# Advanced x86:
# BIOS and System Management Mode Internals
## *SMRAM (System Management RAM)*

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”

# Prelude

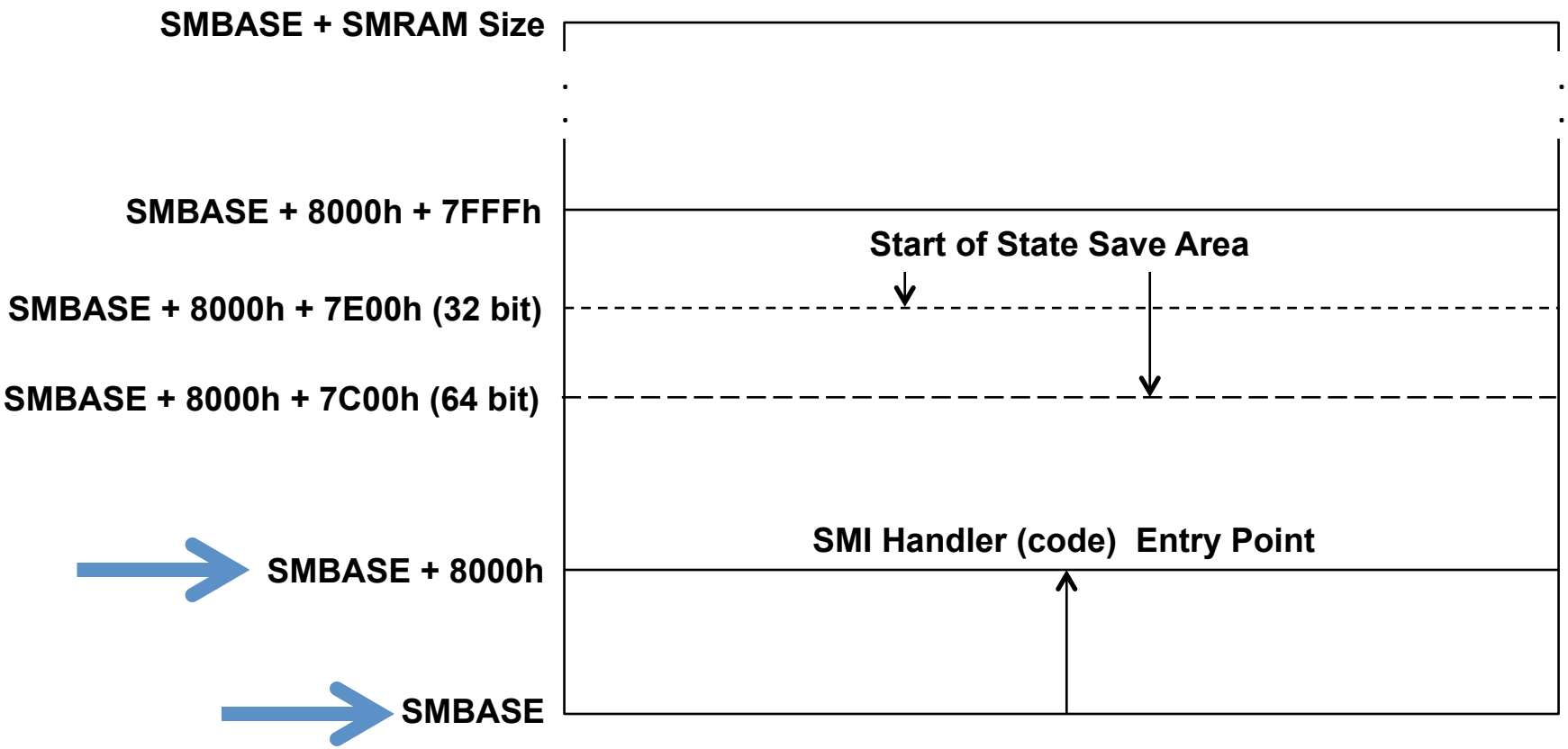
- So we have talked about what causes the system to enter SMM
- And we've (sort of) even "seen" it happen
  - As best as possible
- But we haven't talked about SMM's address space yet …

# SMRAM

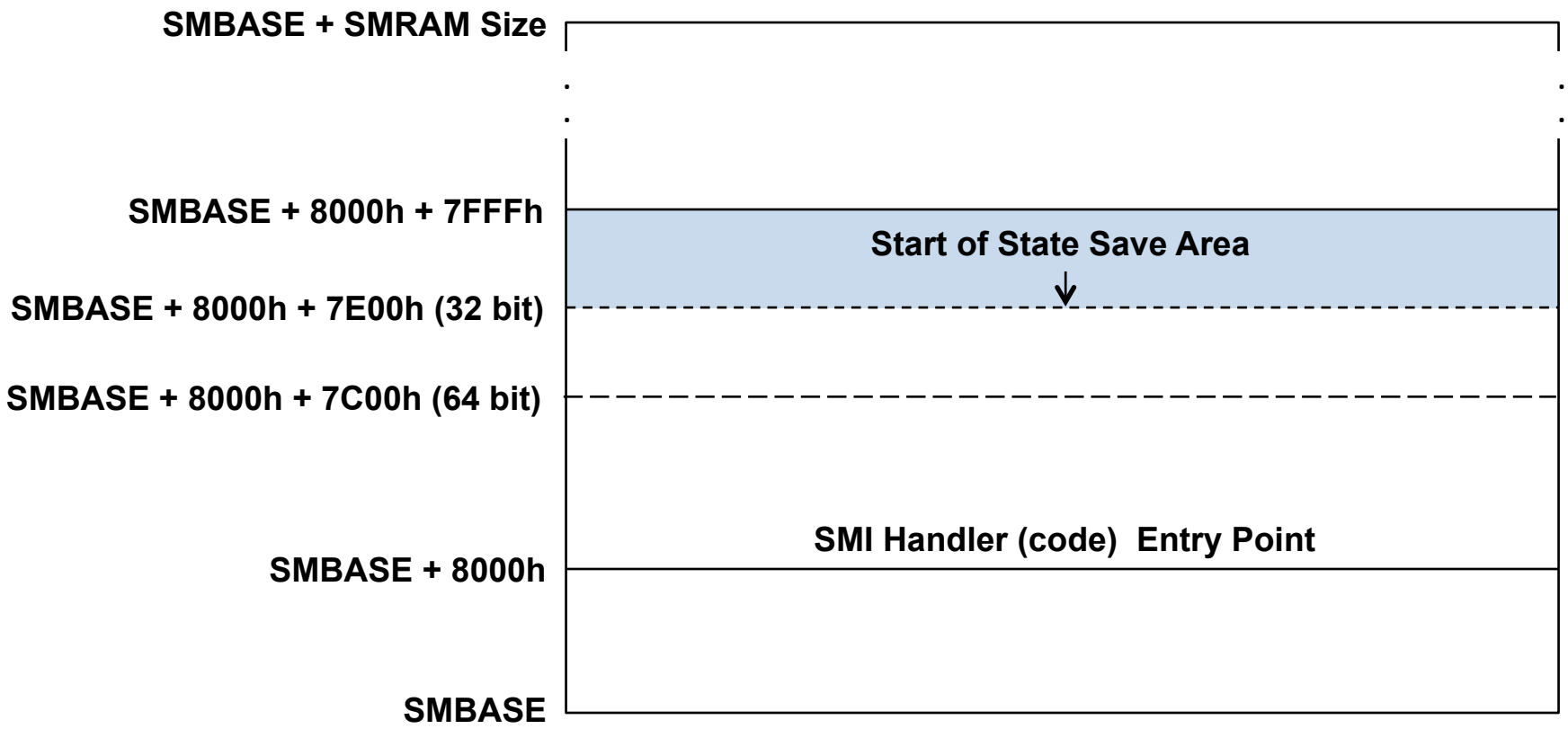
- SMRAM is the address space where the processor switches to upon entering SMM
- This address space contains the SMI handler code and data
- The processor's pre-SMI register context is saved at a pre-defined location in SMRAM (fixed offset from SMBASE)
- SMBASE is the base address of SMRAM and is located in a reserved portion of main RAM
  - Thus access control mechanisms must be based in the memory controller (MCH or CPU)

# Address Space Layout

- First off, lets define the terms SMBASE and SMRAM
- SMRAM refers to the entire range (or ranges) where the SMI handler code and data is located
- SMBASE is a private CPU-internal register that holds the address denoting the base address of SMRAM for a processor (or core)
  - Each core will have its own SMBASE
- The state save area(s) and entry point(s) are fixed offsets from SMBASE
- SMBASE is also found as a field stored in the state save area within SMRAM
  - The stored value is always at the same offset from SMBASE (FEF8h)
  - A 32-bit value containing the physical address of SMRAM (SMBASE)
  - Even in x64 architecture
- Therefore SMRAM is relocatable by changing the saved value of SMBASE, stored in the SMRAM save state area upon SMI
- We'll talk about where in physical memory SMBASE/SMRAM is likely to be located in a bit




- Default SMBASE on startup is 30000h, but can be relocated
- SMI Handler Executable code entry point is always at SMBASE + 8000h
  - CPU always begins executing at SMBASE + 8000h
- Multi-core systems will typically have their SMBASE offset by N bytes from each other. For example:
  - Core 0 defines SMBASE as A_0000h, will enter SMI handler at A_8000h
  - Core 1 defines SMBASE as A_1000h, will enter SMI handler at A_9000h



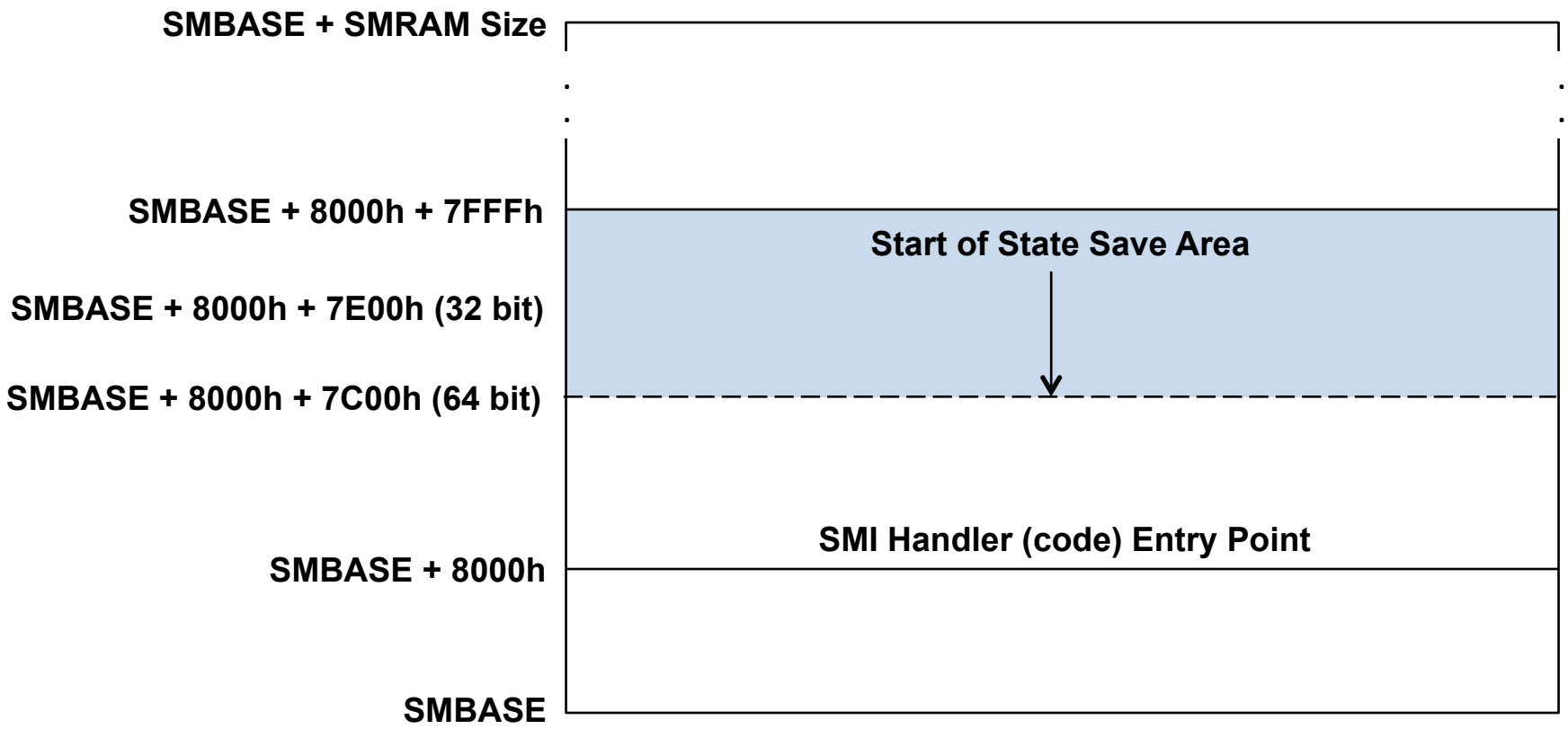

- State save address starts at SMBASE + 8000h + 7FFFh
  - SMBASE + FFFFh
- For 32-bit CPU's, the state-save area is 200h bytes
- State save area extends down to SMBASE + 8000h + 7E00h
  - SMBASE + FE00h

