# Advances in Platform Firmware Beyond BIOS and Across all Intel® Silicon

Vincent Zimmer
Staff Engineer
Enterprise Platforms Group
Intel Corporation

# Table of Contents

(Click on page number to jump to sections)

# Advances in Platform Firmware Beyond BIOS and Across all Intel® Silicon

Vincent Zimmer
Staff Engineer
Enterprise Platforms Group
Intel Corporation

## *Overview: A BIOS Alternative*

There has been rapid evolution of the personal computer platform since the 1980s. These advances have included order-of-magnitude increases in performance, ease-of-use, storage capacity, and connectivity. But there is one element of the PC that has not changed for the past 23 years, namely the BIOS (basic input/output system).

The Intel® Platform Innovation Framework for the Extensible Firmware Interface (EFI) (referred to as "the Framework") offers an opportunity to provide an alternative to BIOS that will allow for faster booting, manageability, and additional features.

## *Some Background*

The task of boot firmware (whether the BIOS or firmware based on the Framework) is to make a collection of hardware before the boot look like a complete system after the boot. For the foreseeable future, it is less expensive to build chips and boards that power up uninitialized so that, when reset, systems built around these components are in a generally primitive state.

These systems rely heavily on the boot firmware to prepare the system to boot the operating system, provide services to the operating system (particularly early in the boot process), and provide manageability data on the system.

## *Today's Issues*

To begin, let's review the role of the BIOS in today's system. The BIOS is stored in some nonvolatile storage on the platform and commences execution upon restart of the system. The BIOS is responsible for the initialization of the system. This is typically referred to as power-on self test (POST).

The POST for BIOS is typically written in some monolithic, statically linked 16-bit, real-mode assembly language and relegated to a small region of code space for execution. The assembly language construction and lack of consistent system services, such as a modern memory manager, coupled with the restricted execution space, impedes algorithm and feature development.

Beyond POST, there is the operating system (OS) invocation and ability to provide services to the OS. Herein, the operating system services are provided by 16-bit software interrupts. These software interrupts include Interrupt 13h for access to the Disk, Interrupt 10h for access to the Video, and Interrupt 16h for access to the Keyboard.

Operating system loads rely on the existence of these services. The limitations of these BIOS services include difficulty in extending new services, limited parameter passing through registers, and restrictions of real-mode. The Extensible Firmware Interface provides an opportunity to have a common operating system loader across different platform architectures, such as IA32 and Intel® Itanium® processor-based platforms. Today's legacy OS loader is relegated to the IA32 PC world.

## *New Technology*

The Framework architecture supports these requirements of initializing a system and providing services to the OS via a series of phases. Each phase is characterized by the resources available to it, the rules by which the code in the phase must abide, and the results of the phase. You can see the phases in **Figure 1**. It should be noted that each phase builds upon the other.

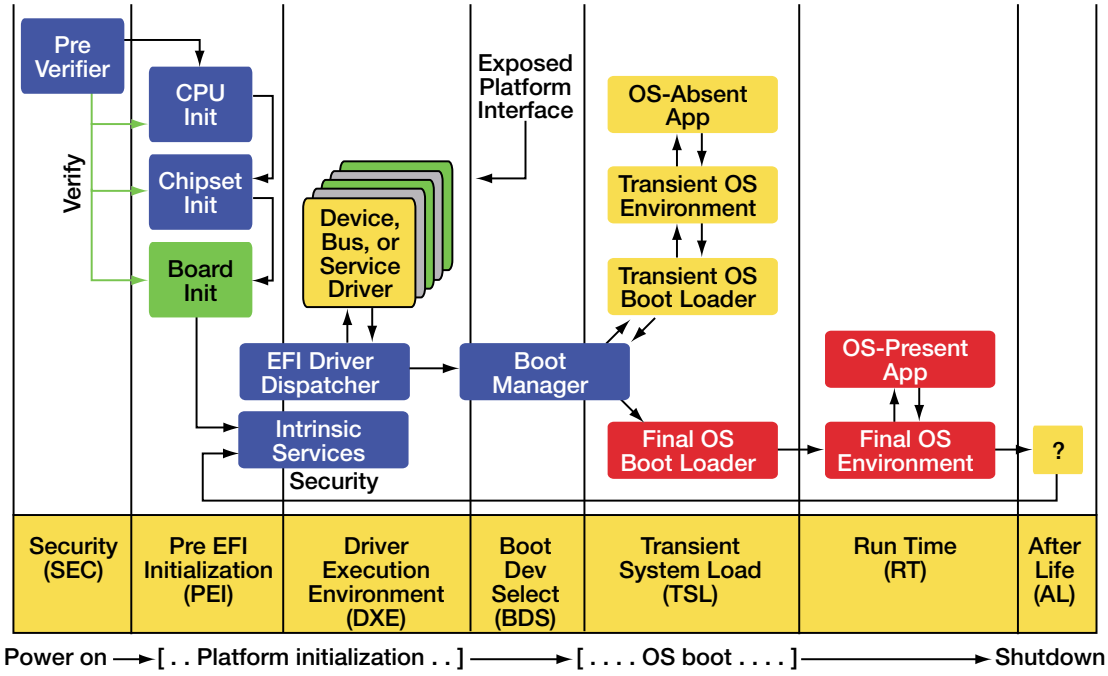


**Figure 1.** Firmware flow.

The infrastructure available in each phase is provided by the central framework, while the platform-specific features are implemented using intercommunicating modules. For the Pre-EFI (PEI) phase of execution, the modules are known as PEI modules (PEIMs). For the driver execution environment (DXE), the modules are alternately DXE or EFI drivers. The relationship of PEI and DXE can be seen in **Figure 2**. Nearly all of the foundation and modules are written in portable C code.

