Advanced x86:
BIOS and System Management Mode Internals
SPI Flash

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”

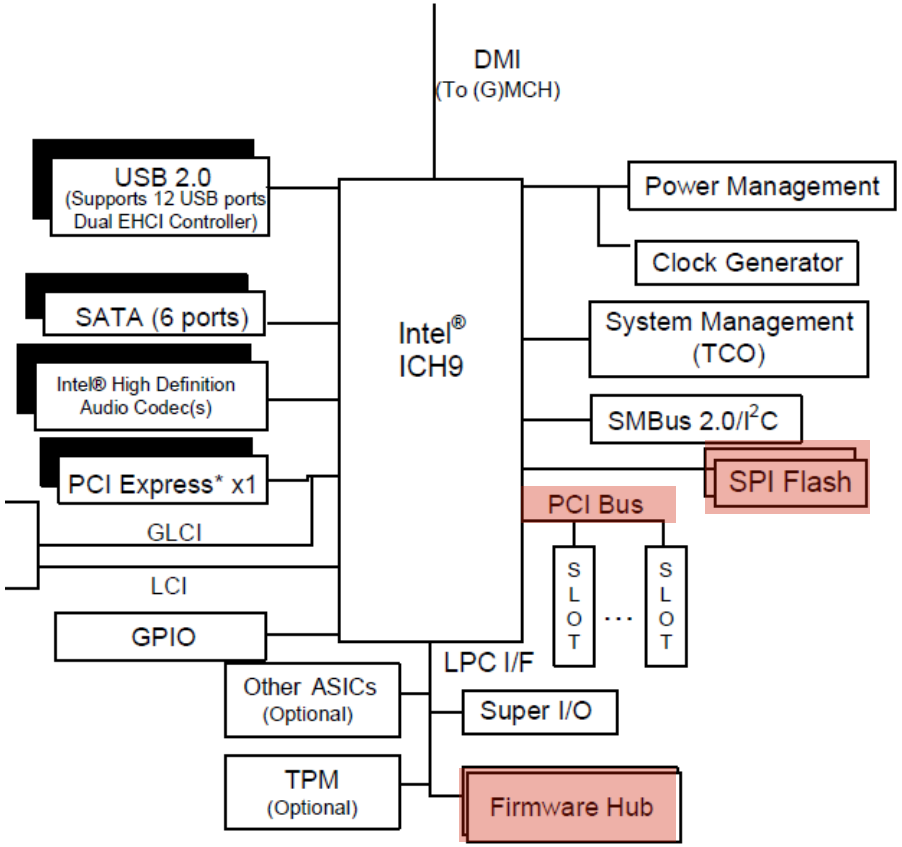
BIOS Flash Overview

- Everything we have talked about so far, although harmful to a system, isn't persistent unless you can write to the BIOS
- But one of the goals an attacker has in establishing a presence in the system is persistence
- To achieve persistence, the attacker will have to figure out a way to write to the BIOS flash so that upon every reboot, his presence is still there

Results of Copernicus checks

- We've used Copernicus to scan all of MITRE, and some other organizations.
- Originally (in 2013) the data said about 35% of systems were vulnerable.
- Then we found more problems and it went up to 55%
- Then people patched and it went down to 35%
- Then we found more problems and it went up to 60%
- Then we found more problems and it went up to 85%
- And if the organizations had never patched, and we looked at our first data with our last knowledge?
- 99.95% vulnerable

BIOS Flash Location



- BIOS can reside in one of 3 locations:

1. Firmware Hub (FWH)
   - Older technology and mostly out of scope for this class
2. SPI Flash
   - Most likely location
3. PCI
   - intended for debugging or recovering from a corrupted BIOS (not supported anymore on newer hardware)

Boot BIOS Flash Location

Signal	Usage	When Sampled	Comment
GNT0#	Boot BIOS Destination Selection 1	Rising Edge of PWROK	This field determines the destination of accesses to the BIOS memory range. Signals have weak internal pull-ups. Also controllable via Boot BIOS Destination bit (Chipset Config Registers:Offset 3410h:bit 11). This strap is used in conjunction with Boot BIOS Destination Selection 0 strap. **Bit11 (GNT0#) / Bit 10 (SPI_CS1#) / Boot BIOS Destination** 0 / 1 / SPI 1 / 0 / PCI 1 / 1 / LPC 0 / 0 / Reserved **NOTE:** Booting to PCI is intended for debug/testing only. Boot BIOS Destination Select to LPC/PCI by functional strap or via Boot BIOS Destination Bit will not affect SPI accesses initiated by Management Engine or Integrated GbE LAN.
SPI_CS1# / GPIO58 Desktop Only) / CLGPIO6 (Digital Office Only)	Boot BIOS Destination Selection 0	Rising Edge of CLPWROK	This field determines the destination of accesses to the BIOS memory range. Signals have weak internal pull-ups. Also controllable via Boot BIOS Destination bit (Chipset Config Registers:Offset 3410h:bit 10). This strap is used in conjunction with Boot BIOS Destination Selection 1 strap. **Bit11 (GNT0#) / Bit 10 (SPI_CS1#) / Boot BIOS Destination** 0 / 1 / SPI 1 / 0 / PCI 1 / 1 / LPC 0 / 0 / Reserved **NOTE:** Booting to PCI is intended for debug/testing only. Boot BIOS Destination Select to LPC/PCI by functional strap or via Boot BIOS Destination Bit will not affect SPI accesses initiated by Management Engine or Integrated GbE LAN.

- The boot destination is decided by the configuration of the following pins on the ICH/ PCH*
- Pins are sampled at power-up to determine location of BIOS
- Intended for static configuration
- PCI boot is intended only for debugging or recovering from corrupt BIOS (so not necessarily static)
- But since these are hardware pins, it's worth checking if PCI is set as the boot location, because you might have a physical hardware implant!

* References to ICH/PCH mean applicable to both legacy and modern chipsets

Example: Find BIOS Boot Destination

GCS—General Control and Status Register

Offset Address: 3410–3413h Attribute: R/W, R/WLO
Default Value: 00000yy0h (yy = xx0000x0b)Size: 32-bit

Bit	Description
11:10	**Boot BIOS Straps (BBS)** — R/W. This field determines the destination of accesses to the BIOS memory range. The default values for these bits represent the strap values of GNT0# (bit 11) at the rising edge of PWROK and SPI_CS1#/GPIO58 (Desktop Only) /CLGPIO6 (Digital Office Only) (bit 10) at the rising edge of CLPWROK. **Bits 11:10 — Description** 0xb — SPI 10b — PCI 11b — LPC When PCI is selected, the top 16MB of memory below 4GB (FF00_0000h to FFFF_FFFFh) is accepted by the primary side of the PCI P2P bridge and forwarded to the PCI bus. This allows systems with corrupted or unprogrammed flash to boot from a PCI device. The PCI-to-PCI bridge Memory Space Enable bit does not need to be set (nor any other bits) in order for these cycles to go to PCI. Note that BIOS decode range bits and the other BIOS protection bits have no effect when PCI is selected. This functionality is intended for debug/testing only. When SPI or LPC is selected, the range that is decoded is further qualified by other configuration bits described in the respective sections. The value in this field can be overwritten by software as long as the BIOS Interface Lock-Down (bit 0) is not set. **NOTE:** Booting to PCI is intended for debug/testing only. Boot BIOS Destination Select to LPC/PCI by functional strap or via Boot BIOS Destination Bit will not affect SPI accesses initiated by Management Engine or Integrated GbE LAN.