# 32-Bit

From Intel Vol. 3. Ch. "System Management Mode"

Table 34-1. SMRAM State Save Map

| Offset (Added to SMBASE + 8000H) | Register | Writable? |
|---|---|---|
| 7FFCH | CR0 | No |
| 7FF8H | CR3 | No |
| 7FF4H | EFLAGS | Yes |
| 7FF0H | EIP | Yes |
| 7FECH | EDI | Yes |
| 7FE8H | ESI | Yes |
| 7FE4H | EBP | Yes |
| 7FE0H | ESP | Yes |
| 7FDCH | EBX | Yes |
| 7FD8H | EDX | Yes |
| 7FD4H | ECX | Yes |
| 7FD0H | EAX | Yes |
| 7FCCH | DR6 | No |
| 7FC8H | DR7 | No |
| 7FC4H | TR[1] | No |
| 7FC0H | Reserved | No |
| 7FBCH | GS[1] | No |
| 7FB8H | FS[1] | No |
| 7FB4H | DS[1] | No |
| 7FB0H | SS[1] | No |
| 7FACH | CS[1] | No |
| 7FA8H | ES[1] | No |
| 7FA4H | I/O State Field, see Section 34.7 | No |
| 7FA0H | I/O Memory Address Field, see Section 34.7 | No |
| 7F9FH-7F03H | Reserved | No |
| 7F02H | Auto HALT Restart Field (Word) | Yes |
| 7F00H | I/O Instruction Restart Field (Word) | Yes |
| 7EFCH | SMM Revision Identifier Field (Doubleword) | No |
| 7EF8H | SMBASE Field (Doubleword) | Yes |
| 7EF7H - 7E00H | Reserved | No |




- State save address starts at SMBASE + 8000h + 7FFFh
  - SMBASE + FFFFh
- For 64-bit CPU's, the state-save area is 400h bytes
- The state-save extends down to SMBASE + 8000h + 7C00h
  - SMBASE + FC00h

64-Bit

From Intel Vol. 3. Ch. "System Management Mode"

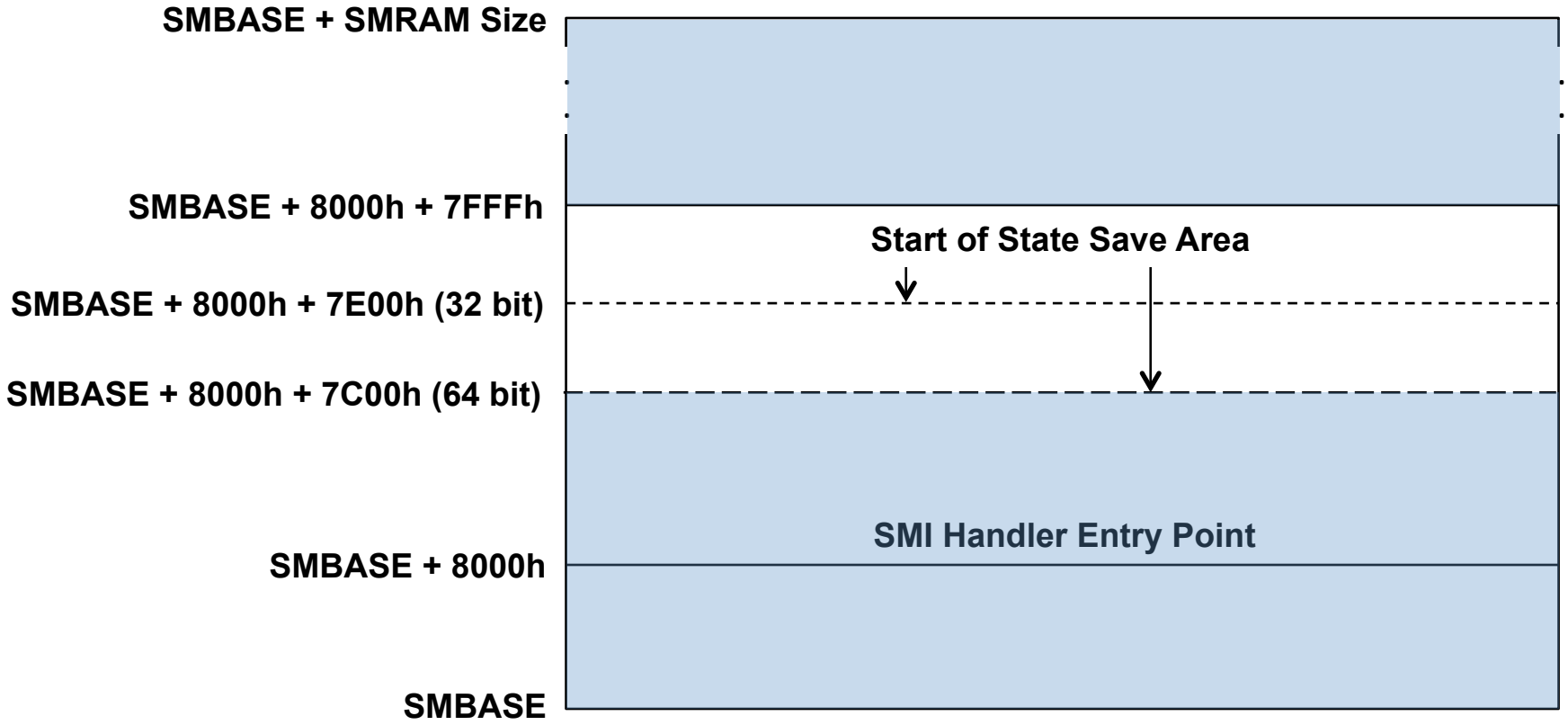
Table 34-3. SMRAM State Save Map for Intel 64 Architecture

| Offset (Added to SMBASE + 8000H) | Register | Writable? |
|---|---|---|
| 7FF8H | CR0 | No |
| 7FF0H | CR3 | No |
| 7FE8H | RFLAGS | Yes |
| 7FE0H | IA32_EFER | Yes |
| 7FD8H | RIP | Yes |
| 7FD0H | DR6 | No |
| 7FC8H | DR7 | No |
| 7FC4H | TR SEL[1] | No |
| 7FC0H | LDTR SEL[1] | No |
| 7FBCH | GS SEL[1] | No |
| 7FB8H | FS SEL[1] | No |
| 7FB4H | DS SEL[1] | No |
| 7FB0H | SS SEL[1] | No |
| 7FACH | CS SEL[1] | No |
| 7FA8H | ES SEL[1] | No |
| 7FA4H | IO_MISC | No |
| 7F9CH | IO_MEM_ADDR | No |
| 7F94H | RDI | Yes |
| 7F8CH | RSI | Yes |
| 7F84H | RBP | Yes |
| 7F7CH | RSP | Yes |
| 7F74H | RBX | Yes |
| 7F6CH | RDX | Yes |
| 7F64H | RCX | Yes |
| 7F5CH | RAX | Yes |
| 7F54H | R8 | Yes |
| 7F4CH | R9 | Yes |
| 7F44H | R10 | Yes |
| 7F3CH | R11 | Yes |
| 7F34H | R12 | Yes |

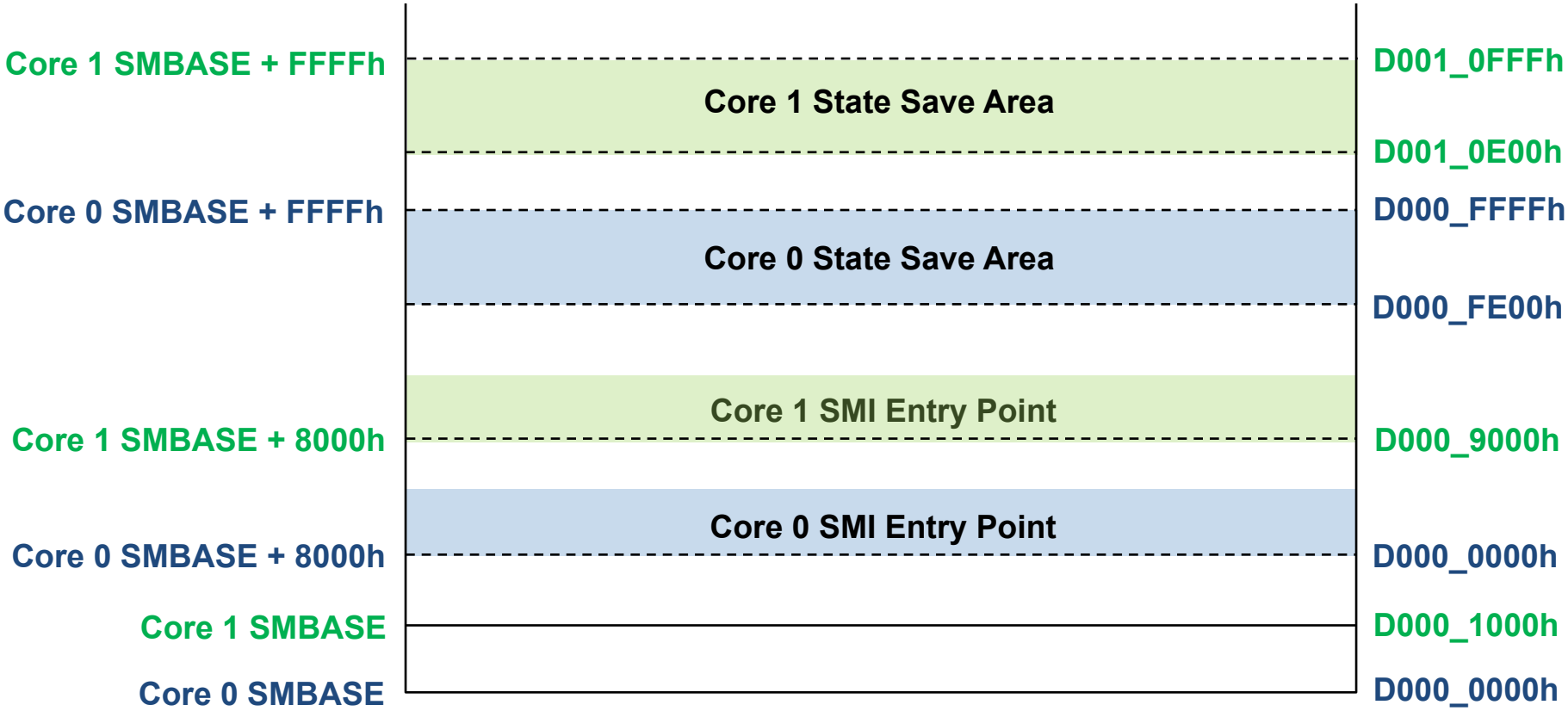
64-Bit

| | | |
|---|---|---|
| 7F34H | R12 | Yes |
| 7F2CH | R13 | Yes |
| 7F24H | R14 | Yes |
| 7F1CH | R15 | Yes |
| 7F1BH-7F04H | Reserved | No |
| 7F02H | Auto HALT Restart Field (Word) | Yes |
| 7F00H | I/O Instruction Restart Field (Word) | Yes |
| 7EFCH | SMM Revision Identifier Field (Doubleword) | No |
| 7EF8H | SMBASE Field (Doubleword) | Yes |
| **Offset (Added to SMBASE + 8000H)** | **Register** | **Writable?** |
| 7EF7H - 7EE4H | Reserved | No |
| 7EE0H | Setting of "enable EPT" VM-execution control | No |
| 7ED8H | Value of EPTP VM-execution control field | No |
| 7ED7H - 7EA0H | Reserved | No |
| 7E9CH | LDT Base (lower 32 bits) | No |
| 7E98H | Reserved | No |
| 7E94H | IDT Base (lower 32 bits) | No |
| 7E90H | Reserved | No |
| 7E8CH | GDT Base (lower 32 bits) | No |
| 7E8BH - 7E44H | Reserved | No |
| 7E40H | CR4 | No |
| 7E3FH - 7DF0H | Reserved | No |
| 7DE8H | IO_RIP | Yes |
| 7DE7H - 7DDCH | Reserved | No |
| 7DD8H | IDT Base (Upper 32 bits) | No |
| 7DD4H | LDT Base (Upper 32 bits) | No |
| 7DD0H | GDT Base (Upper 32 bits) | No |
| 7DCFH - 7C00H | Reserved | No |