EFI drivers are somewhat analogous to device drivers in OSes. They provide the Framework architecture with its extensibility and allow it to do the following:

- Meet requirements from a range of platforms
- Incorporate new initiatives and fixes, as well as new hardware
- Support modular software architecture

EFI drivers can be developed at different times by different organizations. This introduces issues that monolithic, traditional BIOSes did not face. The Framework defines powerful solutions for sequencing EFI driver execution, abstracting EFI driver interfaces, and managing shared resources. The Framework and EFI drivers may optionally be cryptographically validated before use to ensure that a chain of trust exists from power-on until the OS boots and beyond.

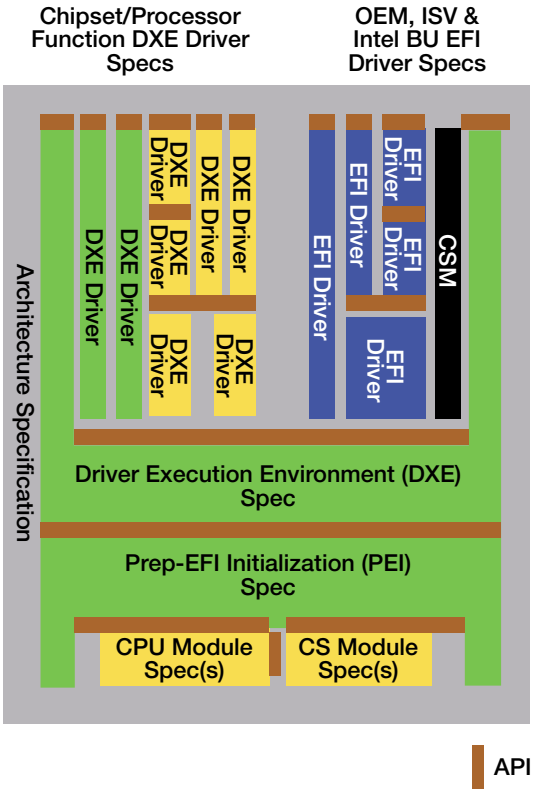


**Figure 2.** Framework binary modules.

The PEIMs and drivers can be deployed as separately constructed binary modules. The modules are collected in a storage abstraction referred to as a firmware volume. The firmware volume can be used to describe the platform nonvolatile storage, among other technologies.

The modules interface to the system and each other via callable interfaces that are named by globally unique identifiers (GUIDs). The GUID is a 128-bit value guaranteed to be statistically unique. This uniqueness allows for extensible service creation without limitation or collision among standard and platform-specific services.

The design-by-interface nature of EFI and the Framework decouples the software abstractions from the particular microarchitecture and platform topologies. As such, the Framework has been ported to IA32 desktop, server, embedded, and mobile systems. Also, the Framework has been deployed on Itanium processor-based servers down to Intel XScale® technology-based platforms. It should be noted that the Framework components were simply cross-compiled and a different complement of modules developed and/or reused to accommodate the Intel XScale technology-based platforms so as not to impede deployment.

The PEI and DXE phases provide for the platform initialization, such as memory initialization, I/O bus resource management, and I/O device discovery that occurred during legacy BIOS POST phase. The binary modules and dynamic service discovery allow for component reuse and platform feature distinction without having to relink all of the code, as with a monolithic BIOS.

In addition to the POST-like platform initialization, the Framework provides for the OS interface. The DXE phase germinates a set of EFI interfaces early in its evolution. This allows for a space-conscious implementation of an EFI-conformant set of interfaces. In addition, the Framework provides support for legacy OS interfaces via a set of drivers. This dual-OS interface support can be seen in **Figure 3**.

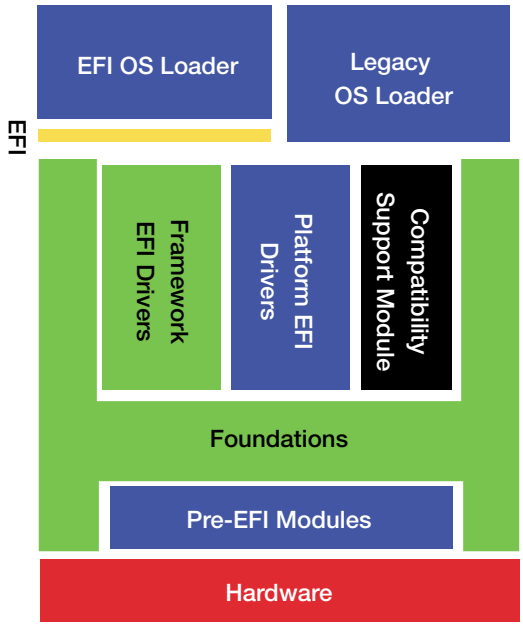


**Figure 3.** Software Layering.

## *Example System*

The Framework can be used to support a platform. **Figure 4** below shows an example system similar to today's PC platform. Notable components include silicon components (blue), memory technology (green), I/O buses (purple), and CPU technology (orange). For the deployment of a platform with the Framework, there are PEIMs that are responsible for initialization of the basic platform fabric, including but not limited to main memory initialization and basic CPU and chipset state.

