International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 03
March 2017

Unified Extensible Firmware Interface in Operating System

Omer Farooq Ahmed & Mustfa Mhamed Ali
Department of Computer Applied Technology, Hust
Email: omaraddeb@yahoo.com , M_muhammed72@yahoo.com

Abstract

The BIOS is very important part in the Motherboard because it's the first IC work and check all the internal devices and the external devices , but the problem is the BIOS is old technology its borne in 1975 and now we are in 2016 we can use the BIOS only anymore because the BIOS depend on 16 bit and now we use 32 and 64 bit operating system and the BIOS can't support more than 2.2 T bytes size of the hard disk and cant divide the hard disk more than 4 partition .So we need to use new technology it's called Unified Extensible Firmware Interface (UEFI) it's just firmware support the BIOS to make it work more efficient .

Our report highlight on the UEFI technology and explain the method and the structure of the work and the phases for the operating system booting in UEFI mode.

Keyword: BIOS, UEIF, ESP, GPT, MBR

Introduction

As we all know that BIOS is important part for accessing boot option. BIOS boots by reading the first sector on a hard disk and executing it; this boot sector in turn locates and runs additional code. The BIOS system is very limiting because of space constraints and because BIOS runs 16-bit code, whereas modern computers use 32-bit or 64-bit CPUs. By contrast, EFI (or UEFI, which is just EFI 2.x) boots by loading EFI program files (with .efi filename extensions) from a partition on the hard disk, known as the EFI System Partition (ESP). These EFI boot loader programs can take advantage of EFI boot services for things like reading files from the hard disk.

As a practical matter, if you're using an OS like Linux that has complex BIOS-mode boot loaders, EFI-mode booting is likely to be similar to BIOS-mode booting, since GRUB 2 (the most popular BIOS-mode boot loader for Linux) has been ported to work under EFI, and many Linux distributions install GRUB 2 by default on EFI systems. OTOH, you can replace or supplement GRUB 2 with other EFI boot loaders. Indeed, the Linux kernel itself can be an EFI boot loader; code was added to do this with the 3.3.0 kernel. Used in this way, the EFI itself loads and runs the Linux kernel, or you can use a third-party boot manager like rEFInd or gummiboot to let you select which OS or kernel to boot.

The BIOS and UEFI window Interface

The Booting Operation

So what is the real different in the "boot with BIOS" and "boot with UEFI"?

EFI *can be* faster, but that's not certain. The biggest speed difference is in hardware initialization early in the process. On my systems, this is a fraction of the total boot time, so a reduction in the hardware initialization time, while good, doesn't make all *that* much difference. It's not like I'm rebooting every ten minutes, after all.

UEFI supports a feature called *Secure Boot* that's intended, as the name suggests, to improve security. It does this by requiring a digital "signature" of boot loaders, which in turn

should require signing of kernels, and so on up the chain. This should make it harder for malware authors to insert their code into the pre-boot process, thus improving security. This sounds good, but it also complicates dual-boot configurations, since code like GRUB and the Linux kernel must be signed. The major Linux distribution vendors are working on ways to make these requirements less of a burden for average Linux users, and they've got some preliminary stuff ready. At the moment, though, disabling Secure Boot is the easiest way to deal with it. This is a practical concern mainly for brand-new computers that ship with Windows 8, since Microsoft is requiring Secure Boot be enabled to get Windows 8 certification. Some people confuse UEFI and Secure Boot (the latter is just one feature of the former), but it deserves mention as a difference between BIOS and UEFI because it's causing some problems for new Windows 8 computers. If you've got an older system or are comfortable enough with firmware setup utilities to disable Secure Boot, this need not be a real problem.

Microsoft ties the boot disk's partition table type to the firmware type (MBR to BIOS; GPT to UEFI). Because MBR tops out at 2TiB (assuming standard sector sizes), this means that UEFI is a practical necessity to boot Windows on over-2TiB disks. You can still use such big disks as data disks under Windows, though, and you can boot some non-Microsoft OSes (such as Linux and FreeBSD) on big disks using GPT under BIOS.

