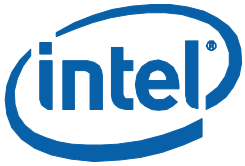


White Paper

Jenny M Pelner
James A Pelner
Firmware Architects
Intel Corporation

Minimal Intel Architecture Boot Loader

Bare Bones Functionality Required for Booting an Intel Architecture Platform

January 2010

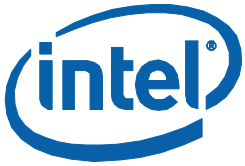
323246

Executive Summary

The intent of this White paper is to describe the minimal initialization steps that are necessary in order to boot to an Intel Architecture (IA) platform.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. www.intel.com/embedded/edc. §

Contents



Business Challenge (or Background)

If a developer wants to write their own BIOS on IA architecture, then they have to gather the appropriate documents (which aren't always known) and guess the order that the items listed must be done in.

There currently isn't one document that describes all the items that need to be done in one place, nor is the order of initialization described anywhere.

There are also many legacy devices that must be initialized and finding documentation on them is challenging.

This document is an attempt to document the order, the minimum steps required, and generate a central repository of the various documents that contain the technical details of each technology / component of a typical platform.

Solution (Pilot Study or Proof of Concept)

There are two approaches that are typically taken by Intel when a new platform is developed and a boot loader solution is required.

- A BIOS is employed
- A custom boot loader is developed

The driving factor for the decision between the two is typically based on the features that are desired. BIOS has all the features and configurability available, whereas, a custom boot loader may work only for a specific board with specific hardware.

Initializing an Intel Architecture Platform from Reset

The bare minimum firmware requirements for making an IA platform operational and booting an OS are presented here in an order recommended by the authors. There may be design-based or segment-based requirements which would add/delete/re-order many of the items presented in this paper. However, for the vast majority of system designs, these steps in this order are sufficient.

Power-Up (Reset Vector) Handling

When an IA bootstrap processor (BSP) powers on, the first address that is fetched and executed is at physical address 0xFFFFFFF0, also known as the reset vector. This accesses the ROM / Flash device at the top of the ROM – 0x10. The boot loader must always contain a jump to the initialization code in these top 16 bytes.

Mode Selection

The processor must be placed into one of the following modes:

- Real Mode
- Flat Protected Mode
- Segmented Protected Mode – Not Recommended for Firmware

Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A section titled "Mode Switching" for more details.

Real Mode

Real Mode is 16-bit code with 16-bit registers. The physical address is calculated by SS << 4 + IP. Real Mode only allows accessing 1MB of memory.

In Real Mode, interrupt handling is through the Interrupt Vector Table (IVT).

For supporting legacy Operating Systems, some form of Real Mode code must be present during system run-time to handle requests from the Operating System. Discussion of these services is out of the scope of this paper.

Flat Protected Mode

Flat Protected Mode (Flat Mode) is 32-bit code where the physical addresses map one to one to the logical addresses.

The Interrupt Descriptor Table (IDT) is used for interrupt handling. For more information on the IDT, refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual.

This is the recommended mode for Firmware.

Segmented Protected Mode

Segmented Protected Mode is typically used by Operating Systems in conjunction with paging mode. It is not practical for use by Firmware.

Initial Processor Mode

When the processor is first powered-on, it will be in a special mode similar to Real Mode, but with the top 12 address lines being asserted high, allowing boot code to be accessed directly from NVRAM (physical address 0xFFFxxxxx). Upon execution of the first long jump, these 12 address lines will be driven according to instructions by firmware. If one of the Protected Modes is not entered before the first long jump, the processor will enter Real Mode, with only 1MB of addressability. In order for Real Mode to work without memory, the chipset needs to be able to alias memory below 1MB to just below 4GB, to continue to access NVRAM. Some chipsets do not have this aliasing and a forcible switch to a normal operating mode will be required before performing the first long jump.

Preparation for Memory Initialization

The following code is executed from the ROM / Flash since memory is not available yet. The least amount of code in this section the better, since executing from ROM is slow.

The following are steps that are taken to get the system ready to initialize memory:

- Processor Microcode Update
- Processor Initialization
- Chipset Initialization

Processor Microcode Update

Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A section titled "Microcode Update Facilities" for details on applying a processor microcode update.