- SMBASE field for both 64-bit and 32-bit architectures is always located at the same offset from SMBASE (FEF8h)




- The remaining area is free for use as SMI handler code and data
- Total size of SMRAM region is defined by the BIOS when it configures SMM




- Each core will have its own SMBASE address offset from the other core(s) SMBASE addresses
  - Like 1000h bytes per the above 32-bit example
- Another core could define its SMBASE in a completely separate memory address
  - In this diagram I show them sharing the same SMRAM memory range
  - In practice, some cores will simply execute a dead loop

# SMRAM Location

- SMRAM can be located anywhere in the 4GB memory address space
- SMBASE can be overwritten by the SMI handler
- Typically SMRAM is relocated at least once:
  - On system startup, the first time the system enters SMM, SMBASE is at 0x30000
  - SMI handler starts executing at 0x38000
  - There is no reason it needs to stay at that address
- Intel defines a few locations for SMRAM
  - But it really is a flexible system and can be put anywhere
  - I think these guidelines are provided to make configuration easier for the BIOS developers and to avoid areas where SMRAM may overlap with other regions
  - This is all part of building that memory map

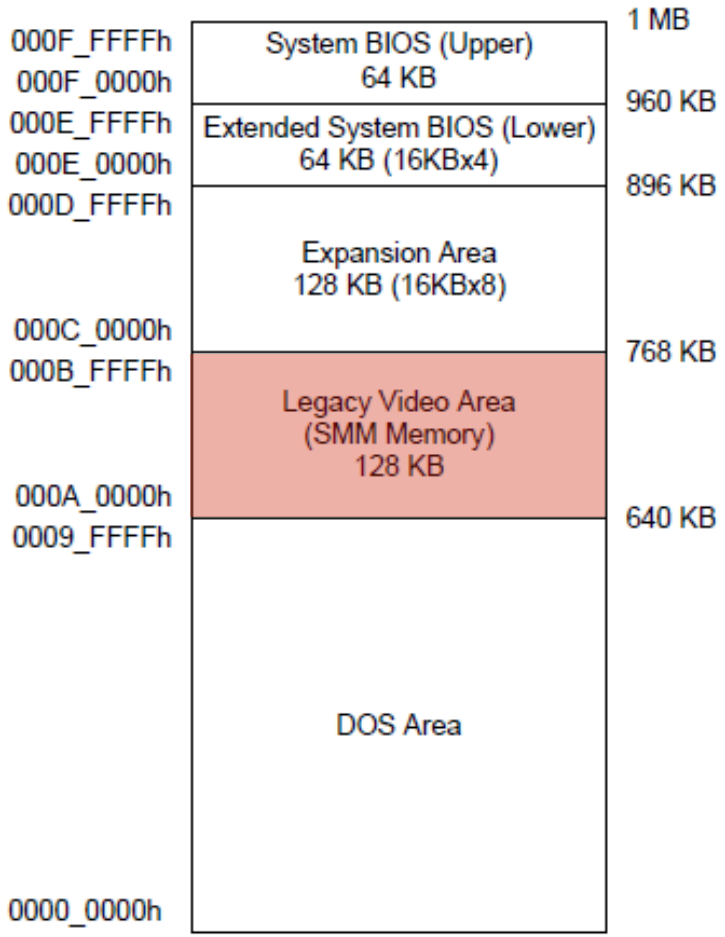
# Standard SMRAM Locations

**SMM Space Definition Summary**

| SMM Space Enabled | Transaction Address Space | DRAM Space (DRAM) |
|---|---|---|
| Compatible (Adr C) | 000A_0000h to 000B_FFFFh | 000A_0000h to 000B_FFFFh |
| High (Adr H) | FEDA_0000h to FEDB_FFFFh | 000A_0000h to 000B_FFFFh |
| TSEG (Adr T) | (TOLUD minus STOLEN minus TSEG) to (TOLUD minus STOLEN) | (TOLUD minus STOLEN minus TSEG) to (TOLUD minus STOLEN) |

- On ICH/MCH chipsets there are 3 standard locations for SMRAM
- On PCH chipsets the High Address (HSEG) is no longer supported (so 2 locations)
- Technically the base address of SMRAM can be relocated by the SMI handler
  - But there are reasons these guidelines should be followed and for all practical purposes SMRAM will be in TSEG

# Compatible SMRAM (Legacy Video Area)




Legacy (DOS) Compatibility Range

- Fixed address space
- Legacy VGA space (A_0000 - B_FFFFh)

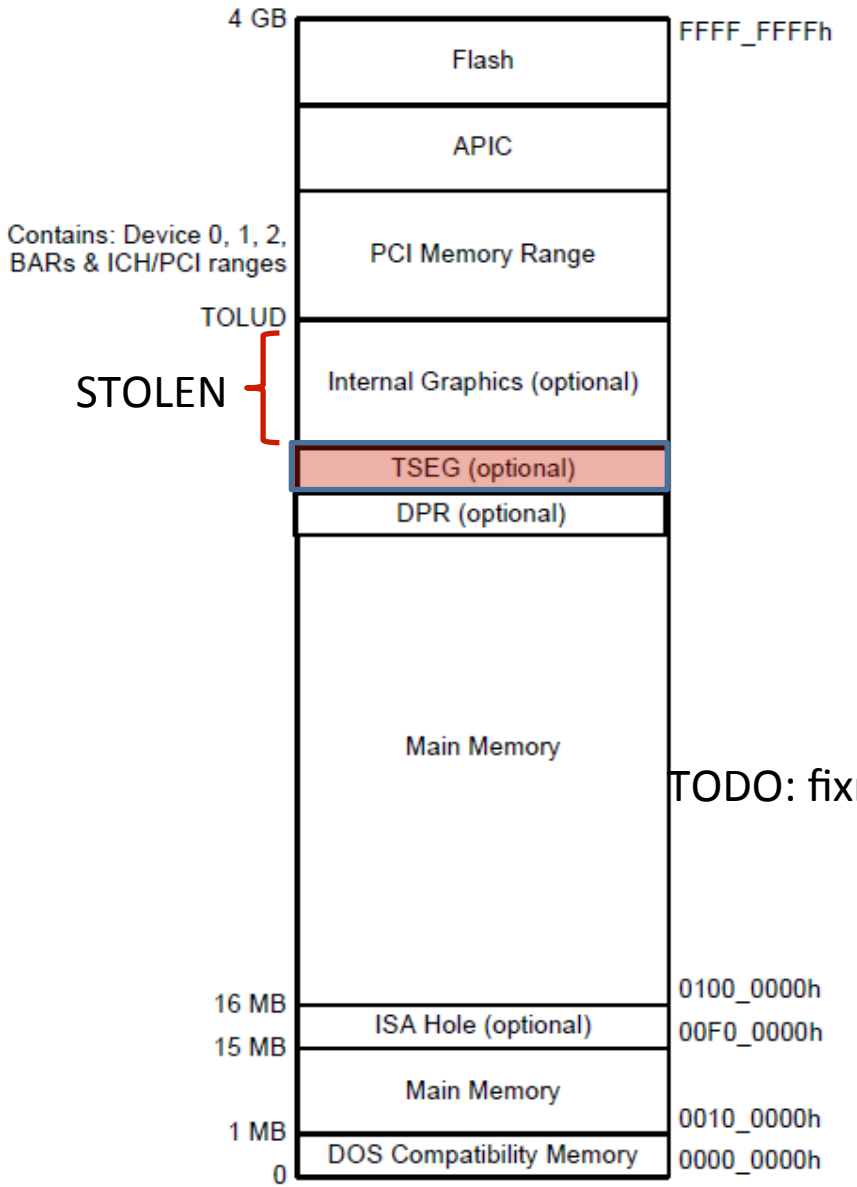
- When compatible SMM space is enabled, SMM-mode processor accesses to this range are routed to physical system memory at this address.
- Non-SMM-mode processor accesses to this range are considered to be to the video buffer area.

# Enabling Compatible SMRAM

| | | | |
|---|---|---|---|
| 3 | R/W/L | 0b | **Global SMRAM Enable (G_SMRARE):** If set to a 1, then Compatible SMRAM functions are enabled, providing 128 KB of DRAM accessible at the A0000h address while in SMM (ADSB with SMM decode). To enable Extended SMRAM function this bit has be set to 1. Refer to the section on SMM for more details.<br>This register is locked in Intel TXT mode (RO in Intel TXT mode). It also locks when D_LCK bit is set. |

- This address space is enabled by asserting the G_SMRAME bit
  - I believe G_SMRARE is a typo in the datasheet
- The register (SMRAMC) containing this bit will be located in a different place depending on the architecture
  - On our E6400 it is located in the DRAM Controller (D0:F0) at offset 9Dh
  - On a Haswell system, for example, it is also located in the DRAM controller but at offset 88h

# TSEG (Top of Main Memory Segment)




- Variable address space
  - In terms of size and location
- Located at:
  TOLUD – STOLEN – TSEG_SZ
  to TOLUD – STOLEN
- STOLEN refers to Graphics Memory Stolen, which is optional and can be zero
- New architecture enables more customized location/size of TSEG

* Xeno thinks this is provided by SMRR now

When extended SMRAM space is enabled, processor accesses to the TSEG range when the processor is not in SMM are treated as invalid

Non-processor originated accesses are not allowed to TSEG range.