As a practical matter if you're concerned about or interested in UEFI, the biggest issue is simply that UEFI is new enough that support for it is a bit spotty, particularly in some older and more exotic OSes. UEFI itself is new enough that most of its implementations are buggy, and those that aren't vary enough amongst themselves that it can be hard to describe things generally. Thus, using UEFI can be a challenge. OTOH, UEFI *is* the future. It's got some modest advantages, some of which will become more important in time (such as the 2TiB boot disk limit of Windows). Switching to a UEFI boot will change a few details of the boot process, but your overall computing experience won't change all that much once you overcome any boot issues you may encounter.

Limitations of legacy BIOS

Over the years, many new configuration and power management technologies were

integrated into BIOS implementations as well as support for many generations of architecture hardware. However certain limitations of BIOS implementations such as 16-bit addressing mode, 1 MB addressable space, PC AT hardware dependencies and upper memory block (UMB) dependencies persisted throughout the years. The industry also began to have need for methods to ensure quality of individual firmware modules as well as the ability to quickly integrate libraries of third party firmware modules into a single platform solution across multiple product lines. These inherent limitations and existing market demands opened the opportunity for a fresh BIOS architecture to be developed and introduced to the market. The UEFI specifications and resulting implementations have begun to effectively address these persisting market needs. One of the critical maintenance challenges for BIOS is that each implementation has tended to be highly customized for the specific motherboard on which it is deployed. Moving component modules across designs typically requires significant porting, integration, testing and debug work. This is one of the market challenges the UEFI architecture promises to address.

UEFI Firmware Phases

The boot flow for UEFI based firmware is conceptualized in the form of six philosophical phases. A brief overview is presented here. For complete information . The **Security (SEC) phase** is the first phase and is responsible for the following:

- Handling all platform restart events
- Creating a temporary memory store
- Serving as the root of trust in the system
- Passing handoff information to the PEI Foundation

The PEI phase will initially operate leveraging only on-processor resources, such as the processor cache as a call stack, to dispatch Pre-EFI Initialization Modules (PEIMs). These PEIMs are responsible for the following:

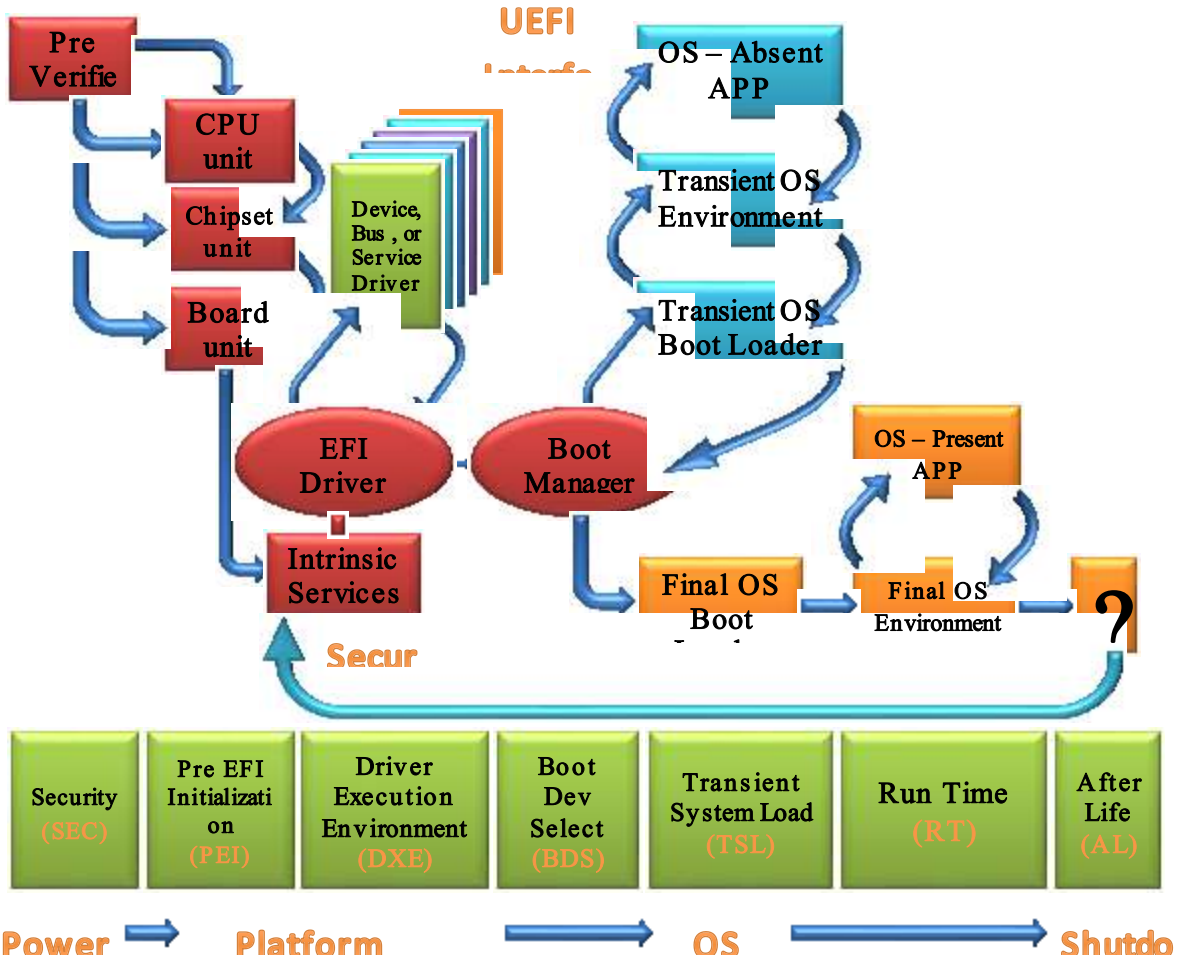
- Initialize some permanent memory complement
- Describe the memory in Hand-Off Blocks (HOBs)