Processor Initialization

Refer to the applicable processor BIOS/Firmware Writer's Guide for details on things that must be initialized prior to memory initialization, as well as, refer to the chipset Memory initialization Reference Code (MRC) documentation for any processor modes that must be initialized prior to calling the MRC.

The following are all different requirements that may be applicable:

- Non-Evict Mode (NEM)
- SRAM
- Register Storage
- Stack-less Setup

Chipset Initialization

Refer to the Chipsets BIOS Writer's Guides (BWG) for details on all initialization steps that must be performed prior to memory initialization.

In general, all Base Address Registers (BARs) should be initialized prior to memory initialization.

On most Integrated Controller Hubs / Integrated Output Hubs (ICHs / IOHs), there is a Watch Dog Timer (WDT) that must be disabled prior to the execution of memory initialization. If the WDT is not disabled, then a random reset can occur.

Memory Initialization

Technical Resources

Memory initialization for Intel platforms differs widely from chipset to chipset. The details about how to initialize the memory subsystem is considered restricted collateral. It comes in two forms:

- Documentation (Chipset BIOS Writer's Guide)
- Memory initialization Reference Code (MRC)

The BIOS/Firmware Writer's Guide documents the minimal steps required to initialize the memory subsystem as well as the boundaries/restrictions of the subsystem. This is useful for vendors that choose to write the memory initialization code themselves.

The MRC is distributed in different forms (depending on the owning Intel division and the platform). Each form that may be available is a fully-functioning implementation of the algorithm in the BIOS/Firmware Writer's Guide. The MRC may be ported into any firmware stack with a minimal amount of effort. Since the MRC supports all technologies and configurations allowed on a platform, it is possible to trim down the MRC to fit a vendor's design in a minimal amount of code space. This is the vendor's responsibility and not directly supported by Intel.

MRC Dependencies

In order to use MRC distributed by Intel, care needs to be taken to provide the appropriate operating environment for the code. The operating environment requirements may include (but may not be limited to):

- Processor Operating Mode
- Cache Configuration
- Memory Geometry Information

MRC may be written to run in 16-bit Real Mode, 32-bit Flat Mode, or (unlikely) a 32-bit Segmented Mode. If the adopted code does not have the appropriate operating environment, performance cannot be guaranteed.

With the higher DRAM speeds and the need for RCOMP and DLL calibration, typical chipsets do not have enough scratchpad space in the register sets. Therefore, it is common for MRC to have cache configuration requirements, such that the MRC has a "chalkboard" to write on during initialization.

PC-based memory configurations are based on removable memory modules (DIMMs). These configurations are dynamically detectable through tiny EPROM chips on the DIMMs. These chips contain specification-defined information about the capability of the DRAM configuration on the DIMM (Serial Presence Detect Data, or SPD data), and is readable through an I2C interface. For chipsets that are intended to be used in PC-based systems and support DIMMs, the MRC will usually have native SPD detection support included. For non-PC-based systems, it may not be present. In these configurations, it is required to hardcode the memory configuration or provide access to the memory geometry information through any vendor-defined mechanism. For memory-down designs, it may be necessary to generate several bytes of SPD data based on the DRAM datasheets and the schematics for the platform, and provide that to the MRC.

All of these dependencies should be documented along with the code that comprises the MRC.

Post Memory Initialization

There are certain things that must be done after MRC but before jumping and executing from memory. The following is a list of things that should be performed:

- Memory Test
- Shadow Firmware
- Memory Transaction Re-Direction (PAMs)
- Stack Setup

- Transfer to DRAM

Memory Test

At the end of memory initialization is the perfect time to attempt to perform some kind of memory integrity test. Some releases of MRC contain a memory test, and some do not. BIOS vendors typically provide some kind of memory test on a cold boot as well. Writing custom firmware will require the authors to choose a balance between thoroughness and speed, as highly embedded/mobile devices require extremely fast boot times.

Why is this the best time to perform memory tests? Simply because memory errors manifest themselves in very random ways, sometimes very inconsistently. Not checking memory integrity immediately after initializing memory increases complexity of firmware debug immensely, as any of the memory manipulation steps or firmware execution from DRAM is suspect to corruption.

There are several types of memory tests that are done in industry. Typically simple memory tests are performed in BIOS. Simple because it decreases code size/complexity and is easy to implement in assembly language to be executed out of NVRAM. Server-based platforms may implement much more thorough and complex memory tests. This paper does not promote any algorithm in specific.