# Enabling TSEG

**ESMRAMC - Extended System Management RAM Control**

| | |
|---|---|
| B/D/F/Type: | 0/0/0/PCI |
| Address Offset: | 9Eh |
| Default Value: | 38h |
| Access: | R/W/L; R/WC; RO |
| Size: | 8 bits |

| | | | |
|---|---|---|---|
| 0 | R/W/L | 0b | **TSEG Enable (T_EN):** Enabling of SMRAM memory for Extended SMRAM space only. When G_SMRAME =1 and TSEG_EN = 1, the TSEG is enabled to appear in the appropriate physical address space.<br>This register is locked in Intel TXT mode (RO in Intel TXT mode). It also locks when D_LCK bit is set. |

- TSEG is enabled differently depending on the architecture
- On MCH chipsets, it was defined in the ESMRAMC register
- Either 1, 2, or 8MB in size for TSEG (defined in bits 2:1)
- On our E6400 it's in D0:F0, offset 9Eh
- Newer systems offer more flexibility in TSEG size and location

# Enabling TSEG on new platforms

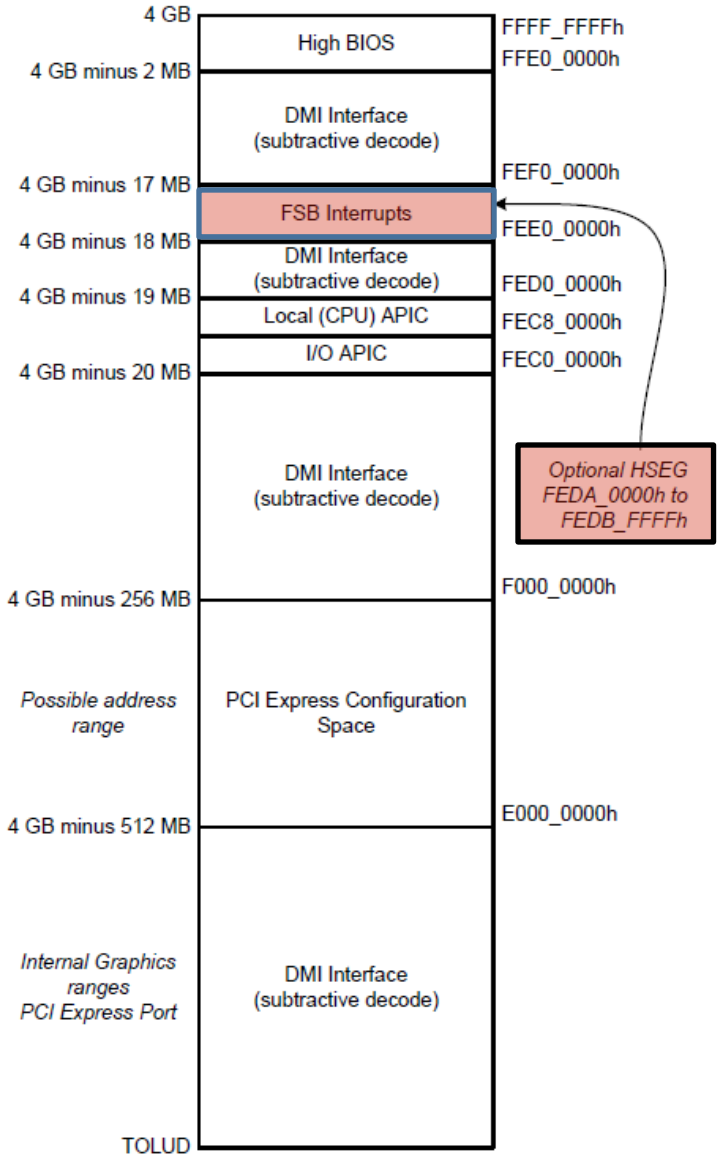
### 3.1.36 TSEGMB—TSEG Memory Base

This register contains the base address of TSEG DRAM memory. BIOS determines the base of TSEG memory which must be at or below Graphics Base of GTT Stolen Memory (PCI Device 0 Offset B4 bits 31:20). NOTE: BIOS must program TSEGMB to a 8MB naturally aligned boundary.

| B/D/F/Type: 0/0/0/CFG | | | Access: RW_KL; RW_L | |
|---|---|---|---|---|
| Size: 32 | Default Value: 00000000h | | Address Offset: B8h | |
| **Bit Range** | **Acronym** | **Description** | **Default** | **Access** |
| 31:20 | TSEGMB | This register contains the base address of TSEG DRAM memory. BIOS determines the base of TSEG memory which must be at or below Graphics Base of GTT Stolen Memory (PCI Device 0 Offset B4 bits 31:20). BIOS must program the value of TSEGMB to be the same as BGSM when TSEG is disabled. | 000h | RW_L |
| 19:1 | RSVD | Reserved. | 00000h | RO |
| 0 | LOCK | This bit will lock all writeable settings in this register, including itself. | 0h | RW_KL |

- On newer systems the size of TSEG is more flexible in its programming
- The offset of this register and its method of programming is dependent on the memory controller (which exists either in the MCH or the processor)

# HSEG (High SMM Memory Space)




- Fixed address space
- FEDA_0000 to FEDB_FFFFh
- When enabled (if supported), A_0000h to B_FFFFh are remapped to high memory
- Not supported in PCH chipsets
- Note: TSEG is located under TOLUD which is located at the bottom of this diagram

# Enabling HSEG

## ESMRAMC - Extended System Management RAM Control

| | |
|---|---|
| B/D/F/Type: | 0/0/0/PCI |
| Address Offset: | 9Eh |
| Default Value: | 38h |
| Access: | R/W/L; R/WC; RO |
| Size: | 8 bits |

| | | | |
|---|---|---|---|
| 7 | R/W/L | 0b | **Enable High SMRAM (H_SMRAME):** Controls the SMM memory space location (i.e., above 1 MB or below 1 MB) When G_SMRAME is 1 and H_SMRAME this bit is set to 1, the high SMRAM memory space is enabled. SMRAM accesses within the range 0FEDA0000h to 0FEDBFFFFh are remapped to DRAM addresses within the range 000A0000h to 000BFFFFh.<br>This register is locked in Intel® TXT mode (RO in Intel TXT mode). It also locks when D_LCK bit is set. |

- HSEG is enabled when bit 7 of the ESMRAMC bit is set
- Not supported on PCH systems and later

# SMRAM Combinations (MCH-based)

**SMM Space Table**

| Global Enable G_SMRAME | High Enable H_SMRAM_EN | TSEG Enable TSEG_EN | Adr C Range | Adr H Range | Adr T Range |
|---|---|---|---|---|---|
| 0 | X | X | Disable | Disable | Disable |
| 1 | 0 | 0 | Enable | Disable | Disable |
| 1 | 0 | 1 | Enable | Disable | Enable |
| 1 | 1 | 0 | Disabled | Enable | Disable |
| 1 | 1 | 1 | Disabled | Enable | Enable |

- Up to two memory locations can be used for SMRAM on a system
- There is still only one SMBASE per core
- Global Enable means that SMM compatible space is turned on
- Disabling the C-range disables all other ranges
- So if you're using TSEG, there is guaranteed to be either the C-range or the H-range also present
  - I removed H-range discussion from this class for time reasons. It's pretty straightforward, and you can see the manuals if you're interested. But you probably mostly all have PCH-based systems so…

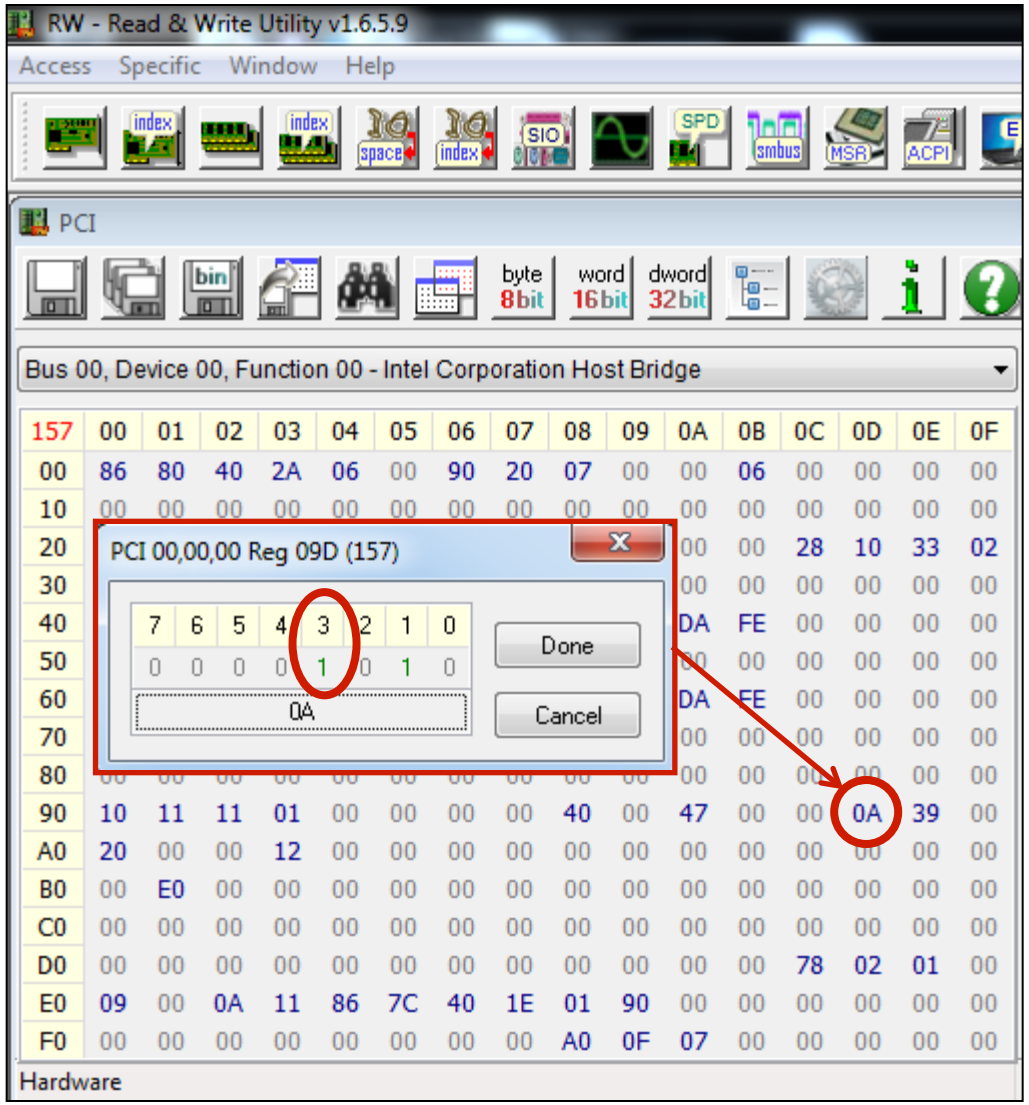
# SMRAM Combinations (PCH-based)

**SMM Space Table**

| Global Enable G_SMRAME | High Enable H_SMRAM_EN | TSEG Enable TSEG_EN | Adr C Range | Adr H Range | Adr T Range |
|---|---|---|---|---|---|
| 0 | X | X | Disable | Disable | Disable |
| 1 | 0 | 0 | Enable | Disable | Disable |
| 1 | 0 | 1 | Enable | Disable | Enable |
| 1 | 1 | 0 | Disabled | Enable | Disable |
| 1 | 1 | 1 | Disabled | Enable | Enable |