01b SPI (typo in datasheet)

- To programmatically find where your BIOS is configured to boot from, you can also view bits 11:10 in the General Control and Status Register (GCS)
- Located at memory-mapped offsets 3410-3413h in the Chipset Configuration Registers
- Chipset Configuration Registers are mapped starting at the address held by RCBA…you know, RCRB? :)

Verify GCS location on your datasheet if not using the class E6400.

Reminder: RCBA/RCRB

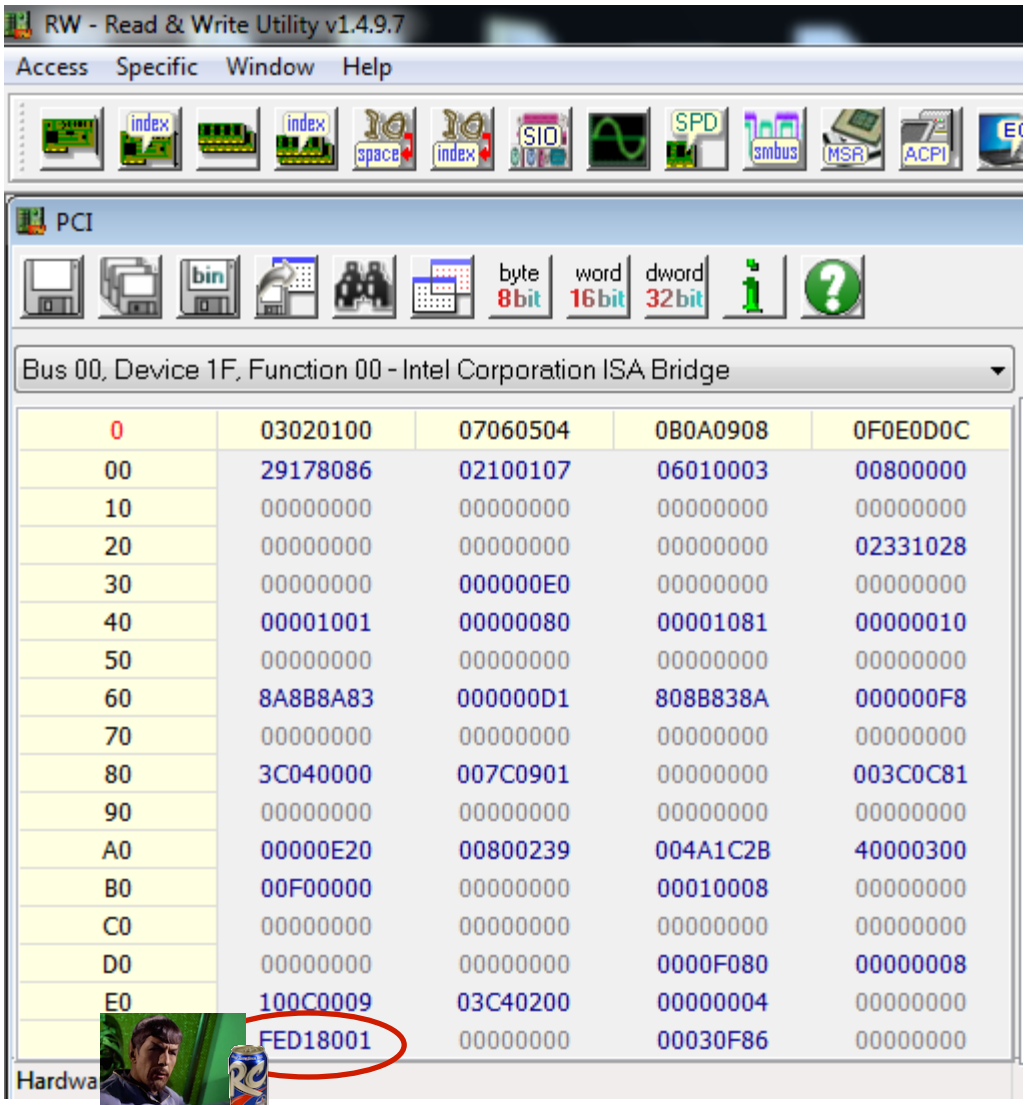
RCBA—Root Complex Base Address Register (LPC I/F—D31:F0)

Offset Address:	F0-F3h	Attribute:	R/W
Default Value:	00000000h	Size:	32 bit

Bit	Description
31:14	**Base Address (BA)** — R/W. Base Address for the root complex register block decode range. This address is aligned on a 16-KB boundary.
13:1	Reserved
0	**Enable (EN)** — R/W. When set, enables the range specified in BA to be claimed as the Root Complex Register Block.

- The Root Complex Register Block (RCRB) decode range is located in the Root Complex Base Address (RCBA) register located in the LPC (D31:F0, offset F0-F3h)
- The root complex is PCI-Express related. It connects the processor and memory to the PCI Express devices.
  - If you want to know more about the inner workings of PCI Express, there are a number of good sources, such as (Darmawan):
  - http://resources.infosecinstitute.com/system-address-map-initialization-x86x64-architecture-part-2-pci-express-based-systems/

