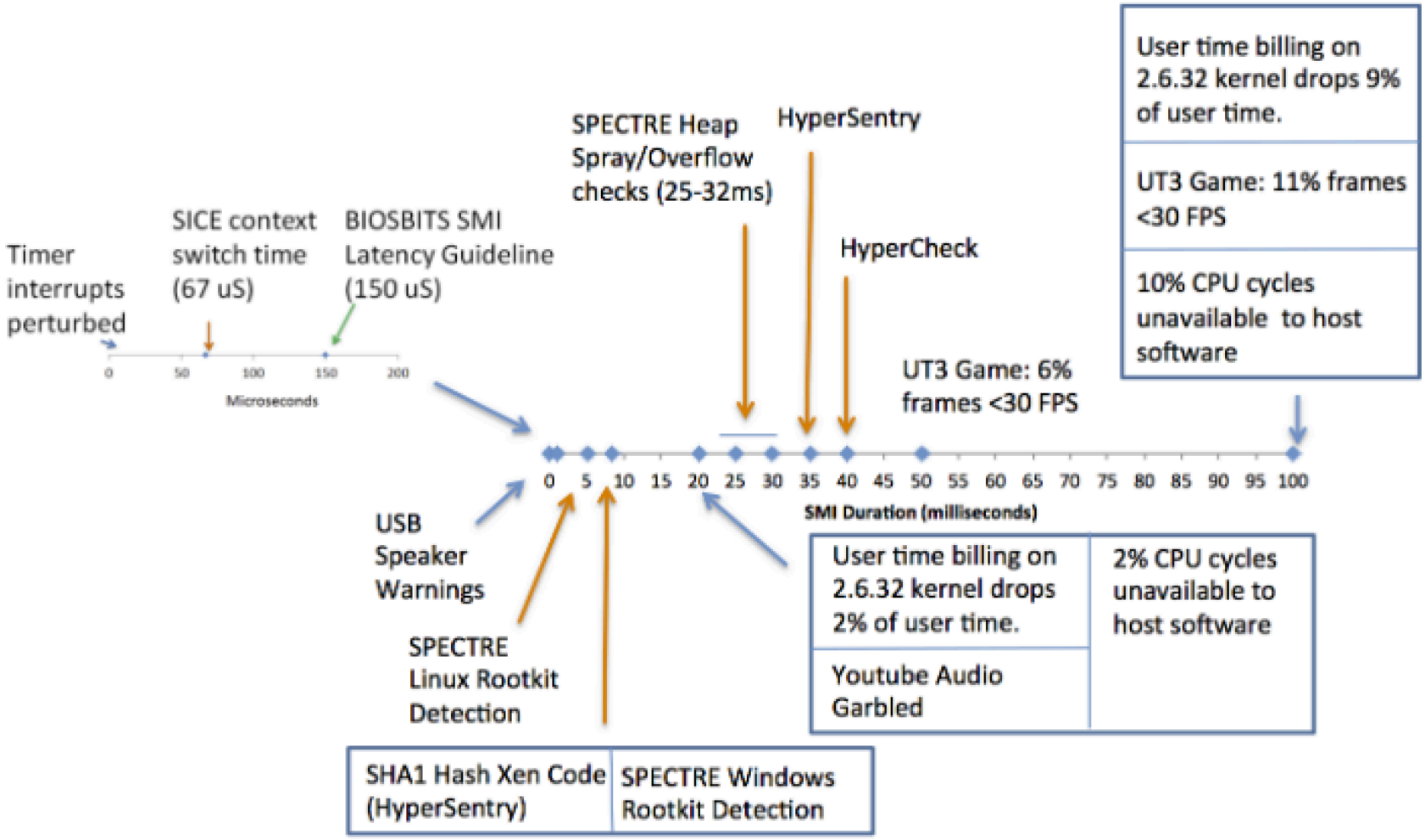


Figure 8: SMM Preemption Effects, one SMI per second