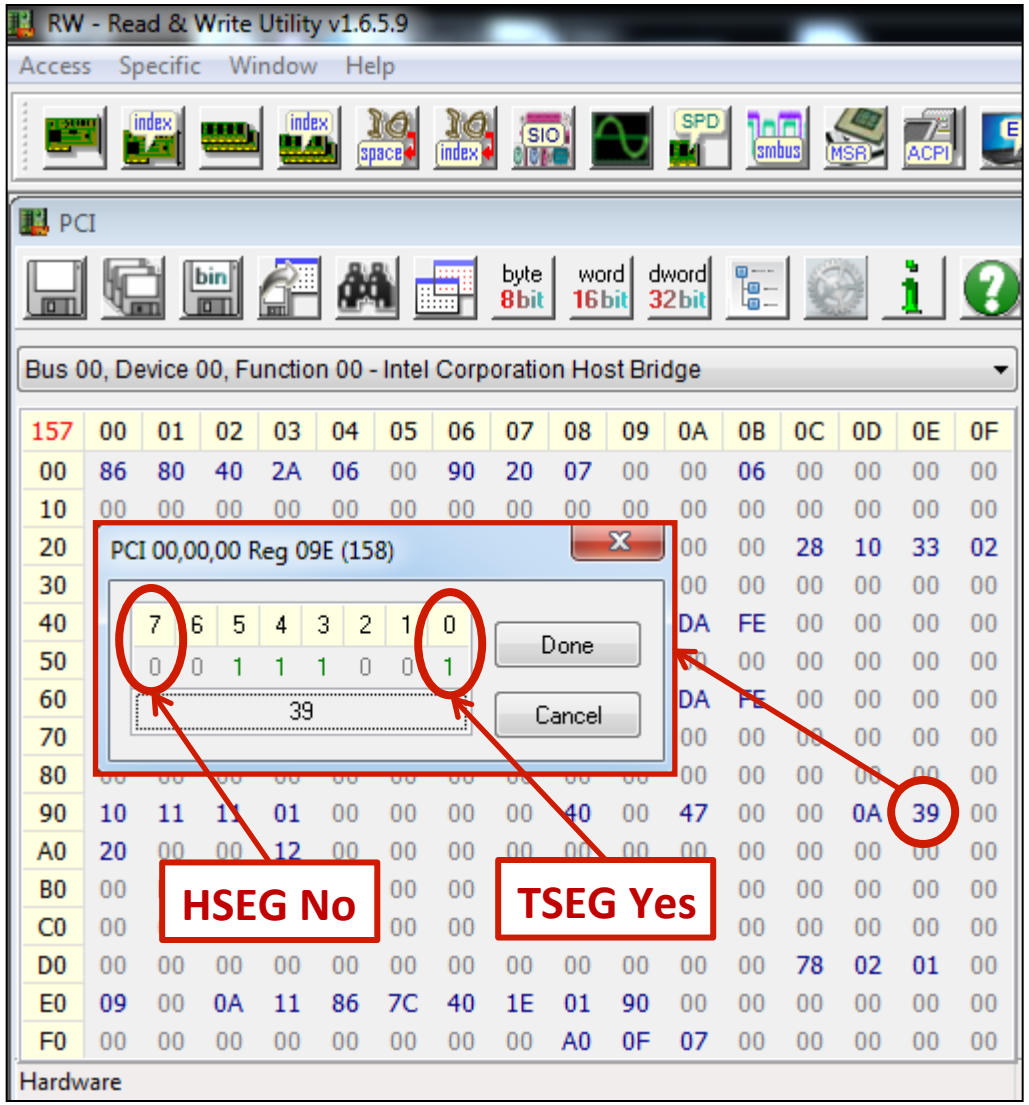
- Up to two memory locations can be used for SMRAM on a system
  - Which is good since on PCH there is only the Compatible and TSEG ranges; HSEG is no longer supported
- As you can see, this means that the Compatible range is always enabled if TSEG is enabled
- So the only question is whether or not you use TSEG

# Demo: Locating SMRAM



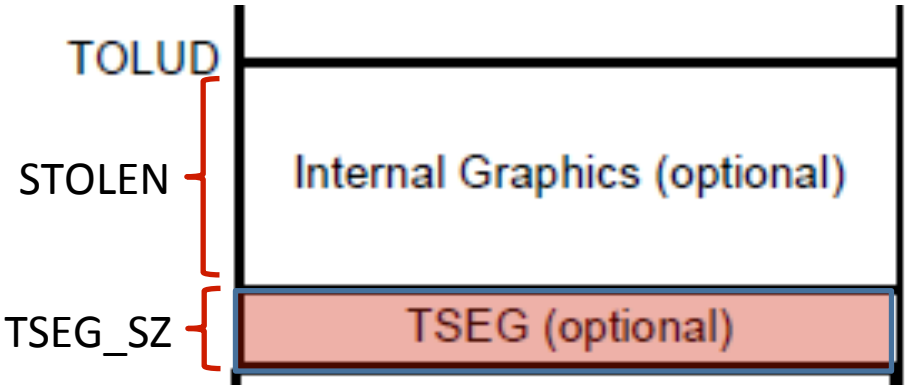

- First let's see what address ranges are enabled on our system for SMRAM
- Open RW-Everything and select PCI devices, device 0, function 0 (the DRAM Controller)
- Look at offset 9Dh (SMRAMC register)
- See if bit 3 (G_SMRAME) bit is set
- Compatible SMRAM at A_0000 to B_FFFFh is set

# Demo: Locating SMRAM




- Let's now check to see if TSEG (or HSEG) is enabled
- We know we cannot be using both along side the compatible C range
- Look at the register at offset 9Eh (ESMRAMC)
- Notice that TSEG Enable bit 0 is asserted
  - By default then, HSEG is not enabled, but we can also see that bit 7 is not asserted

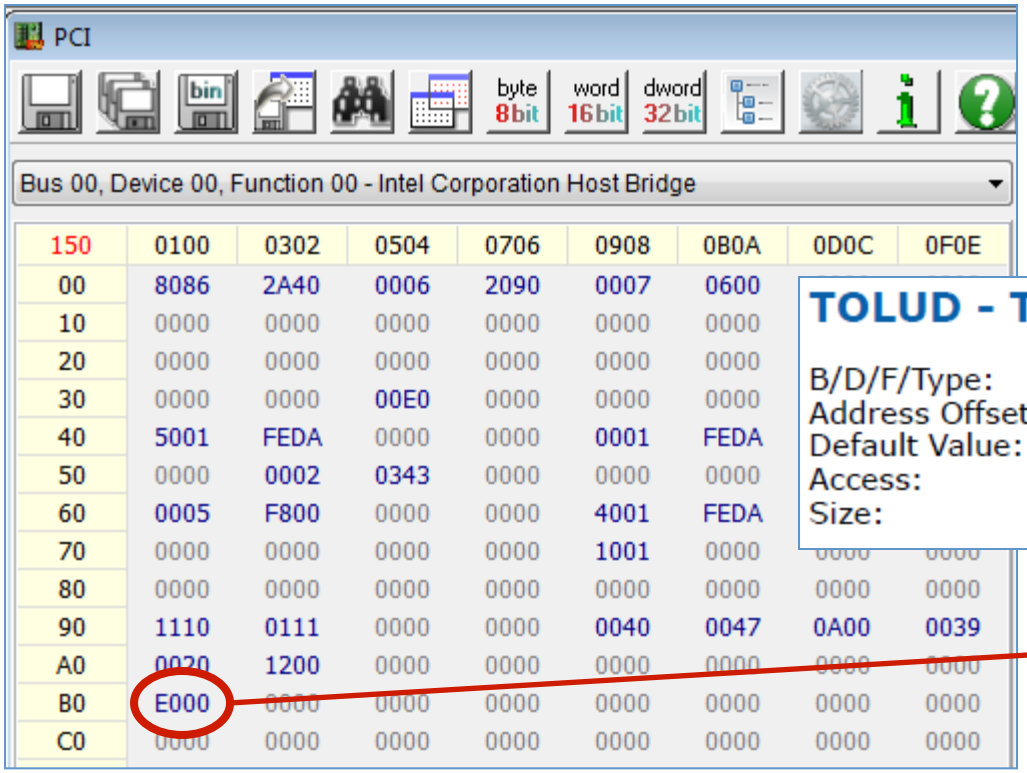
# Example: Find TSEG Base/Limit




- So we know that Compatible SMRAM is enabled, and that is always at a fixed address A_0000 to B_FFFFh
- But TSEG is dependent on other addresses
- The TSEG location can be calculated (if you are analyzing a system that does not have an explicit TSEG register):
- From the manual, TSEG Range *for this machine* is calculated as:

**(TOLUD – STOLEN – TSEG_SZ) to (TOLUD – STOLEN)**

# Example: Find TSEG: TOLUD




| 150 | 0100 | 0302 | 0504 | 0706 | 0908 | 0B0A | 0D0C | 0F0E |
|---|---|---|---|---|---|---|---|---|
| 00 | 8086 | 2A40 | 0006 | 2090 | 0007 | 0600 | | |
| 10 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | | |
| 20 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | | |
| 30 | 0000 | 0000 | 00E0 | 0000 | 0000 | 0000 | | |
| 40 | 5001 | FEDA | 0000 | 0000 | 0001 | FEDA | | |
| 50 | 0000 | 0002 | 0343 | 0000 | 0000 | 0000 | | |
| 60 | 0005 | F800 | 0000 | 0000 | 4001 | FEDA | | |
| 70 | 0000 | 0000 | 0000 | 0000 | 1001 | 0000 | 0000 | 0000 |
| 80 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 90 | 1110 | 0111 | 0000 | 0000 | 0040 | 0047 | 0A00 | 0039 |
| A0 | 0020 | 1200 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| B0 | E000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| C0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

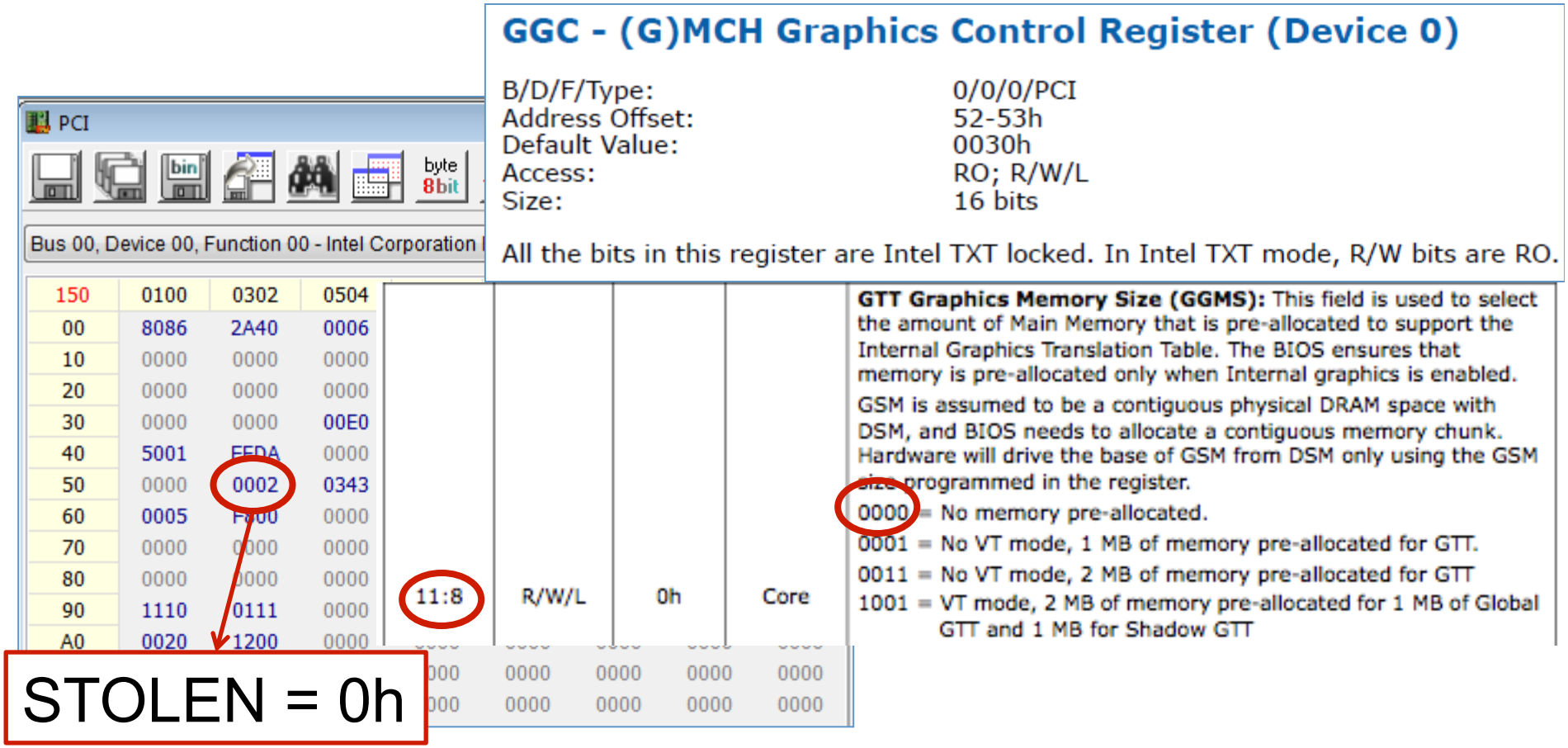
**TOLUD - Top of Low Used DRAM Register**

| | |
|---|---|
| B/D/F/Type: | 0/0/0/PCI |
| Address Offset: | B0-B1h |
| Default Value: | 0010h |
| Access: | R/W/L; RO |
| Size: | 16 bits |