Firmware Shadow

The concept of shadowing is simple: take code from slow non-volatile storage and copy it to DRAM. Why? Simply put, executing from DRAM is much faster. Cache misses that cause frequent fetches to non-volatile storage slows systems down greatly. Therefore it is recommended that system firmware is copied from non-volatile storage to DRAM as soon as possible after memory is initialized. Then, the non-volatile latency hit is only felt once.

For PC-based systems, the shadow must be from non-volatile storage to somewhere in the upper part of the lower 1MB of system memory. For non-PC systems, the end location is arbitrary, and up to the vendor and the requirements of the overlying software stack. For most applications, it's recommended to keep firmware code below 1MB.

Memory Transaction Re-Direction

Intel chipsets usually come with memory aliasing capabilities that allow reads and writes to sections of memory below 1MB to be either routed to/from DRAM or non-volatile storage located just under 4GB. The registers that control this aliasing are typically referred to as PAMs (Programmable Attribute Maps). Manipulation of these registers may be required before, during and after firmware shadowing. The control over the redirection of memory access varies from chipset to chipset. For example, some chipsets allow control over reads and writes, while others only allow control over reads. Consult the chipset datasheet for details on the memory redirection feature controls applicable to the target platform.

Stack Setup

The stack must be setup before jumping into memory. A memory location must be chosen for stack space. The stack will count down so the top of the stack must be entered and enough memory must be allocated for the maximum stack.

If the system is in real mode, then SS:SP must be set with the appropriate values. If Protected Flat Mode is used, then SS:ESP must be set to the correct memory location.

Transfer to DRAM

This is where the code makes the jump into memory. As mentioned before, if a memory test has not been performed up until this point, the jump could very well be to garbage. System failures indicated by a POST code between "end of memory initialization" and the first following POST code, almost always indicates a catastrophic memory initialization problem.

Miscellaneous Platform Enabling

Miscellaneous things in the system must be configured in the boot loader for proper operation. The only way to know what has to be programmed is to review the schematics. The following things are typically required to be programmed, but it is platform dependent:

- Clock Chip programming
- GPIO configuration – Refer to the Chipset BIOS Writers Guide for more details.

Interrupt Enabling

IA has several different methods of interrupt handling. The following or a combination of the following can be used to handle interrupts:

- Programmable Interrupt Controller (PIC) or 8259
- Local Advanced Programmable Interrupt Controller (APIC)
- Input / Output Advanced Programmable Interrupt Controller (IOxAPIC)
- Messaged Signaled Interrupt (MSI)

Programmable Interrupt Controller (PIC)

The PIC contains two cascaded 8259s with fifteen available IRQs. IRQ2 is not available since it is used to connect the 8259s.

Refer to the Chipset BIOS Writers Guide for more information on initializing the PIC.

Local Advanced Programmable Interrupt Controller (LAPIC)

The local APIC is contained inside the processor and controls the interrupt delivery to the processor. Each local APIC contains is own set of associated registers as well as a Local Vector Table (LVT). The LVT specifies the manner in which the interrupts are delivered to each processor core.

Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual for more information on initializing the local APIC.

I/O Advanced Programmable Interrupt Controller (IOxAPIC)

The IOxAPIC is contained in the ICH / IOH and expands the number of IRQs available to 24. Each IRQ has an associated redirection table entry that can be enabled / disabled and selects the IDT vector for the associated IRQ. This mode is only available when running in protected mode.

Refer to the Chipset BIOS Writers Guide for more information on initializing the IOxPIC.

Message Signaled Interrupt (MSI)

The boot loader does not typically use MSI for interrupt handling.

Processor Interrupt Modes

PIC Mode

When the PIC is the only interrupt device enabled, it is referred to as PIC Mode. This is the simplest mode where the PIC handles all the interrupts. All APIC components are bypassed and the system operates in single-thread mode using LINT0.

Virtual Wire Mode

Virtual Wire Mode uses the one of the APICs to create a virtual wire and operates the same as PIC mode. Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A for more details.

Interrupt Vector Table (IVT)

The IVT is the Interrupt Vector Table located at memory location 0p and containing 256 interrupt vectors. The IVT is used in real mode. Each vector address is 32 bits and consists of the CS:IP for the interrupt vector. Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A section titled "Exception and Interrupt Reference" for a list of Real Mode interrupts / exceptions.