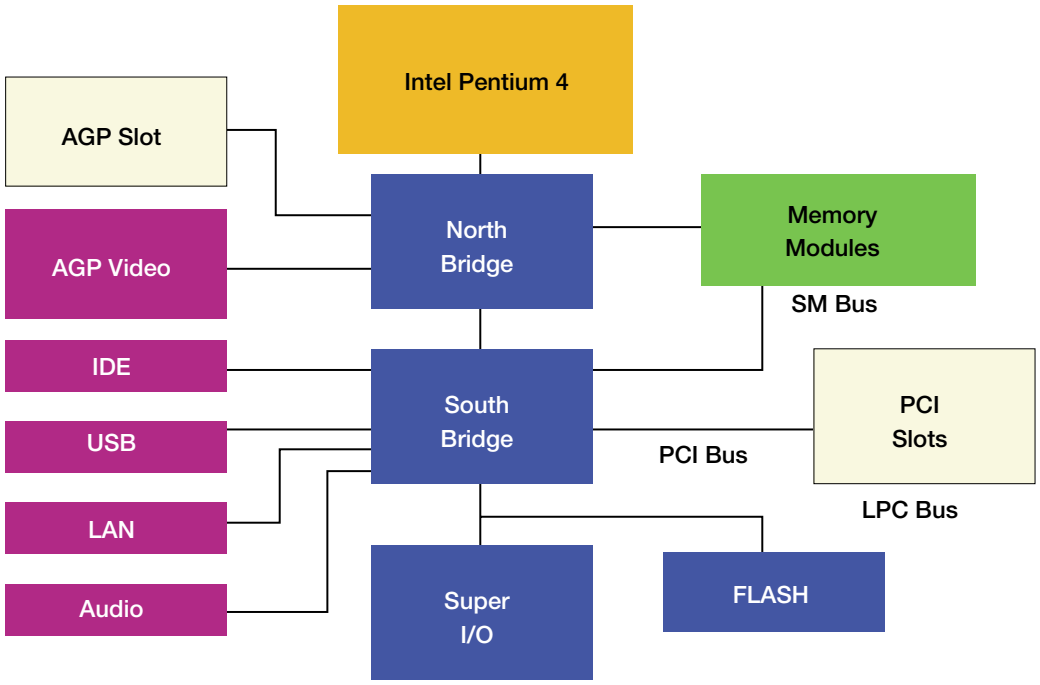


**Figure 4.** System block diagram

The PEI phase of system initialization proceeds by having the PEI foundation invoke a collection of PEIMs. These PEIMs include modules such as the North Bridge initialization PEIM, memory technology initiation PEIM, platform PEIM, and a CPU PEIM. This segregation allows for having a CPU PEIM, for example, that can be reused across a large class of platforms with different designs and chipsets.

Once the basic platform state has been provided, the DXE phase of execution commences. The bulk of the platform initialization occurs during the DXE phase. Here, there would be a driver for PCI bus resource allocation, a different driver to abstract fault-tolerant write access to the flash, console access (video, USB keyboard), and so on. It is also in DXE that the support for both legacy OS and EFI OS booting is provided.

Just as in the case of PEI, different permutations of the platform can be supported in DXE by the replacement of some sets of drivers. A different flash component or video device would entail only a single driver update, respectively, in each case. Also, a high-availability server versus a fixed-configuration embedded system may only differ in a single driver that describes the boot policy.

## Summary

The Intel Platform Innovation Framework for the Extensible Firmware Interface is a new product-strength implementation of EFI. It offers an alternative to BIOS that will enable faster booting, manageability, and additional features. The Framework is a set of robust architectural interfaces, implemented in C, which provides a way to build platform firmware via modular components. It has been designed to enable the BIOS industry and Intel customers to accelerate the evolution of innovative, differentiated, platform designs.

**Feedback**

Tell us what you think about this article.

## More Info

Learn more information about the Framework.
Learn more about the Extensible Firmware Interface.

## Author Bio

Vincent Zimmer has been with Intel seven years, and is a staff engineer in the Enterprise Platforms Group (EPG). He has over 12 years experience in embedded software development, and currently has over 100 U.S. patents pending or issued. Zimmer has a B.S.E.E. degree from Cornell University and an M.S.C.S. degree from the University of Washington, Seattle.

*—End of Technology@Intel Magazine Article—*