TOLUD = E000_0000h

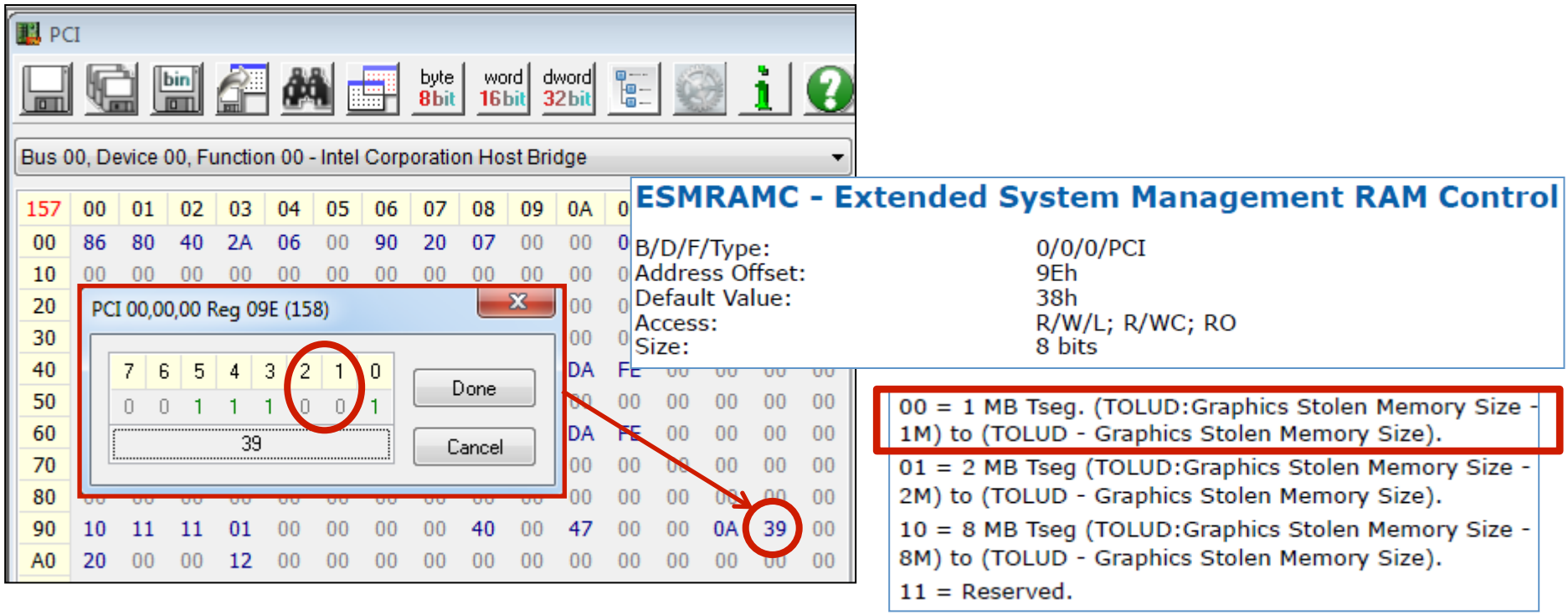
- (TOLUD – STOLEN – TSEG_SZ) to (TOLUD – STOLEN)
- (E0000000 - ? - ?) to (E0000000 - ?)

# Lab: Find TSEG: STOLEN



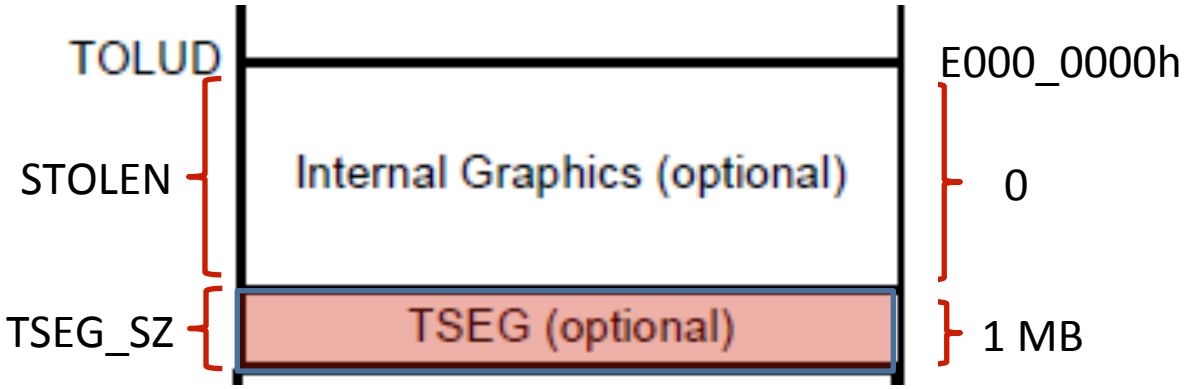

- Next is to find the amount of memory (if any) that has been stolen from graphics
- Bits 11:8 determine the amount of graphics memory stolen
- In this case it is 0
- (TOLUD – STOLEN – TSEG_SZ) to (TOLUD – STOLEN)
- (E0000000 - 0 - ?) to (E0000000 - 0)

# Lab: Find TSEG: TSEG_SZ



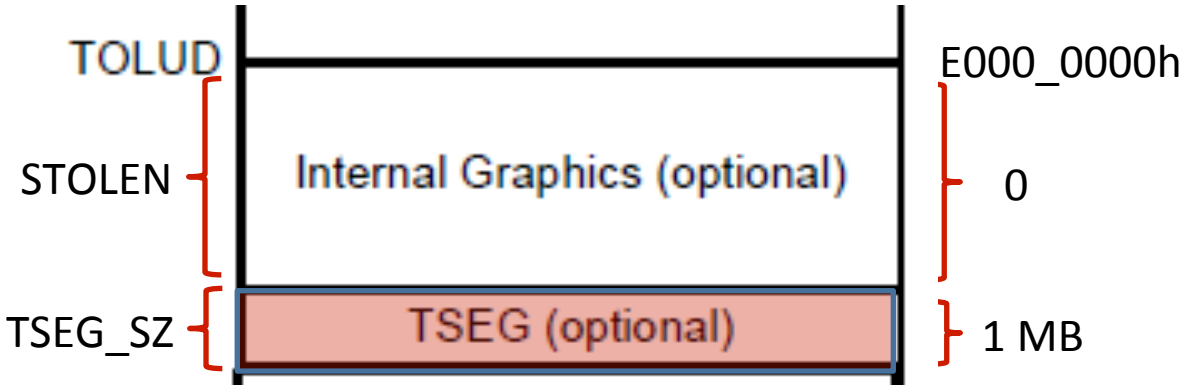

- And lastly we need to determine the size of TSEG
- This will differ based on architecture but for our system it's 8 bits
- (TOLUD – STOLEN – TSEG_SZ) to (TOLUD – STOLEN)
- (E000_0000 - 0 - 10_0000) to (E0000000 - 0)

# Calculate TSEG Base/Limit




- We plug our known values into:
- (TOLUD – STOLEN – TSEG_SZ) to (TOLUD – STOLEN)
- (E000_0000h – 0 – 1 MB) to (E000_0000 – 0)
- Provides us the range:
- DFF0_0000h to E000_0000h
  - Technically it's DFF0_0000 to DFFF_FFFFh
- This *should* be the SMBASE address (which is relocatable of course but in all likelihood will be here)
- You can also read the SMRR PHYSBASE MSR if it's supported or on a newer system read the TSEG Base/Limit from the TSEG register
  - SMRRs covered in a little bit

# Calculate TSEG Base/Limit




- The TSEG base address marks the beginning of the protected SMRAM range
- Therefore the TSEG base *should* equate to SMBASE
  - Or the lowest SMBASE value in a multi-core system, assuming shared SMRAM range
- We'll see in a bit that this isn't necessarily the case

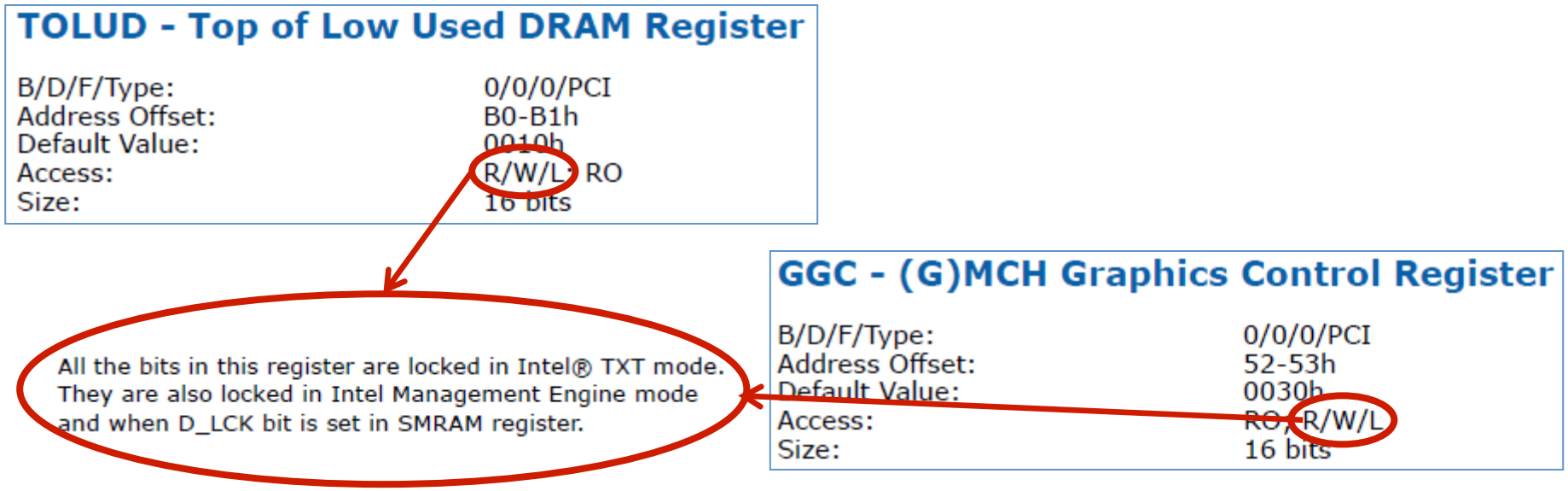
# TSEG STOLEN varies

- For the MCH 4, the TSEG range is defined as (TOLUD – STOLEN – TSEG_SZ) to (TOLUD – STOLEN)
- But different systems will have different values, and you have to look it up in the datasheets. E.g. on a 4th gen Haswell:
- (TOLUD – DSM SIZE – GSM SIZE – TSEG SIZE) to (TOLUD – DSM SIZE – GSM SIZE)

# Homework heads up

- Determine if your system's TSEG/TOLUD are locked, or if they could be moved by an attacker
- On some systems they will be locked by D_LCK, and on some TSEGMB will have its own lock bit. You need to determine which is the case for your hardware.

