BEYOND BIOS: EXPLORING THE MANY DIMENSIONS OF THE UNIFIED EXTENSIBLE FIRMWARE INTERFACE

Contributors

Mark Doran
Intel Corporation

Vincent J. Zimmer
Intel Corporation

Michael A. Rothman
Intel Corporation

This article describes the basic capabilities of the specifications produced by the UEFI Forum as well as the history of how these standards evolved.

Introduction

The purpose of this article is to describe the basic capabilities of the UEFI Forum Specifications including the UEFI Specification version 2.3.1, the Platform Initialization Specification version 1.2, along with the structure and use of UDK2010, an open source implementation of the UEFI Forum Specifications. The history and evolution of these technologies provides some context for those descriptions.

The Chicken and the Egg

The very first effort that is considered a direct ancestor of UEFI technology had a very specific tactical goal. In the course of 1997 people at Intel were working on how to boot computers based on the prospective Itanium® Processor family. The original plan was to use the conventional BIOS code base for this job: while more or less everything else about the machines would be new—processors, chipsets, board designs, operating systems, and so on—it was felt that keeping stability in one element of the machine recipe a known quantity would be of some advantage. Without getting to specifics, this plan ultimately proved infeasible for technical and business reasons. This left the problem of how to boot an OS on these platforms, with something less than a year of time for resolution.

This challenge spawned the effort inside Intel that became known as the Intel Boot Initiative (IBI), specifically targeting development of a boot paradigm for Itanium Processor based machines. The IBI effort considered a set of alternatives, "make" versus "buy," and that included among others adoption of the IEEE Open Firmware standard, use of the ARC platform standard, and of course building a solution from scratch. The Open Firmware standard offered a good technical solution but fell short in terms of business infrastructure for deployment in the time available while the ARC platform standard ended up being too prescriptive on platform design. Similarly other "buy" alternatives offered no clear path to deployment in the time available. Thus the decision was taken to pursue in-house development of a new mechanism.

"A high-level C language interface between platform firmware and the OS loader seemed like a natural."

A high-level C language interface between platform firmware and the OS loader seemed like a natural for Itanium Processor machines given the complexity of low level programming and the desirability of having the OS know as little about the platform hardware specifics as possible in advance

of being able to load OS drivers. Having made that leap, it was a short hop from there to imagine a CPU-architecture–neutral API for firmware and OS communication for the boot process.

An Abstract Interface to Promote Innovation for OS and Firmware

The value of having such an interface and having it be broadly applicable to computers in general was driven home by experience on IA-32 platforms: there, the OS historically had hard-coded assumptions about the presence an operation of platform hardware devices and intimate familiarity with internals of many parts of system BIOS. All these factors on IA-32 discouraged change and made innovation tricky and relatively expensive. Obviously a successor technology that alleviates the need for the platform and OS to share intimate details for their respective implementation would provide an opportunity to decouple the rate of innovation for both the platform and the operating system.

This set of realizations led to the first big scope increase for the fledgling firmware interface. By taking on board the CPU-ISA–neutral approach, the scope of the program could increase to cover more of the Intel Processor family. Even in the late 1990s it was apparent that the conventional BIOS used on IA-32 computers was starting to become something of a drag on innovation. The designers of the new firmware interface thus turned their attention to including the ability to boot IA-32 processors as well as the original mission. It was around this time in late 1999 that the IBI program produced initial specifications for the new interface. The change in scope and the deliberate intention to foster a pro-innovation environment in the pre-OS space informed choice of the name for the new specification: the Extensible Firmware Interface (EFI)—neutral to any particular type of computer, deliberately describing only the interface (and explicitly not implementation of either producer [BIOS] or consumer [OS]), and calling out the idea that the interface would be a baseline for future additions.

"Even in the late 1990s it was apparent that the conventional BIOS used on IA-32 computers was starting to become something of a drag on innovation."

EFI was adopted for the very first generation of Itanium Processor based computers and has been the boot interface there ever since. The initial version of the EFI specification 1.02 was published by Intel in December 2000 covering the operational scope needed to transfer control from platform firmware to the operating system. From the outset Intel kept the barriers to adoption for EFI as low as practical with little if any licensing restrictions and royalty-free sample implementation code. The principle of low barrier to adoption remains central to the management of the technology right up to present day.

"EFI was adopted for the very first generation of Itanium Processor based computers and has been the boot interface there ever since."

To that initial publication the EFI Specification 1.10 was added in late 2002. This updated specification added a firmware driver model that addressed the problem of using add-in card devices in the boot process and providing code to operate those without requiring changes to the operating system boot loaders per device. In essence this provided a path to replace the fragile system of 16-bit option ROMs first advanced for ISA bus devices and later adopted for PCI boot devices as well.

"a group of industry stakeholders comprising BIOS vendors, OS vendors, system manufacturers, and silicon production companies agreed to form the Unified EFI Forum."