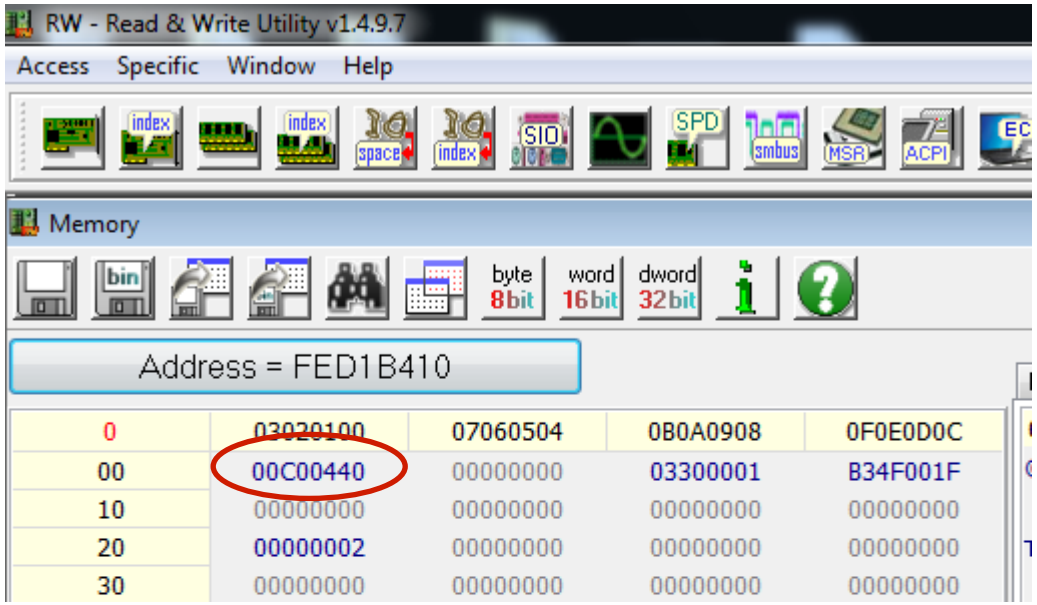
Example: Find BIOS Boot Location



- Locate RCRB:
- Bit 0 is just an enable bit (the nibble this bit is in is still part of the address, but change it to 0)
- Here the RCRB begins at FED1_8000h
- The GCS register is located at in the chipset configuration registers.
- At RCRB + 3410h = FED1_B410h

The root complex base address will differ on different systems.

Example: Find BIOS Boot Location



Bits 11:10

0	1	SPI
1	0	PCI
1	1	LPC
0	0	Reserved

- GCS at FED1_B410h yields the following value on our lab system:
- 00C0_0440h
- Bits 11:10 are 01b which indicates that this BIOS boots from SPI
- But how can we trust what this says? We're not actually sampling the Controller's pins in this register

Example: Change BIOS Access Destination

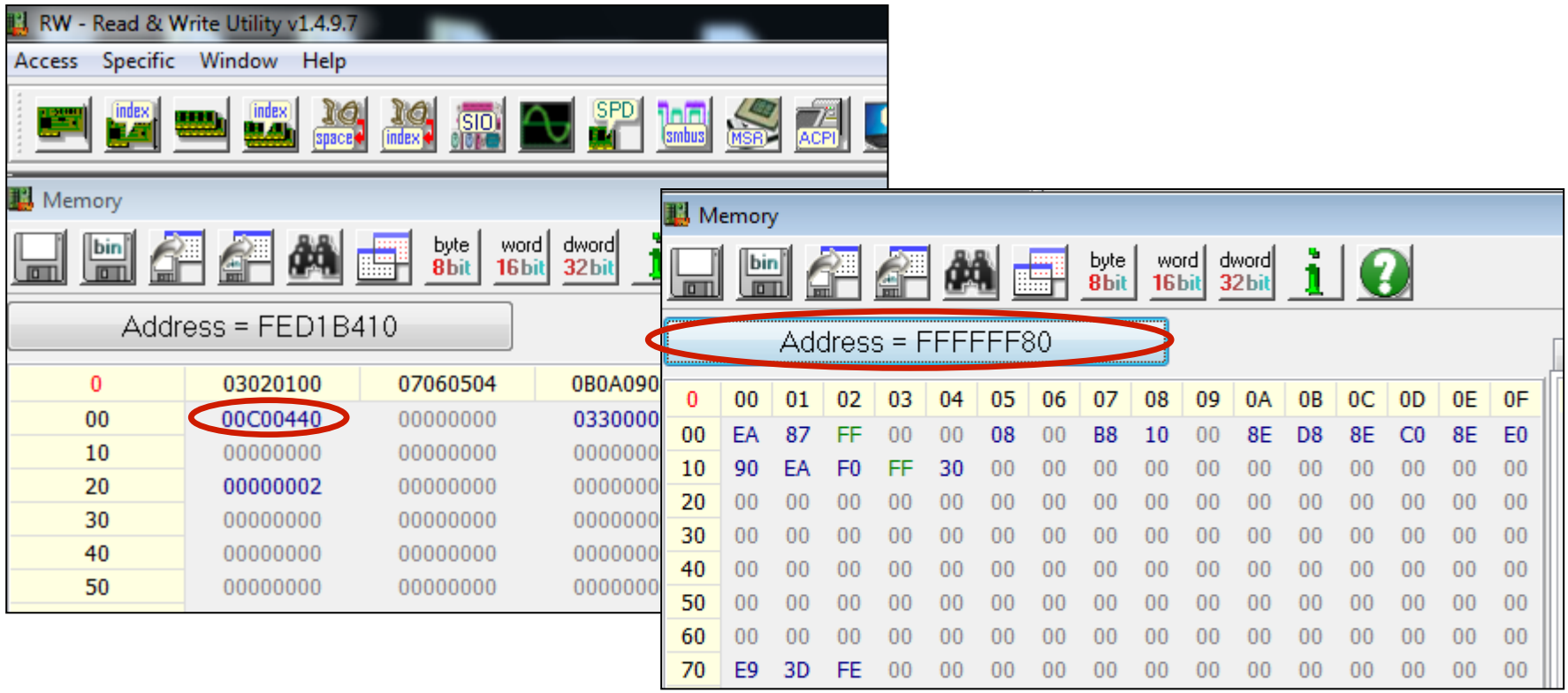
GCS—General Control and Status Register

Offset Address: 3410–3413h Attribute: R/W, R/WLO
Default Value: 00000yy0h (yy = xx0000x0b)Size: 32-bit

Bit	Description
11:10	**Boot BIOS Straps (BBS)** — R/W. This field determines the destination of accesses to the BIOS memory range. The default values for these bits represent the strap values of GNT0# (bit 11) at the rising edge of PWROK and SPI_CS1#/GPIO58 (Desktop Only) /CLGPIO6 (Digital Office Only) (bit 10) at the rising edge of CLPWROK. **Bits 11:10** — **Description** 0xb — SPI 10b — PCI 11b — LPC When PCI is selected, the top 16MB of memory below 4GB (FF00_0000h to FFFF_FFFFh) is accepted by the primary side of the PCI P2P bridge and forwarded to the PCI bus. This allows systems with corrupted or unprogrammed flash to boot from a PCI device. The PCI-to-PCI bridge Memory Space Enable bit does not need to be set (nor any other bits) in order for these cycles to go to PCI. Note that BIOS decode range bits and the other BIOS protection bits have no effect when PCI is selected. This functionality is intended for debug/testing only. When SPI or LPC is selected, the range that is decoded is further qualified by other configuration bits described in the respective sections. The value in this field can be overwritten by software as long as the BIOS Interface Lock-Down (bit 0) is not set. **NOTE:** Booting to PCI is intended for debug/testing only. Boot BIOS Destination Select to LPC/PCI by functional strap or via Boot BIOS Destination Bit will not affect SPI accesses initiated by Management Engine or Integrated GbE LAN.