# Memory Map Protection




- The location of TSEG is dependent on the values of TOLUD and Stolen Memory
- Modifying this value is something that an attacker could try, to shift the TSEG region
- However these registers can be locked down by D_LCK bit in the SMRAM register (a key-bit)

# SMRAM Lock-Down

**SMRAM - System Management RAM Control**

| | |
|---|---|
| B/D/F/Type: | 0/0/0/PCI |
| Address Offset: | 9Dh |
| Default Value: | 02h |
| Access: | RO; R/W/L; R/W |
| Size: | 8 bits |

The SMRAMC register controls how accesses to Compatible and Extended SMRAM spaces are treated. The Open, Close, and Lock bits function only when G_SMRAME bit is set to a 1. Also, the OPEN bit must be reset before the LOCK bit is set.

| | | | |
|---|---|---|---|
| 4 | R/W/L | 0b | **SMM Space Locked (D_LCK):** When D_LCK is set to a 1 then D_OPEN is reset to 0 and D_LCK, D_OPEN, G_SMRARE, C_BASE_SEG, H_SMRAM_EN, GMS, TOLUD, TOM, TSEG_SZ and TSEG_EN become read only. D_LCK can be set to 1 via a normal configuration space write but can only be cleared by a Full Reset. The combination of D_LCK and D_OPEN provide convenience with security. The BIOS can use the D_OPEN function to initialize SMM space and then use D_LCK to "lock down" SMM space in the future so that no application software (or BIOS itself) can violate the integrity of SMM space, even if the program has knowledge of the D_OPEN function.<br>This bit when set locks itself. |

- D_LCK is pretty much a necessity and is rarely left unset (5% or so of measured BIOS have D_LCK not set)
- When set prevents changes to a lot of registers

# Where can you find the all-important D_LCK bit?

- MCH3 & 4 = “SMRAM” register 0/0/0/9D
- 2nd Gen (Sandy Bridge) CPU and newer = “SMRAMC” register 0/0/0/88

# D_OPEN

**SMRAM - System Management RAM Control**

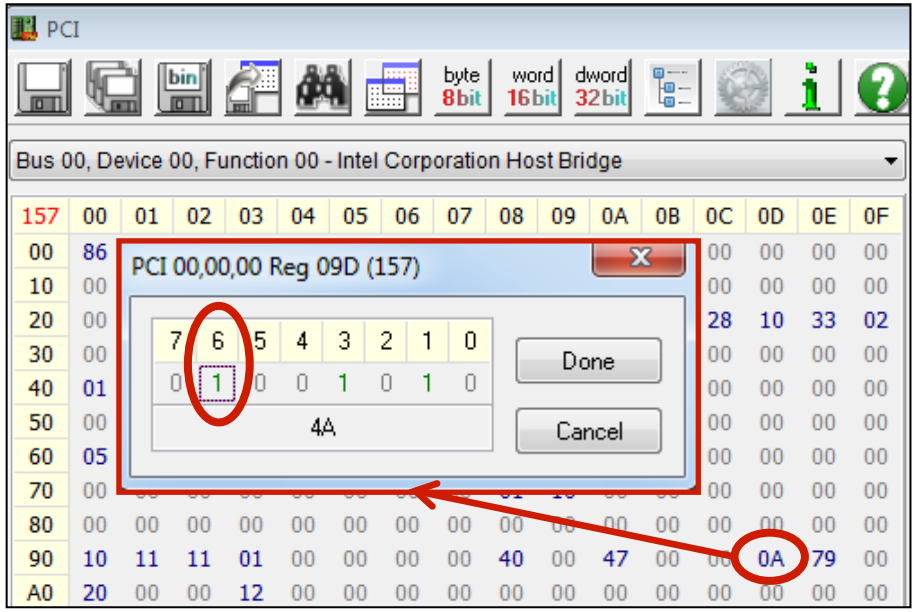
| | |
|---|---|
| B/D/F/Type: | 0/0/0/PCI |
| Address Offset: | 9Dh |
| Default Value: | 02h |
| Access: | RO; R/W/L; R/W |
| Size: | 8 bits |

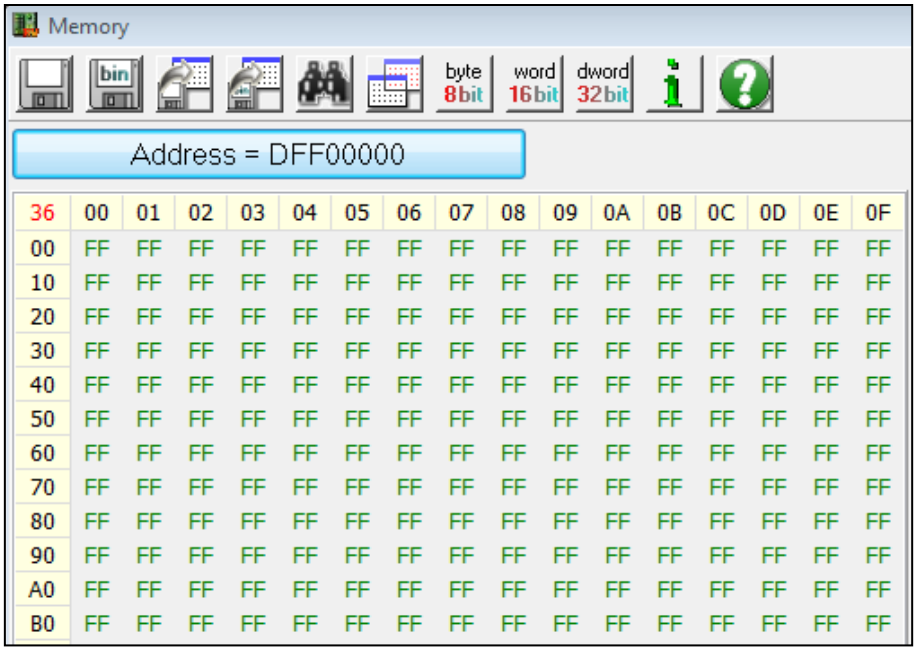
The SMRAMC register controls how accesses to Compatible and Extended SMRAM spaces are treated. The Open, Close, and Lock bits function only when G_SMRAME bit is set to a 1. Also, the OPEN bit must be reset before the LOCK bit is set.

| Bit | Access | Default Value | Description |
|---|---|---|---|
| 7 | RO | 0b | **Reserved** |
| 6 | R/W/L | 0b | **SMM Space Open (D_OPEN):** (When D_OPEN=1 and D_LCK=0, the SMM space DRAM is made visible even when SMM decode is not active. This is intended to help BIOS initialize SMM space. Software should ensure that D_OPEN=1 and D_CLS=1 are not set at the same time. This register is locked in Intel® TXT mode (RO in Intel TXT mode). It also locks when D_LCK bit is set. |

- To help the BIOS configure SMRAM, the chipset provides a means for leaving SMRAM open even when the processor is not in SMM
- D_LCK prevents this bit from being asserted

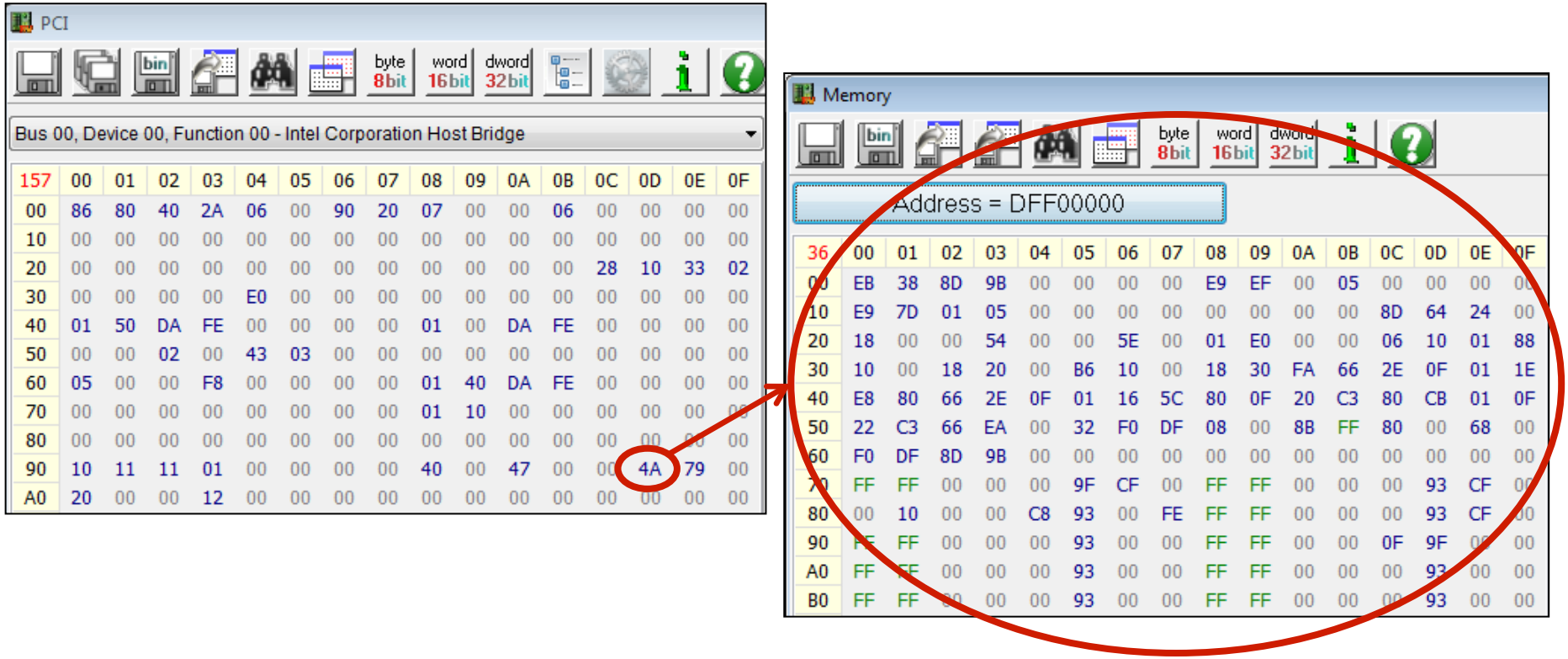
# vulnBIOS specific: Viewing SMRAM



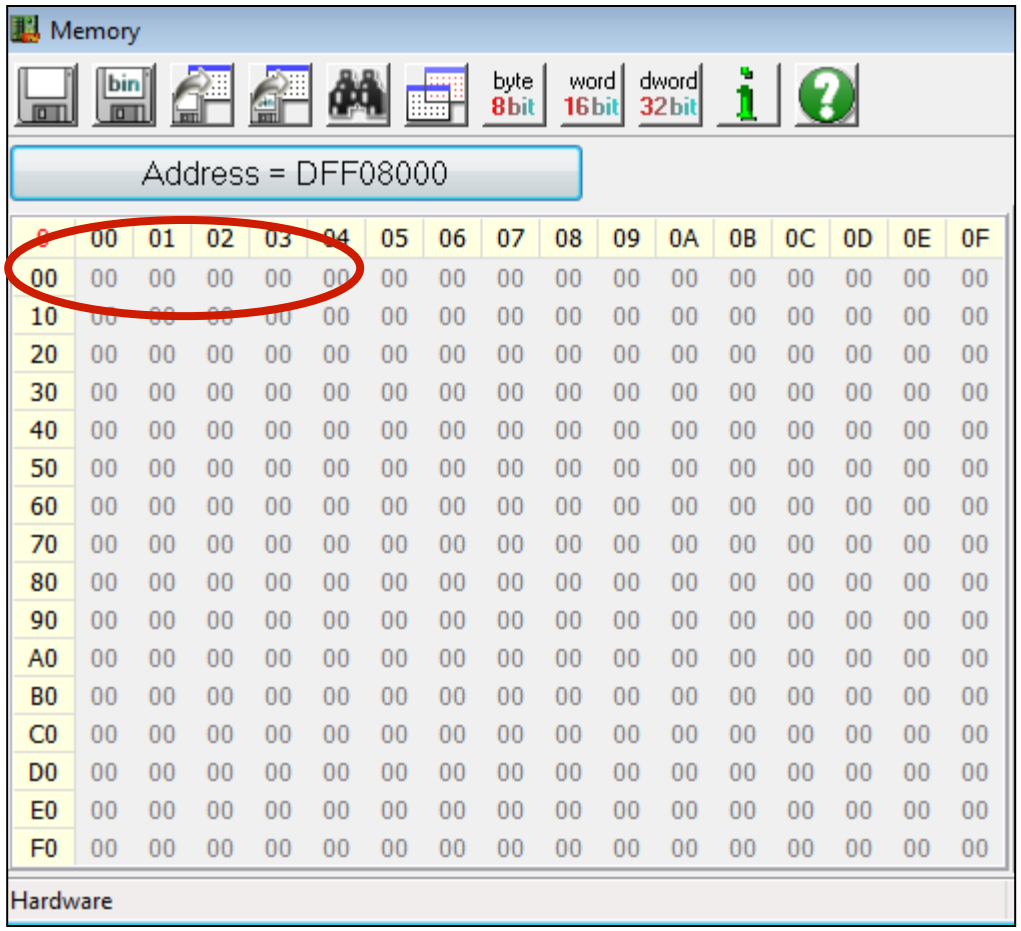

- If SMRAM is properly locked down this isn't possible
- Assert the D_OPEN bit in the SMRAMC register (D0:F0, offset 9Dh)
- Since it's unlocked, let's take a look at SMRAM
- According to our TSEG calculations it should be located at DFF0_0000h
- This *should be* SMBASE…but is it? Let's see…