Interrupt Descriptor Table (IDT)

The IDT is the Interrupt Descriptor Table and contains the exceptions / interrupts in Protected Mode. There are also 256 interrupt vectors and the exceptions / interrupts are defined in the same locations as the IVT. Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A for a detailed description of the IDT.

Exceptions

Exceptions are routines that run to handle error conditions. Examples are page fault, general protection fault, etc. At a minimum placeholders (dummy functions) should be used for each exception handler. Otherwise the system could exhibit unwanted behavior if an exception is encountered that isn't handled.

Real Mode Interrupt Service Routines (ISRs)

Real mode ISRs are used to communicate information between the boot loader and the OS. For example INT10h is used for video services such as changing video modes, resolution, etc.

There are some legacy programs and drivers that assume these real mode ISRs are available and directly call the INT routine.

Timers

Programmable Interrupt Timer (PIT)

The PIT (8254) resides in the IOH / ICH and contains the system timer also referred to as IRQ0. Refer to Chipset Datasheet for more details.

High Precision Event Timer (HPET)

HPET resides in the IOH / ICH and contains three timers. Typically the boot loader does not need to do any initialization of HPET and the functionality is used only by the OS.

Refer to Chipset BIOS Writers Guide for more details.

Real Time Clock (RTC)

The RTC resides in the IOH / ICH and contains the system time (seconds / minutes / hours / etc). These values are contained in CMOS which is explained later in the document. The RTC also contains a timer that can be utilized by Firmware.

Refer to the appropriate Chipset Datasheet for more details.

System Management TCO Timer

The TCO timers reside in the IOH / ICH and contain the Watch Dog Timer (WDT). The WDT can be used to detect system hangs and will reset the system.

***Note*:** It is important to note that for debugging any type of firmware on IA chipsets that implement a TCO Watch Dog Timer that it should be disabled by firmware as soon as possible coming out of reset. Halting system for debug prior to disabling this Watch Dog Timer on chipsets that power-on with this timer enabled will result in system resets, which doesn't allow firmware debug. The OS will re-enable the Watch Dog if it so desires. Consult the chipset datasheet for details on the specific implementation of the TCO Watch Dog Timer.

Refer to the Chipset BIOS Writer's Guide for more details.

Local APIC (LAPIC) Timer

The Local APIC contains a timer that can be used by Firmware. Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A for a detailed description of the Local APIC timer.

Memory Caching Control

Memory regions that must have different caching behaviors applied will vary from design to design. In the absence of detailed caching requirements for a platform, the following guidelines provide a "safe" caching environment for typical systems:

- Default Cache Rule – Uncached.
- 00000000-0009FFFF – Write Back.

- 000A0000-000BFFFF – Write Combined or Uncached.
- 000C0000-000FFFFF – Write Back or Write Protected.
- 00100000-TopOfMemory – Write Back.
- TSEG – Uncached.
- Graphics Stolen Memory – Write Combined or Uncached.
- Hardware Memory-Mapped I/O (e.g. PCI devices) – Uncached.

The Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A section on "Memory Cache Control" contains all the details on configuring caching for all memory regions.

See the appropriate Chipset BIOS Writer's Guide for caching control guidelines specific to the chipset.

Processor Discovery and Initialization

CPUID – Threads and Cores

Since Intel processors are packaged in various configurations, there are different terms that must be understood when considering processor initialization:

- Thread – A logical processor that shares resources with another logical processor in the same physical package.
- Core – A processor that coexists with another processor in the same physical package that does not share any resources with other processors.
- Package – A "chip" that contains any number of cores and threads.

Threads and Cores on the same package are detectable by executing the CPUID instruction. See the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A for details on the information available with the CPUID instruction on various processor families.

Detection of additional packages must be done "blindly". If a design must accommodate more than one physical package, the BSP needs to wait a certain amount of time for all potential APs in the system to "log in". Once a timeout occurs or the maximum expected number of processors "log in", it can be assumed that there are no more processors in the system.

Startup Inter-Processor Interrupt (SIPI)

In order to wake up secondary threads or cores, the BSP sends a SIPI to each thread and core. This SIPI is sent by using the BSP's LAPIC, indicating the physical address that the Application Processor (AP) should start executing from. This address must be below 1MB of memory and be aligned on a 4KB boundary.