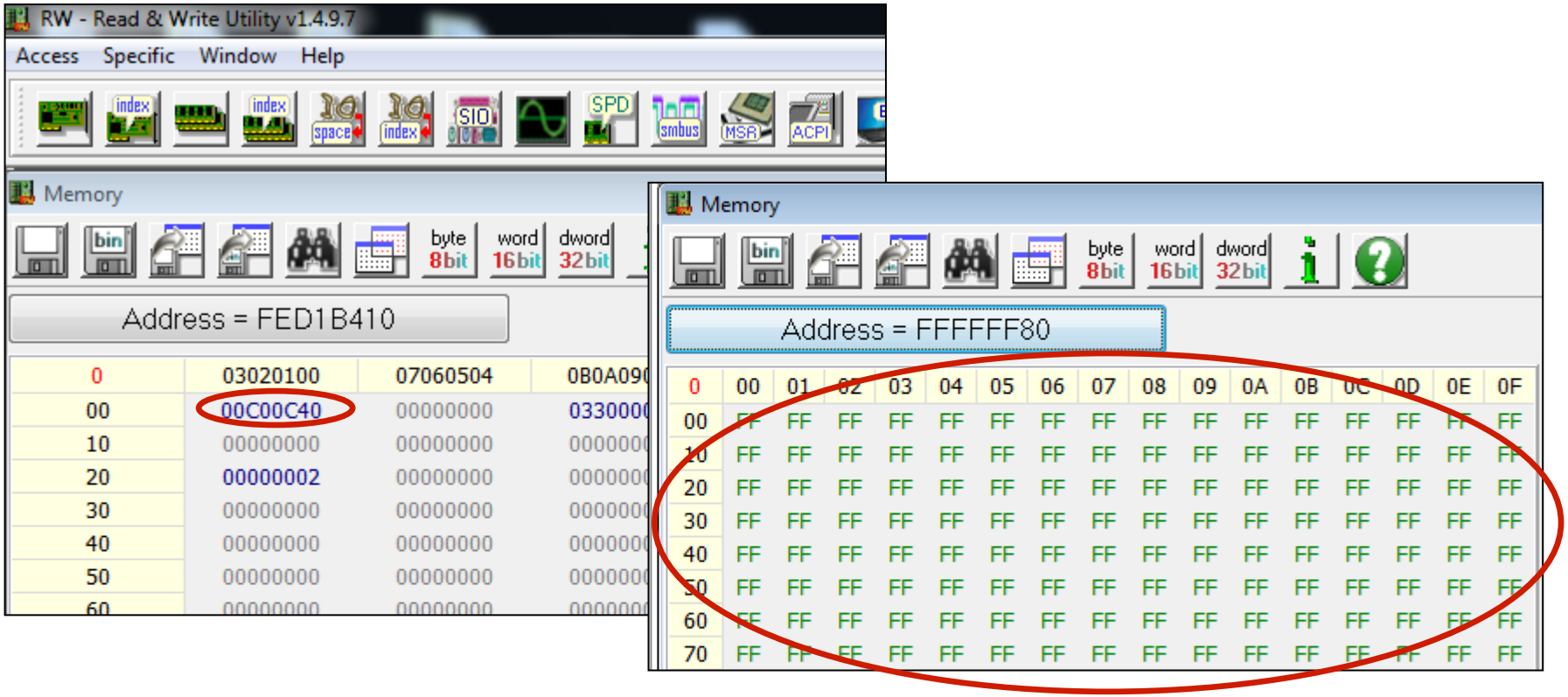
- Notice these bits are R/W?
- You can change the destination for BIOS accesses
- Likely this is to help the system recover from a corrupted BIOS
- But it could be certainly misused as well
- Note just to be clear: The bits in GCS alter accesses to the BIOS *only* after the BIOS has begun booting
  - Chipset Configuration registers must be mapped to memory, etc.
- The functional straps are physical pins which cannot be altered and decide the BIOS Boot Location

Example: Change BIOS Access Destination



- Bring up a memory window and go to an address which shows the memory-mapped BIOS (like FFFF_FF80h which will show us the entry vector)
- You should see the BIOS in memory

Example: Change BIOS Access Destination



- Modify the GCS register to 00C00C40h, bits 11:10 are 11b now which point the device to the LPC
- On our lab system the LPC has no firmware BIOS so this translates to reads of all 1's (0xFF)
- Your personal system may differ and you may actually see valid binary here.