- Describe the firmware volume locations in HOBs
- Pass control into the Driver Execution Environment (DXE) phase

The PEI phase is intended to be the thinnest amount of code to achieve the ends listed above. As such, any more sophisticated algorithms or processing should be deferred to the DXE phase of execution.

The state of the system at the end of the PEI phase is passed to the DXE phase through a list of data structures called **Hand-Off Blocks (HOBs)**. HOBs and the HOB list are created during the PEI phase. The HOB list is passed to and consumed by the DXE phase.

The **Driver Execution Environment (DXE)** phase is where most of the system initialization is performed. Pre-EFI Initialization (PEI), the phase prior to DXE, is responsible for initializing permanent memory in the platform so that the DXE phase can be loaded and executed.

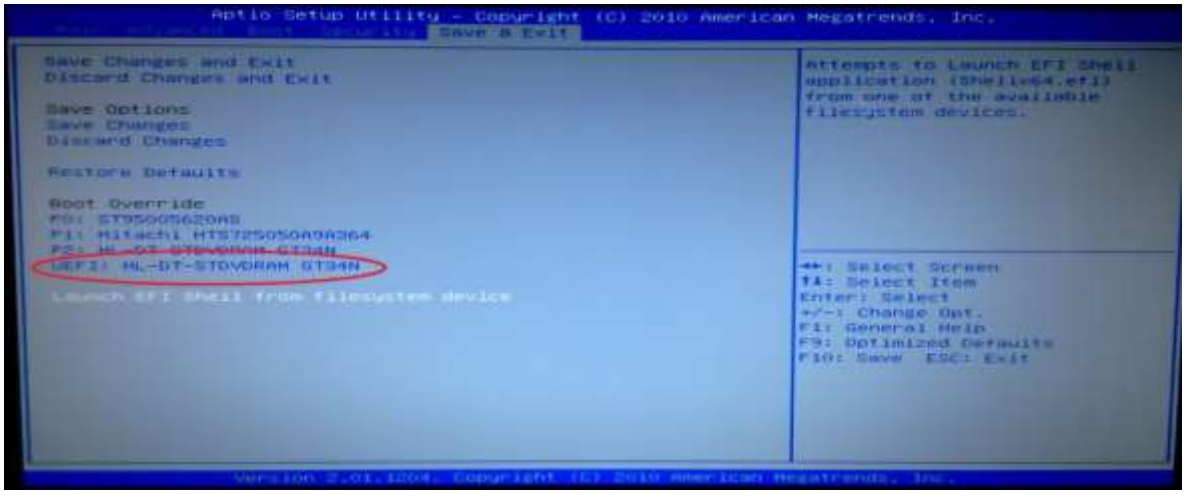


Booting by UEFI firmware phases

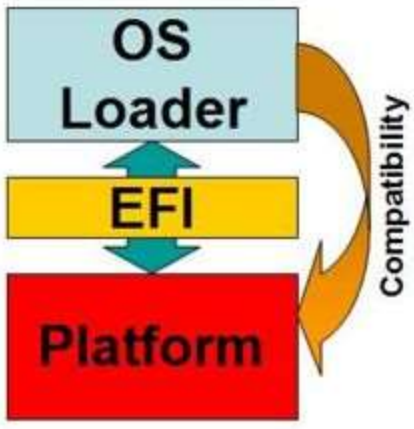
UEFI Work

UEFI is a cross-platform firmware interface that replaces the x86-specific firmware standard named BIOS. Many UEFI implementations also include a BIOS compatibility/"legacy" mode, to enable booting from MBRs and presenting a BIOS-like interface to OSes; however, this is not required by the standard.

If you have a UEFI-compatible motherboard that offers compatibility / legacy BIOS booting, then its firmware menu will provide choices such as setting a default boot mode or even booting single devices in either UEFI or BIOS mode:



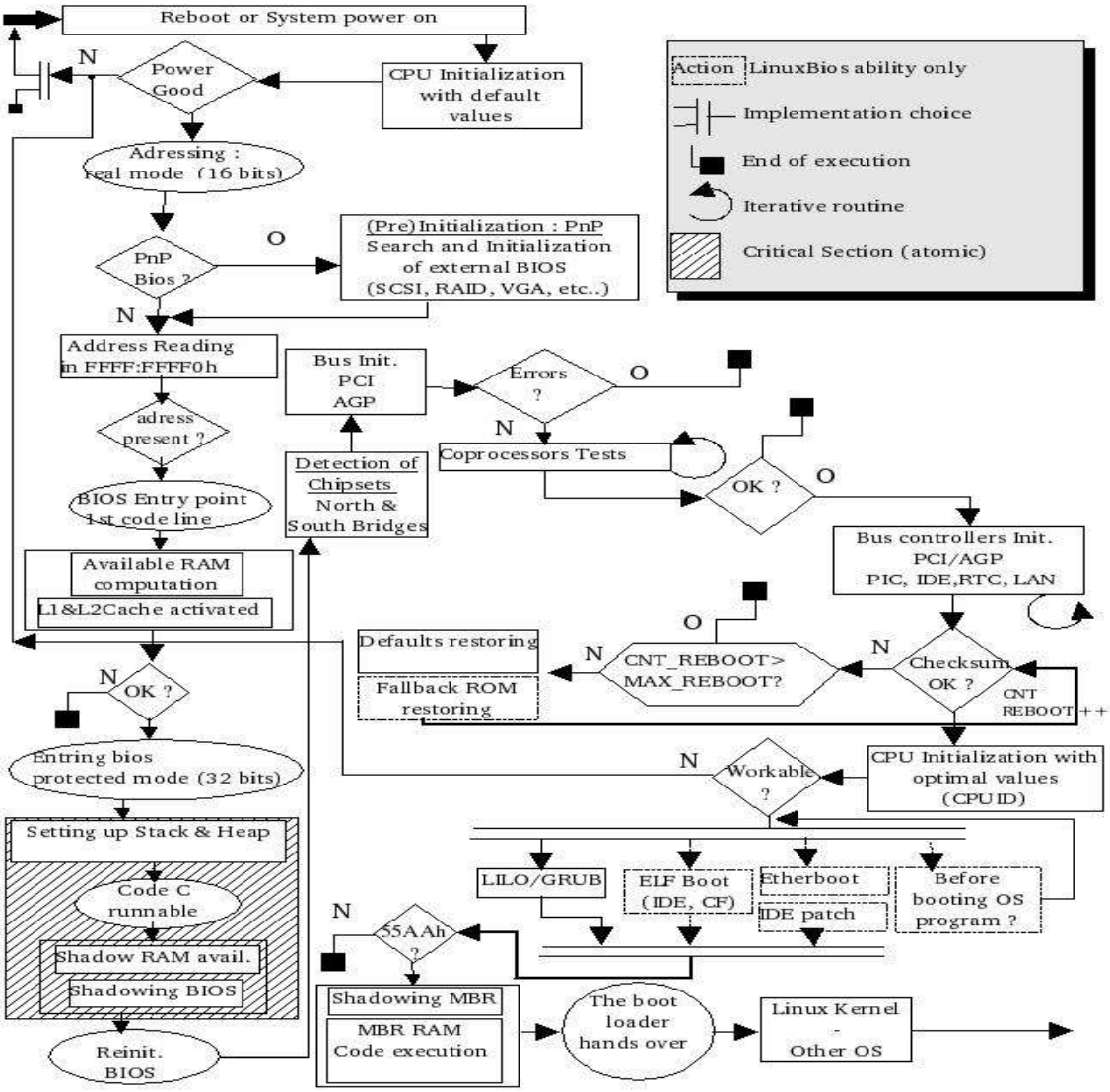
Otherwise, there might not be an easy way to tell, without e.g. probing the firmware using an OS.