AP Wakeup State

Upon receipt of the SIPI, the AP will start executing the code pointed to by the SIPI message. As opposed to the BSP, when the AP starts code execution it is in Real Mode. This requires that the location of the code that the AP starts executing is located below 1MB.

Wakeup Vector Alignment

The starting execution point of the AP has another architectural restriction that is very important and is commonly forgotten. The entry point to the AP initialization code must be aligned on a 4KB boundary. Refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A section titled "MP Initialization Protocol Algorithm for Intel Xeon Processors."

The Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A section labeled "Typical AP Initialization Sequence" illustrates what is typically done in by the APs after receiving the SIPI.

Caching Considerations

Because of the different types of processor combinations and different attributes of shared processing registers between threads, care must be taken to ensure that the caching layout of all processors in the entire system remain consistent such that there are no caching conflicts.

The Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A contains a section labeled "MTRR Considerations in MP Systems" that outlines a safe mechanism for changing the cache configuration in all systems that contain more than one processor. It is recommended that this be used for any system with more than one processor present.

AP Idle State

Behavior of APs during firmware initialization is dependent on the firmware implementation, but is most commonly restricted to short durations of initialization followed by entering a halt state with a HLT instruction, awaiting direction from the BSP for another operation.

Once the firmware is ready to attempt to boot an OS, all AP processors must be placed back in their power-on state ("Wait-for-SIPI"), which can be accomplished by the BSP sending an INIT ASSERT IPI followed by an INIT DEASSERT IPI to all APs in the system (all except self). See the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A for details on the INIT IPI, and the MultiProcessor Specification 1.4 for details on BIOS AP requirements.

I/O Devices

Refer to the board schematics to determine which IO devices are in the system. Typically a system will contain one or more of the following:

Embedded Controller (EC)

An Embedded controller is typically used in mobile or low power systems. The EC contains separate FW that controls the power management functions for the system as well as PS2 keyboard functionality. Refer to the specific EC data sheet for more details.

Super IO (SIO)

An SIO typically controls the PS2, serial, parallel, etc interfaces. Most systems still support some of the legacy interfaces rather than implementing a legacy free system. Refer to the specific SIO datasheet for details on programming information.

Legacy Free Systems

Legacy free systems use USB as the input device. If pre-OS keyboard support is required, then the legacy keyboard interfaces must be trapped. Refer to the IOH / ICH BIOS Specification for more details on legacy free systems.

Miscellaneous IO Devices

There may be other IO devices that require initialization by the boot loader. Refer to those device's datasheets for programming information.

PCI Device Discovery

Peripheral Connect Interface (PCI) device discovery is a generic term that refers to detecting which PCI compliant devices are in the system. The discovery process assigns the resources needed by each device including the following:

- IO space
- Memory Mapped IO (MMIO) space
- IRQ assignment
- Expansion ROM detection and execution

PCI device discovery applies to all the newer (non-legacy) interfaces such as PCI Express (PCIe), USB, SATA, SPI, etc devices. These newer interfaces all comply to the PCI specification.

Refer to PCI Specification for more details. A list of all the applicable specifications is in the References section.

Resource allocation. Resource allocation is the method of searching through the possible range of buses / devices / functions.

1 When a Vendor ID (VID) / Device ID (DID) is found that is not all 0xFFFFs, then resources are defined as follows:

a. Write all 0xFFFFFFFFs to all Base Address Registers (BAR)s. Read the BAR back. Bit 0 will give an indication as to whether this is an MMIO or IO request.

i. If MMIO is requested (bit 0 is a 0), then the value read back indicates how much memory is requested. Reserve the appropriate memory and write the address assigned into the BAR.

ii. IO – If IO is requested (bit 0 is a 1), then the value read back indicates how many bytes of IO are requested. Reserve that amount of IO and write the IO address assigned to the BAR.

b. IRQs

i. Refer to the Schematics to determine how the IRQ routing should occur for any PCI / PCIe slots or onboard devices.

ii. For chipset PCI devices, the routing may be pre-determined in the Chipset Datasheet. Some chipsets allow control of the internal routing as well. Refer to the Chipset Datasheet for more details.