Example: LOCK BIOS Access Destination

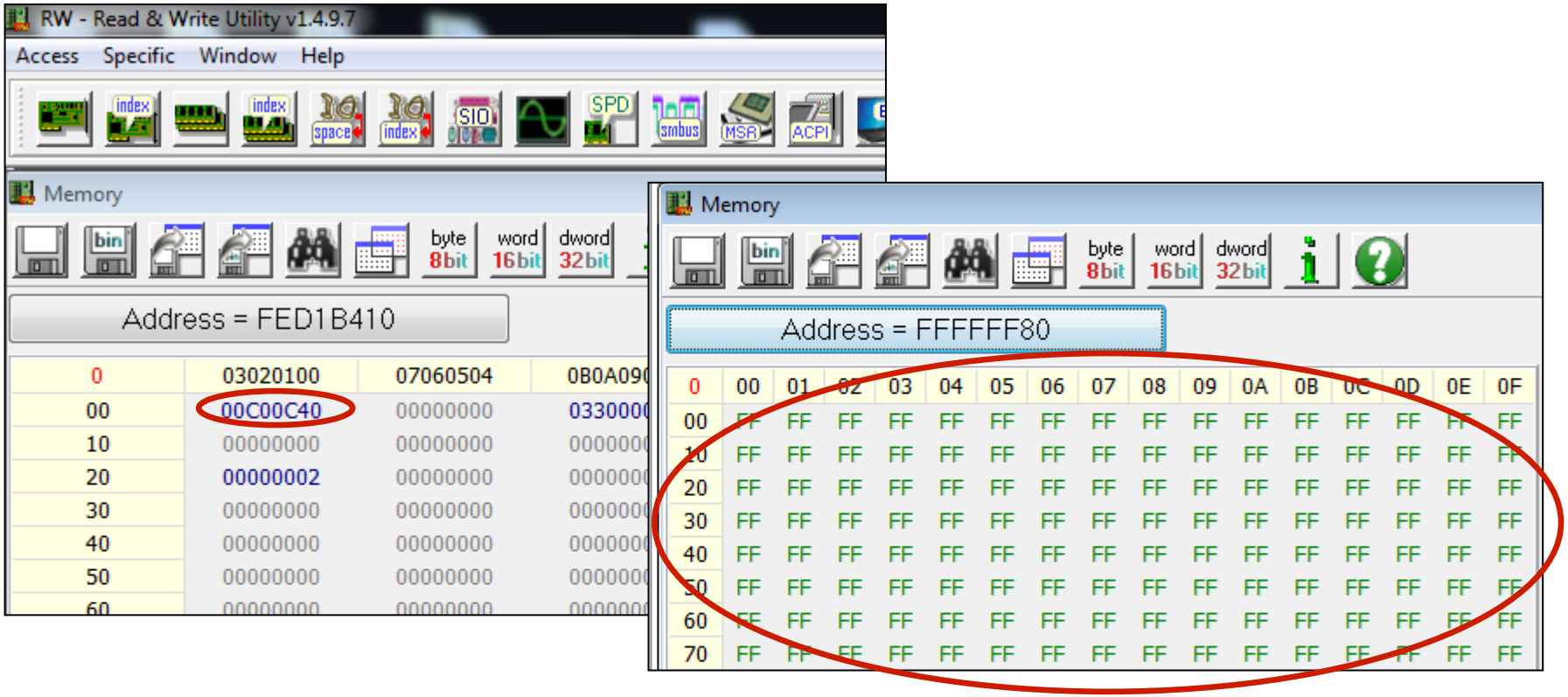
GCS—General Control and Status Register

Offset Address: 3410–3413h Attribute: R/W, R/WLO
Default Value: 00000yy0h (yy = xx0000x0b) Size: 32-bit

Bit	Description
11:10	**Boot BIOS Straps (BBS)** — R/W. This field determines the destination of accesses to the BIOS memory range. The default values for these bits represent the strap values of GNT0# (bit 11) at the rising edge of PWROK and SPI_CS1#/GPIO58 (Desktop Only) /CLGPIO6 (Digital Office Only) (bit 10) at the rising edge of CLPWROK. **Bits 11:10** — **Description** 0xb — SPI 10b — PCI 11b — LPC When PCI is selected, the top 16MB of memory below 4GB (FF00_0000h to FFFF_FFFFh) is accepted by the primary side of the PCI P2P bridge and forwarded to the PCI bus. This allows systems with corrupted or unprogrammed flash to boot from a PCI device. The PCI-to-PCI bridge Memory Space Enable bit does not need to be set (nor any other bits) in order for these cycles to go to PCI. Note that BIOS decode range bits and the other BIOS protection bits have no effect when PCI is selected. This functionality is intended for debug/testing only. When SPI or LPC is selected, the range that is decoded is further qualified by other configuration bits described in the respective sections. The value in this field can be overwritten by software as long as the BIOS Interface Lock-Down (bit 0) is not set. **NOTE:** Booting to PCI is intended for debug/testing only. Boot BIOS Destination Select to LPC/PCI by functional strap or via Boot BIOS Destination Bit will not affect SPI accesses initiated by Management Engine or Integrated GbE LAN.
0	**BIOS Interface Lock-Down (BILD)** — R/WLO. 0 = Disabled. 1 = Prevents BUC.TS (offset 3414, bit 0) and GCS.BBS (offset 3410h, bits 11:10) from being changed. This bit can only be written from 0 to 1 once.

- Intel provides a way to lock down the destination of BIOS accesses
- When bit 0 in the General Control and Status Register (GCS) is set, bits 11:10 become Read-Only
- The BIOS should lock this down!

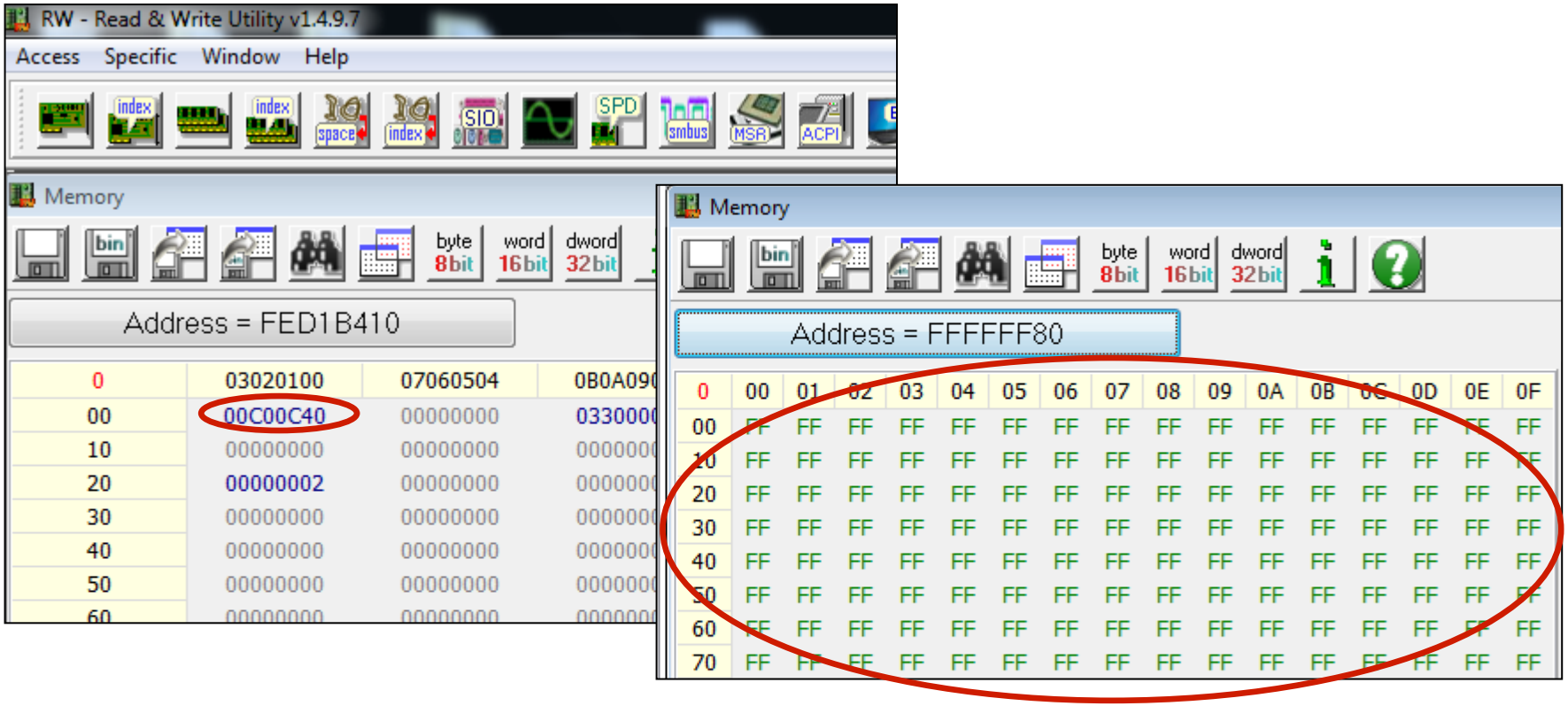
Example: Change BIOS Access Destination



- Set bits 11:10 in the GCS register back to their original values (01b for SPI)*
- Assert bit 1 in GCS, now GCS is 00C00441h
- Now find that bits 11:10 are fixed in place