# vulnBIOS specific: Viewing SMRAM



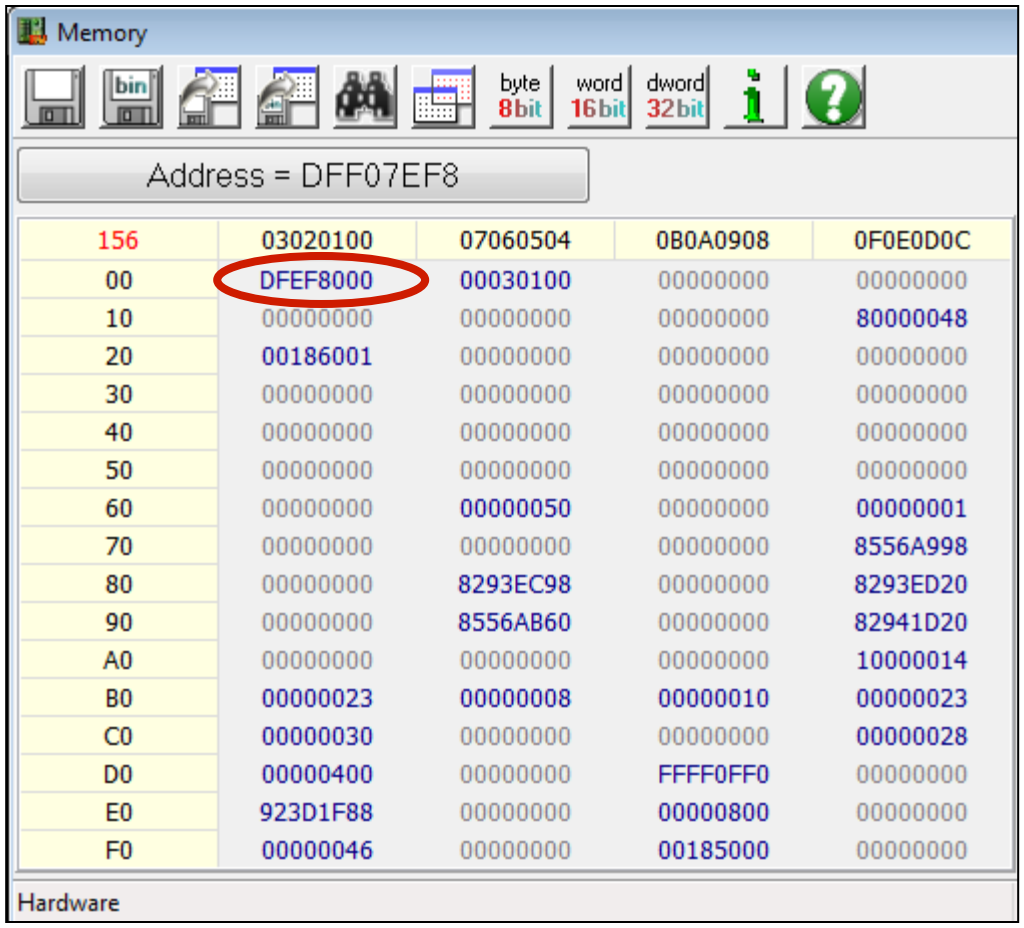

- Well, there is some binary at DFF0_0000h
- EB 38 is a JMP instruction which would take us to DFF0_003Ah
- But shouldn't our code enter at SMBASE + 8000h (DFF0_8000h) instead? Maybe this is just random bits.

# vulnBIOS specific: Viewing SMRAM




- Let's look at address DFF0_8000h
- Definitely *not* executable code
- Call it a hunch but let's look at address DFF0_7EF8h
- Recall that the SMBASE field in the state save register is located at offset SMBASE + 8000h + 7EF8h
- Let's see what that shows us

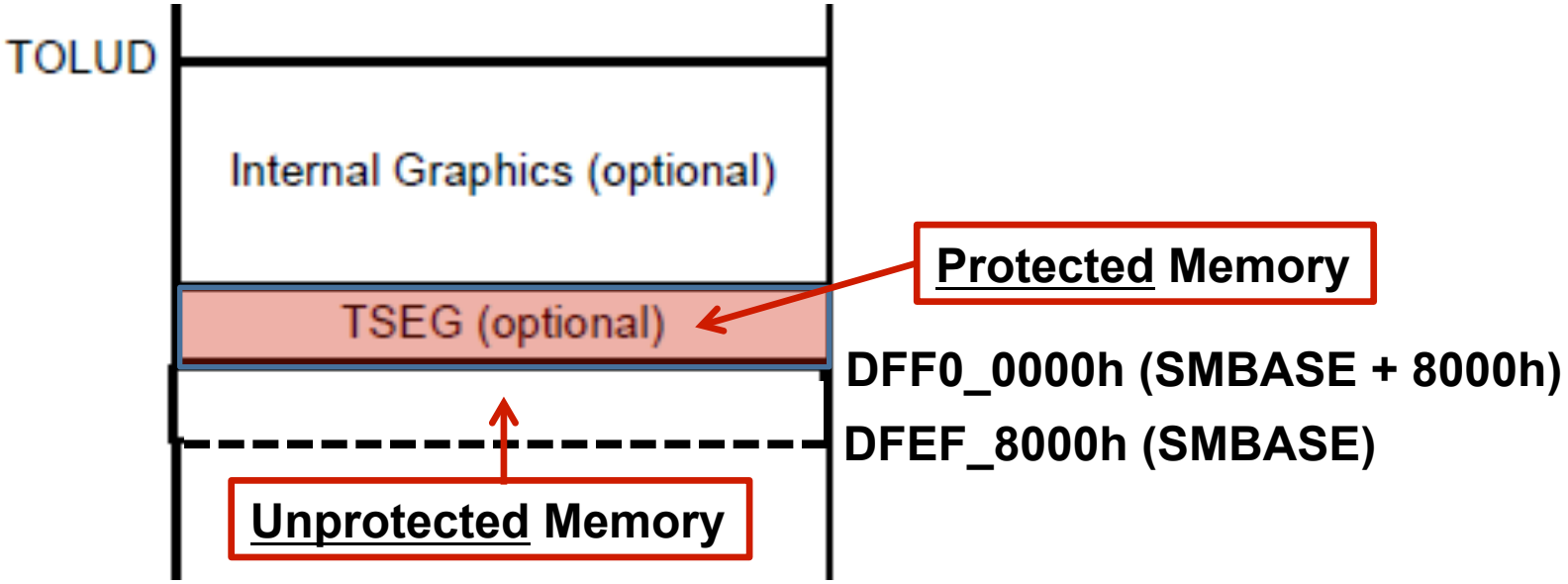
# vulnBIOS specific: Viewing SMRAM




| 156 | 03020100 | 07060504 | 0B0A0908 | 0F0E0D0C |
|---|---|---|---|---|
| 00 | DFEF8000 | 00030100 | 00000000 | 00000000 |
| 10 | 00000000 | 00000000 | 00000000 | 80000048 |
| 20 | 00186001 | 00000000 | 00000000 | 00000000 |
| 30 | 00000000 | 00000000 | 00000000 | 00000000 |
| 40 | 00000000 | 00000000 | 00000000 | 00000000 |
| 50 | 00000000 | 00000000 | 00000000 | 00000000 |
| 60 | 00000000 | 00000050 | 00000000 | 00000001 |
| 70 | 00000000 | 00000000 | 00000000 | 8556A998 |
| 80 | 00000000 | 8293EC98 | 00000000 | 8293ED20 |
| 90 | 00000000 | 8556AB60 | 00000000 | 82941D20 |
| A0 | 00000000 | 00000000 | 00000000 | 10000014 |
| B0 | 00000023 | 00000008 | 00000010 | 00000023 |
| C0 | 00000030 | 00000000 | 00000000 | 00000028 |
| D0 | 00000400 | 00000000 | FFFF0FF0 | 00000000 |
| E0 | 923D1F88 | 00000000 | 00000800 | 00000000 |
| F0 | 00000046 | 00000000 | 00185000 | 00000000 |

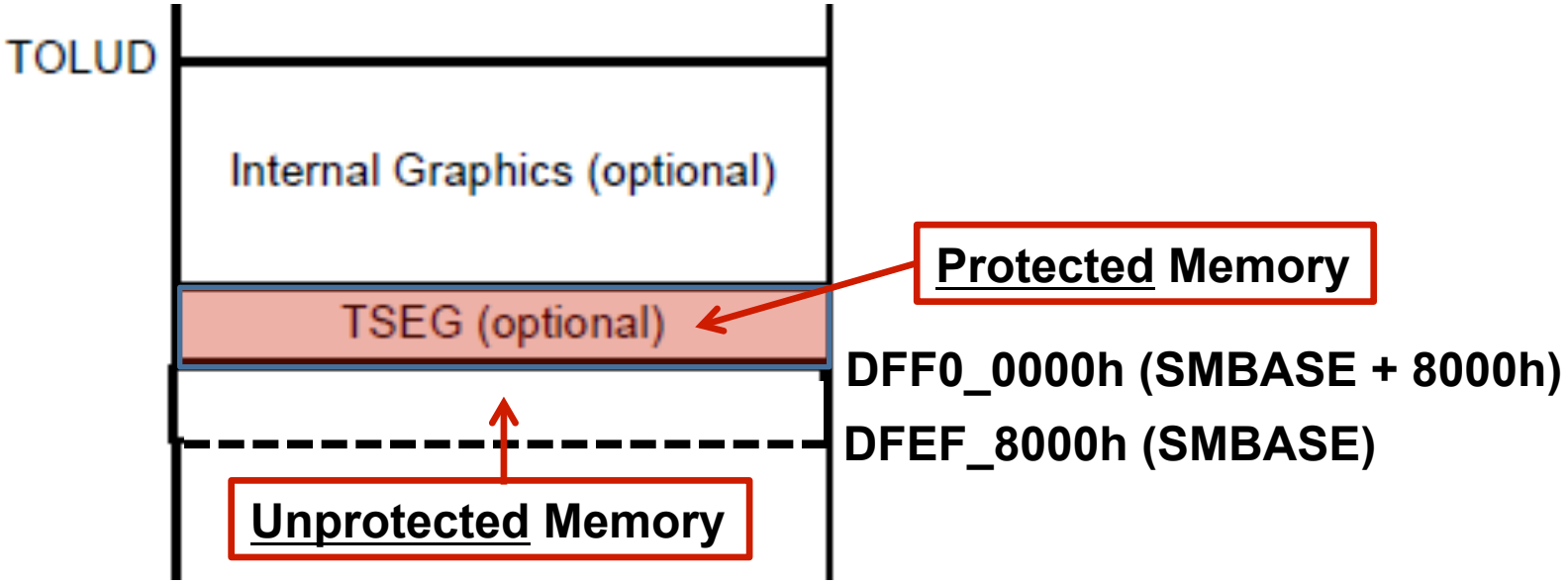
- This is the SMBASE value and its value is DFEF_8000h
- So SMBASE + 8000h is DFF0_0000h which is our TSEG base
- Technically, the SMRAM range is outside of the TSEG protected area

# Unprotected SMRAM Range




- We can see that it's outside the range if we go to an address just under DFF0_0000h and write some bytes
- Then we can toggle the D_OPEN bit off and on and see that the bytes we just wrote are still present whether SMRAM is open or closed

# So how bad is this?




- It's definitely not good! But…
- In *this* case the SMI handler neither references nor calls anything in this unprotected range
- So it's "okay" in this case, but could be catastrophic in another
- The moral of the story: Ensure that all SMI handler accessed code/data is within the protected memory range
  - ITL found multiple bugs in Intel's SMM code where it was accessing data outside the protected ranges, which could consequently be attacker controlled (which led to simple "change a function pointer to jump to my code" type attacks, and could lead to buffer overflow attacks)

# TODO

- Needs a discussion of TSEG as DMA protection