iii. There are multiple things involved in IRQ routing when using the PIC. The devices are assigned to INTA – INTD which in turn gets assigned to PIRQA –

PIRQH which gets mapped to IRQs in chipset specific registers (0x60-0x63, 0x68-0x6B). Refer to the IOH / ICH BIOS Writers Guide and Chipset Datasheet for more details.

a. Read the Expansion ROM address register. If it is non-zero, then there is an Expansion ROM (also referred to as an Option ROM, OROM) that needs executed.

i. Video BIOS (vBIOS) is handled differently than typical OROMs. Typically it is executed prior to PCI enumeration in order to give certain display devices time to warm up. The vBIOS is 16-bit code that needs to run in real mode. The boot loader loads the vBIOS at C000:0 and jumps to address C000:3.

ii. Typical OROMs get loaded after the vBIOS starting at C800:0 and can continue up to E000:0. This limits the number and size of OROMs that can exist in a particular system.

iii. Some OROMs are also bootable such as Ethernet OROMs that have PXE support.

Booting a Legacy OS

Booting a legacy OS (non-EFI) consists of loading the first stage OS or OS boot loader in to memory location 7C0:0 and jumping to that location while the processor is in Real Mode.

Master Boot Record (MBR). The MBR is located on the first sector of a partitioned mass storage device. It contains the partition table as well as code. The Internet has details and can be reference at http://en.wikipedia.org/wiki/Master_boot_record.

OS Handover Requirements. Depending on the desired features that are enabled by the boot loader, there are different tables that the OS needs. The following is a list of those tables:

- Memory Map (INT15h / Function E820h)
- Programmable Interrupt Routing ($PIR)
- Multi-Processor Specification (_MP_)

- Simple Firmware Interface (SFI)
- Advanced Configuration and Power Interface (ACPI)

SFI and ACPI tables are only needed if those features are enabled by the boot loader and required by the OS.

The _MP_ table is needed if there is more than one Intel processing agent (thread or core) in the system. Details on the _MP_ table may be found in the MultiProcessor Specification.

The $PIR table and interrupt-based Memory Map are almost always needed. Details on the $PIR table may be found in the $PIR Specification. The Memory map is discussed in more detail in the following sections.

Memory Map

In addition to defining the caching behavior of different regions of memory for consumption by the OS, it is also firmware's responsibility to provide a "map" of the system memory to the OS so that it knows what regions are actually available for its consumption.

The most widely used mechanism for a boot loader or an OS to determine the system memory map is to use Real Mode interrupt service 15h, function E8h, sub-function 20h (INT15/E820), which firmware must implement (one example of this is detailed at http://www.uruk.org/orig-grub/mem64mb.html).

Region Types

There are several general types of memory regions that are described by this interface:

- Memory (1) – General DRAM available for OS consumption.
- Reserved (2) – DRAM address not for OS consumption.
- ACPI Reclaim (3) – Memory that contains all ACPI tables that firmware does not require run-time access to. See the applicable ACPI specification for details.
- ACPI NVS (4) – Memory that contains all ACPI tables that firmware requires run-time access to. See the applicable ACPI specification for details.
- ROM (5) – Memory that decodes to non-volatile storage (e.g. flash).
- IOAPIC (6) – Memory that is decoded by IOAPICs in the system (must also be uncached).
- LAPIC (7) – Memory that is decoded by Local APICs in the system (must also be uncached).

Region Locations

The following regions are typically reserved in a system memory map:

- 00000000-0009FFFF – Memory
- 000A0000-000FFFFF – Reserved
- 00100000-???????? – Memory (The ???????? indicates that the top of memory changes based on "reserved" items listed below and any other design-based reserved regions.)
- TSEG – Reserved
- Graphics Stolen Memory – Reserved
- FEC00000-FEC01000* – IOAPIC
- FEE00000-FEE01000* – LAPIC

See the applicable Chipset BIOS Writer's Guide for details on chipset-specific memory map requirements. See the appropriate ACPI specification for details on ACPI-related memory map requirements.

Non-Volatile (NV) Storage

There are two types of NV storage in typical IA systems; CMOS and Flash.

Complimentary Metal-Oxide Semiconductor (CMOS)

CMOS is part of the RTC and consists of two banks of 128 bytes each for a total of 256 bytes. The first 14 bytes are reserved for the RTC data. The rest of the bytes are available for any data that needs retained after the system is powered-off. The typical things that are stored are setup options to allow the user to select the primary boot device for example.