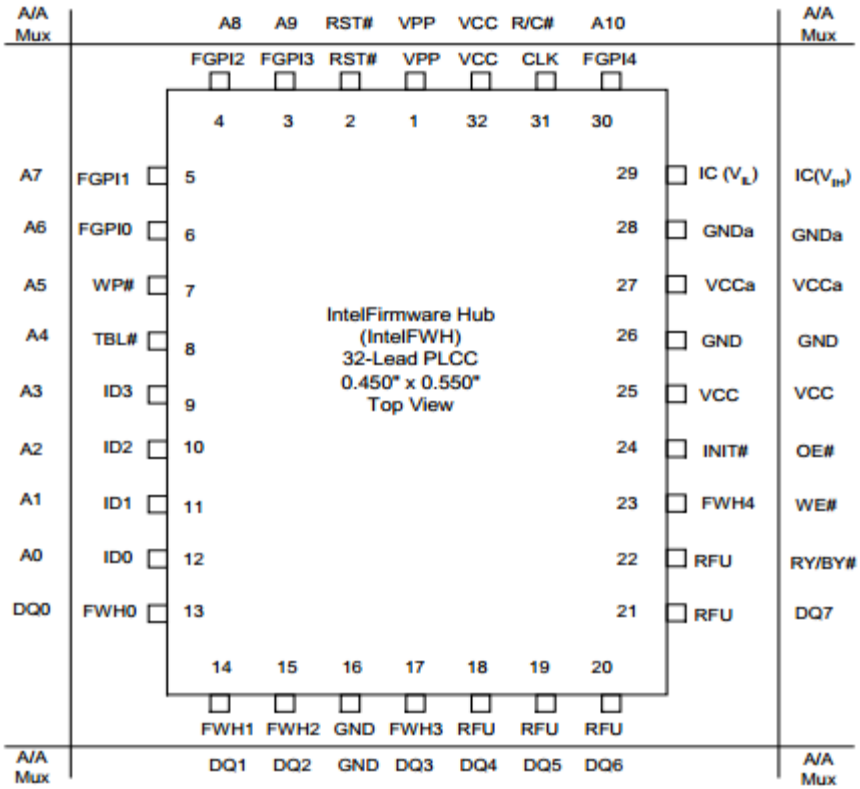
*Or leave them pointing to nothing, this is not permanent and nothing a reboot won't reset

A Word About This



- This only affects *direct* (memory) accesses to BIOS flash
- Programs (like Copernicus or Flashrom) that read directly from the BIOS flash using the SPI programming registers (for example) will still successfully read the BIOS binary from the chip

Firmware Hub (FWH)



- Provides register-based R/W protection for each code/data storage block
- Has hardware write-protect pins for the top boot block and the remaining code/data storage blocks
- Contains a Random Number Generator (RNG)
- More than one FWH device can be supported
- Operates at 33 MHz (synchronous to the PCI bus)
- Has a lot of pins compared to SPI

Intel 82802AB/82802AC Firmware Hub (FWH)

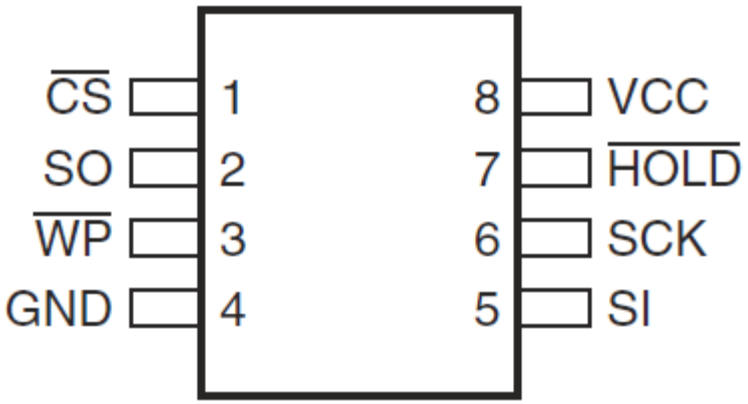
Firmware Hub (FWH)

Memory Address	Mnemonic	Register Name	Default	Type
FFBF0002h	T_BLOCK_LK	Top Block Lock Register (4-8-Mbit FWH)	01h	R/W
FFBE0002h	T_MINUS01_LK	Top Block [-1] Lock Register (4-8-Mbit FWH)	01h	R/W
FFBD0002h	T_MINUS02_LK	Top Block [-2] Lock Register (4-8-Mbit FWH)	01h	R/W
FFBC0002h	T_MINUS03_LK	Top Block [-3] Lock Register (4-8-Mbit FWH)	01h	R/W
FFBB0002h	T_MINUS04_LK	Top Block [-4] Lock Register (4-8-Mbit FWH)	01h	R/W
FFBA0002h	T_MINUS05_LK	Top Block [-5] Lock Register (4-8-Mbit FWH)	01h	R/W
FFB90002h	T_MINUS06_LK	Top Block [-6] Lock Register (4-8-Mbit FWH)	01h	R/W
FFB80002h	T_MINUS07_LK	Top Block [-7] Lock Register (4-8-Mbit FWH)	01h	R/W
FFB70002h	T_MINUS08_LK	Top Block [-8] Lock Register (8-Mbit FWH)	01h	R/W
FFB60002h	T_MINUS09_LK	Top Block [-9] Lock Register (8-Mbit FWH)	01h	R/W
FFB50002h	T_MINUS10_LK	Top Block [-10] Lock Register (8-Mbit FWH)	01h	R/W
FFB40002h	T_MINUS11_LK	Top Block [-11] Lock Register (8-Mbit FWH)	01h	R/W
FFB30002h	T_MINUS12_LK	Top Block [-12] Lock Register (8-Mbit FWH)	01h	R/W
FFB20002h	T_MINUS13_LK	Top Block [-13] Lock Register (8-Mbit FWH)	01h	R/W
FFB10002h	T_MINUS14_LK	Top Block [-14] Lock Register (8-Mbit FWH)	01h	R/W
FFB00002h	T_MINUS15_LK	Top Block [-15] Lock Register (8-Mbit FWH)	01h	R/W
FFBC0100h	FGPI_REG	FWH General-Purpose Input Register	N/A	RO
FFBC015Fh		RNG Hardware Status Register	40h*	R/W
FFBC0160h		RNG Data Status Register	0	RO
FFBC0161h		RNG Data Register	N/A	RO

Intel 82802AB/82802AC Firmware Hub (FWH)

- Memory-mapped interface
- Programmable Erase, Read, Write commands
- Each block can be locked down to prevent Reads and/or Writes
- Firmware hubs are rare (at least in modern PC's) and we have never seen one
- Sample FWH datasheet:
- http://download.intel.com/design/chipsets/datashts/29065804.pdf
- If you ever encounter a system with a firmware hub email me and tell me the make/model please