There are many differences :

- UEFI defines a similar OS-firmware interface like BIOS but is not specific to any processor architecture. BIOS is specific to the Intel x86 processor architecture, as it relies on the 16-bit "real mode" interface supported by x86 processors.
- UEFI can be configured to expedite various parts of the booting process, for example, UEFI on Gigabyte GA-EP45-DS3 initializes in 11 seconds versus BIOS in 19 seconds.

UEFI mode may present different firmware/hardware features to the same installed OS than BIOS mode would.



The Booting structure

Conclusion

UEFI helps a characteristic called Secure Boot that is planned, as the name recommends, to enhance security. It does this by obliging a computerized "signature" of boot loaders, which thus might as well oblige marking of parts, et cetera up the chain. This might as well make it harder for malware creators to embed their code into the reboot process, accordingly enhancing security. This sounds great, however it likewise confounds double boot setups, since code like GRUB and the Linux part must be agreed upon. The significant Linux circulation merchants are taking a shot at approaches to make these prerequisites to a lesser degree a trouble for normal Linux clients, and they've got some preparatory stuff primed. At the minute, however, incapacitating Secure Boot is the most effortless approach to manage it. This is a useful concern chiefly for brand-new workstations that ship with Windows 8, since Microsoft is obliging Secure Boot be empowered to get Windows 8 certificate. Some individuals befuddle UEFI and Secure Boot (the last is only one characteristic of the previous), however it merits say as a distinction between BIOS and UEFI since its bringing on a few issues for new Windows 8 machines. Assuming that you've got a more senior framework or are agreeable enough with firmware setup utilities to debilitate Secure Boot, this requirement not be a genuine issue.

Microsoft ties the boot circle's segment table sort to the firmware sort (MBR to BIOS; GPT to UEFI). Since MBR tops out at 2tib (expecting standard part sizes), this implies that UEFI is a pragmatic need to boot Windows on over-2tib circles. You can even now utilize such huge circles as information plates under Windows, however, and you can boot some non-Microsoft Oses, (for example, Linux and Freebsd) on enormous plates utilizing GPT under BIOS.

Reference

- Building reliable SMM backdoor for UEFI based platforms
  http://blog.cr4.sh/2015/07/building-reliable-smm-backdoor-for-uefi.html
- Firmware papers and presentations timeline
  http://timeglider.com/timeline/5ca2daa6078caaf4

- Archive of OS X/iOS and firmware papers & presentations
  https://reverse.put.as/papers/
- Intel ATR - Black Hat 2015 / Def Con 23 - Firmware rootkit
  https://www.youtube.com/watch?v=sJnIiPN0104&app=desktop