Refer to the Chipset Datasheet for more details on CMOS / RTC.

Non-Volatile Flash

Flash can also be used to store NV data as well. The boot loader must have routines to erase and write the data if Flash is to be used. Refer to the specific Flash datasheet for more details.

§

References

- Advanced Configuration and Power Interface Specification http://www.acpi.info/spec.htm
- Applicable Chipset Datasheets http://www.intel.com/products/embedded/chipsets.htm?iid=embed_portal+hdprod_chipsets#s1=all&s2=Intel%AE%20QM57%20Express%20Chipset&s3=all
- ATA/ATAPI Command Set Specifications http://www.t13.org/Documents/MinutesDefault.aspx?keyword=atapi
- Chipset BIOS Writers Guides. See your Intel account representative. If you don't have an Intel account representative click here to get help online (http://edc.intel.com/Get-Help/).
- Intel® 64 and IA-32 Architectures Software Developer's Manual http://developer.intel.com/products/processor/manuals/index.htm
- JEDEC DRAM Specifications http://www.jedec.org/
- Memory Reference Code. See your Intel account representative. If you don't have an Intel account representative click here to get help online (http://edc.intel.com/Get-Help/).
- Multiprocessor Specification 1.4 http://www.intel.com/design/pentium/datashts/24201606.pdf
- PCI Express® Base Specification http://www.pcisig.com/specifications/pciexpress/base2/
- PCI Firmware Specification http://www.pcisig.com/specifications/conventional/pci_firmware/
- PCI Local Bus Specification http://www.pcisig.com/specifications/conventional/
- $PIR Specification http://www.microsoft.com/whdc/archive/pciirq.mspx
- SD Specifications Part 1 Physical Layer Simplified Specification http://www.sdcard.org/developers/tech/sdcard/pls/
- SD Specifications Part 2A SD Host Controller Simplified Specification http://www.sdcard.org/developers/tech/host_controller/simple_spec/
- SD Specifications Part E1 SDIO Simplified Specification http://www.sdcard.org/developers/tech/sdio/sdio_spec/
- Serial ATA http://www.sata-io.org/
- Simple Firmware Interface Specification http://www.simplefirmware.org
- Universal Serial Bus Specification http://www.usb.org/developers/docs/

Conclusion

This document is an attempt to document the order, the minimum steps required, and generate a central repository of the various documents that contain the technical details of each technology / component of a typical platform.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. http://intel.com/embedded/edc.

Authors

Jenny M Pelner is a Firmware Architect with ECG at Intel Corporation.

James A Pelner is a Firmware Architect with the Ultra-Mobile Group (UMG) at Intel Corporation.

Acronyms

ACPI	Advanced Configuration and Power Interface
AP	Application Processor
BAR	Base Address Register
BSP	Boot Strap Processor
BWG	BIOS Writer's Guide
CMOS	Complimentary Metal-Oxide Semiconductor
DID	Device ID
EC	Embedded Controller
EDC	Embedded Design Center
GDT	Global Descriptor Table
GPIO	General Purpose IO
HPET	High Precision Event Timers
IA	Intel Architecture
ICH	Integrated Controller Hub
IDT	Interrupt Descriptor Table
IOH	Integrated Output Hub

ISR	Interrupt Service Routing
IVT	Interrupt Vector Table
LVT	Local Vector Table
MBR	Master Boot Record
MMIO	Memory Mapped IO
MP	Multi-Processor
MRC	Memory initialization Reference Code
MSI	Message Signaled Interrupt
NEM	Non-Evict Mode
NV	Non-Volatile
PAM	Programmable Attribute Maps
PIC	Programmable Interrupt Controller
PIT	Programmable Interval Timer
PCI	Peripheral Component Interface
PCIe	Peripheral Component Interface Express
RTC	Real Time Clock
SIO	Super IO
SIPI	Startup Inter-Processor Interrupt
SPI	Serial Peripheral Interface
SATA	Serial ATA
SFI	Simple Firmware Interface
USB	Universal Serial Bus
VID	Vendor ID
WDT	Watch Dog Timer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This paper is for informational purposes only. THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, Dialogic, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel EP80579 Integrated Processor, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2010 Intel Corporation. All rights reserved.

§