Serial Peripheral Interface (SPI)



Typically 8 pins, can be 16

- Intel's ICH/PCH implements a SPI interface for the BIOS flash device
- Used as a replacement for the Firmware Hub (FWH) on LPC
- SPI is required in order to support the Management Engine (ME), Gigabit Ethernet (GbE), and others.
- Each SPI flash device can be up to 16 MB (2^{24} bits)

- SPI controller can support 1 or 2 devices for 32 MB maximum addressable space
- Lower cost alternative (per Intel datasheet)
- Memory-mapped programming interface offset from RCRB (consult your datasheet for its exactly offset)

*Based on datasheet information and that the Flash Address Register accepts addresses occupying bits 24:0

SPI Overview

- SPI protocol can support data rates up to 100 MHz
  - Intel's implementation is configurable to operate at either 20 MHz or 33 MHz (or 50 MHz on the newer PCI Express systems), or 66MHz
- Intel abstracts most of the low-level SPI protocol from you
- SPI protocol is not a fixed standard
  - Different chips will support different commands and so forth
- Intel defines a set of minimum requirements for a chip to support.
  - Likely though each chip will support more than just that bare minimum
- So we'll be covering Intel's implementation and interface to SPI, not really the SPI protocol itself (they intertwine somewhat of course).

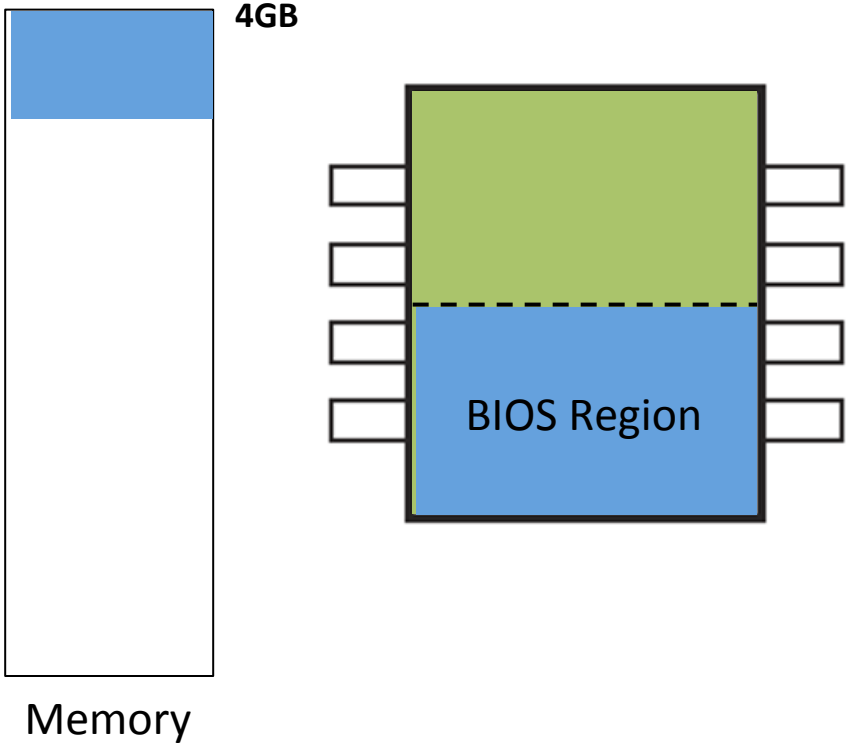
SPI Operating Modes

- Since I/O Controller Hub version 8, the SPI flash has been able to support 2 distinct operating modes:
- Non-Descriptor Mode (~~RIP, deceased '09~~)
  - IT LIVES! (On embedded Intel Atom devices like MinnowBoard!)
  - In ICH7 this is the only supported operating mode
- Descriptor Mode
  - Since ICH8 (so ICH8, ICH9, ICH10, and PCH)
- For systems that have a Platform Controller Hub device (PCH), non-descriptor mode has been phased out and is no longer supported

Descriptor Mode

- Enables chipset features like:
  - Integrated Gigabit Ethernet, Host processor for Gigabit Ethernet Software, Management Engine
- Provides support for two SPI flash chips
- Divides the SPI flash into regions
- Provides hardware enforced security restricting region access
- Chipset Soft Strap region provides the ability to use Flash NVM as an alternative to hardware pull-up/pull-down resistors for both ICH and PCH
  - On reset, the controller hub reads the soft strap data out of the SPI flash
- Can be programmed (at a minimum) using the commands specified in the Intel ICH/PCH datasheet
  - But each chip can support additional commands, not very standardized

Memory Mapping: Descriptor Mode



- All of the flash chip is mapped to high memory
- In Descriptor Mode, only the BIOS region of the flash is readable in memory
- All other regions return 0xFF on reads
  - We'll get to the other regions in a bit

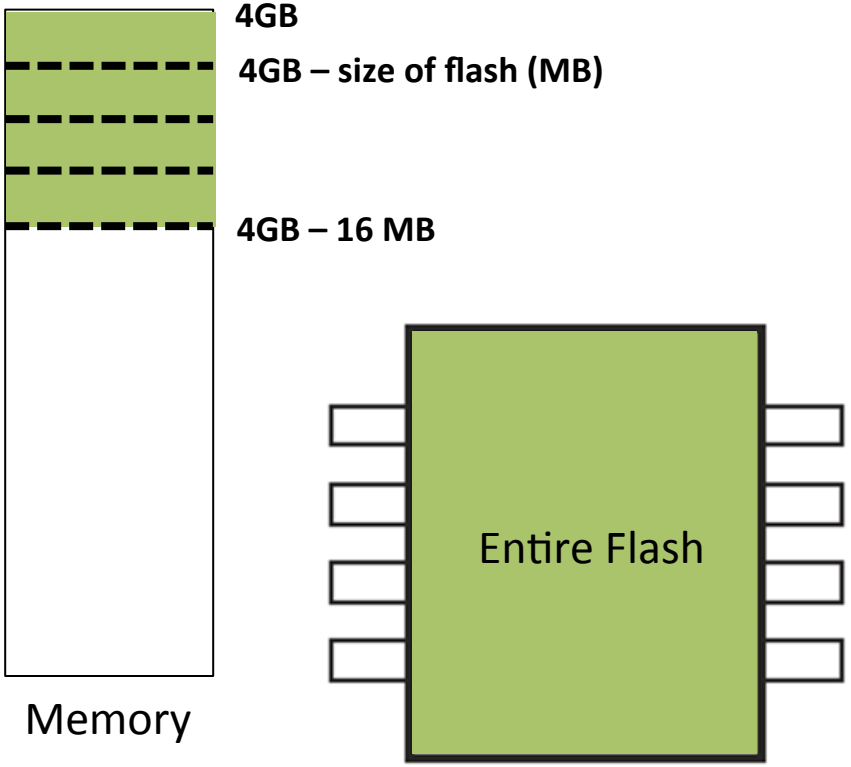
Flash Contents

Flash contents that are viewable in Memory

Non-Descriptor Mode

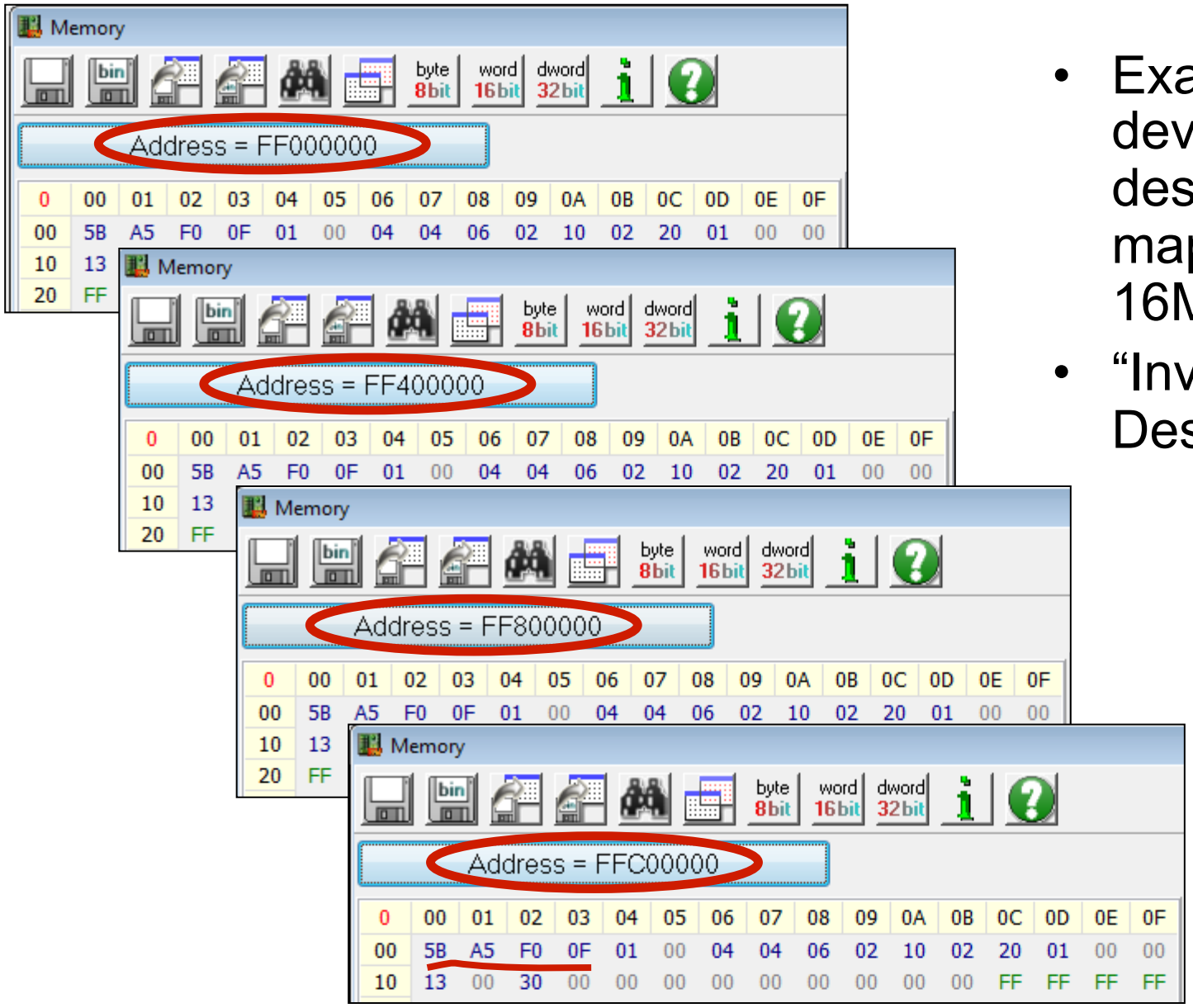
- Best described by its lack of features (as compared to Descriptor mode)
- The entire flash is used for BIOS (this does not mean the BIOS will be larger)
- Security features available in Descriptor mode are not available in Non-Descriptor mode
  - The BIOS/CPU can read/write to the flash without restriction
- Therefore there is also no support for Gb Ethernet, Management Engine, or chipset soft straps
- Interesting quote in Intel's ICH datasheet (10, in this case): "[in Non-Descriptor Mode], Direct read and writes are not supported."
- 'Non-Descriptor Mode == !Descriptor Mode'
- No longer a viable option on the newer PCH systems, since they require a valid flash descriptor

Memory Mapping: Non-Descriptor Mode



- In Non-Descriptor Mode the entire flash contents are visible in memory (more than just BIOS, if any more is present)
- If flash is < 16 MB and the FWH decoders are enabled in LPC, you will see the BIOS mapped repeatedly (think ribbons) at high memory
  - A 4MB BIOS is mapped 4 times in the high 16 MB of memory space
- A flash device in descriptor mode that has its descriptor signature “corrupted” will be viewable in memory in its entirety
  - But the descriptor signature is protected, so that would require physical flash access to corrupt

Non-Descriptor Mode Memory Mapping



- Example of 4 MB device in “non-descriptor” mode mapped to high 16MB of memory
- “Invalid” Flash Descriptor

0FF0A55Bh instead of 0FF0A55Ah

Why is some of the chip visible in memory in one mode but not the other?

- Has to do with the type of flash access as well as permissions to read that memory:
- There is an SPI “rule” that states:
  - Every SPI Master has direct read access to it’s own region only
  - Direct Access refers to memory reads in mapped memory
  - Thus the BIOS Master can read the BIOS region in memory (mapped to high mem at 4 GB)
- In Descriptor mode, the SPI flash is divided into regions
  - BIOS region, Flash Descriptor, etc. (we’ll cover in more detail soon)
- Therefore, in Descriptor Mode, only the BIOS region can be seen in high mapped-memory
- In Non-Descriptor mode, there is no concept of regions
  - It’s just “the BIOS”
- So therefore, the entire “BIOS” (entire flash) can be seen in memory when the SPI flash is in Non-Descriptor mode

Flash Accesses: Direct vs. Register

- Direct Access
  - This applies to memory accesses (mapped to high-memory)
  - Masters are allowed to read *only* their own region
    - CPU/BIOS can read the BIOS region
    - Management Engine can read only the ME region
    - GbE controller can read the GbE region (GbE software must use the programming registers)

- Register Access
  - Access a region by programming the base address registers
  - Register accesses are not allowed to cross a 4 KB aligned boundary
  - Cannot execute a command that may extend across to a second SPI flash (if present)
  - Software must know the SPI flash linear address it is trying to read