Industry Backing: Advent of the Unified EFI Forum

Adoption on the IA-32 family would follow gaining momentum slowly but by early 2005 business conditions and technical constraints made it clear that the conventional BIOS technology would eventually run out of steam. In recognition of the fact that becoming a critical piece of the infrastructure for delivering IA-32 platforms to market is a multilateral industry intercept, a group of industry stakeholders comprising BIOS vendors, OS vendors, system manufacturers, and silicon production companies agreed to form the Unified EFI Forum in mid-2005 and the long-term home and governance model for this technology.

Intel contributed the EFI Specification as a starting point for the new Forum's work and the founding Promoter members worked in a truly unified fashion to produce a specification with broad industry support and endorsement. This initial publication from the Forum, the UEFI Specification 2.0, was published in January 2006.

In parallel with work on the interface between firmware and operating system, the Forum agreed to take on work to standardize interfaces for the internal construction mechanisms within an implementation of the UEFI Specification. This work led to the publication of the Platform Initialization (PI) Specification 1.0 in October 2006. This five-volume set aims to make it possible for silicon component support firmware to work unmodified with firmware on platforms developed by a variety of system building companies, simplifying and shortening deployment work for new product generations. The latest version of the PI Specification is version 1.1 published in February 2008.

The Forum continued to build consensus around updates to the UEFI Specification, publishing version 2.1 in January 2007. Among other things this introduced infrastructure that results in more graphical, better localized user interfaces for the pre-OS space.

Version 2.2 came along in September 2008 introducing IPv6 support for networking and also improved platform security primitives. Version 2.2's reign as the latest/greatest was relatively short-lived, however, largely as a result of work in the implementation world behind the specification.

Open Source Firmware Implementation

Intel had initially made available open source sample implementations of the original EFI Specification. That work continued as the EFI Specification evolved into the UEFI Specification and also delivered an implementation of the PI Specifications. This implementation found a permanent home as the EFI Developers Kit open source project still housed at www.TianoCore.org. This is known as the EDK for historical reasons although today of course the implementation conforms to the UEFI Forum's Specifications in its EDK II (second generation) form.

"In addition to implementations of the various specifications, the Forum has also promoted the creation of test suites both for the UEFI Specification and for the PI Specification."

In addition to implementations of the various specifications, the Forum has also promoted the creation of test suites both for the UEFI Specification and for the PI Specification. These tests are designed to help developers build high

quality implementations of the specifications and are yet another example of the philosophy of making UEFI an easy technology to adopt in this case by making useful tools freely available for developers.

Completing the Specification Picture

As commonly happens with open source projects, interested parties come along and find new and interesting ways to use the code. In the case of the EDK, several companies found that the code was useful on ARM based platforms. Following successful ports to ARM platforms, it was proposed to add an ARM binding for the UEFI Specification. This was completed by the Forum in May of 2009 leading to the publication of the 2.3 version of the specification.

The 2.3 version of the Specification represents an interesting milestone for the community around Intel Architecture firmware. That specification represents the first point in time where all the interfaces for the boot process are written down in a formal document with industry-wide agreement on the content.

"With the state of the specifications now caught up to the present day platform design needs, attention is turning to driving technology forward to improve and expand capabilities in the pre-OS space."

Looking Forward

The most recent version of the UEFI Specification is now 2.3.1, which as the name suggests, is an incremental release based on the 2.3 version. The new areas refine support for scalable platform security solutions and help to support faster and more sophisticated look and feel for the boot process.

With the state of the specifications now caught up to the present day platform design needs, attention is turning to driving technology forward to improve and expand capabilities in the pre-OS space. One of the first such efforts, radical reduction in boot time, may seem counterintuitive in that frame—the innovation is in fact to do less not more in the pre-OS space. However, this clearly represents a step forward in terms of appeal to the market as a whole and it is equally something that depends in large part on the abstraction of firmware and OS implementation from each other that is integral to the UEFI design—each part of the implementation of boot, firmware platform component initialization, and operating system startup can be optimized to work best with each other, yielding significant improvements overall.

UEFI technology is already in widespread use, in everything from smart phones to printers, notebooks, servers, and even supercomputers. There are new devices and platform technologies in prospect that will benefit from easier enabling through UEFI shortening time to market. There are new types of platforms like system-on-chip starting to adopt UEFI technologies for infrastructure in new product categories. In short, UEFI technology is helping to power the leading edge of compute platform innovations backed by broad industry collaboration for deployment and support.

"UEFI technology is already in widespread use, in everything from smart phones to printers, notebooks, servers, and even supercomputers."

"UEFI is a pure interface specification that does not dictate how the platform firmware is built; the "how" is relegated to PI."

Where Does This Fit in the Ecosystem?

When we discuss UEFI, we need to emphasize that UEFI is a pure interface specification that does not dictate how the platform firmware is built; the "how" is relegated to PI. The consumers of UEFI include but are not limited to operating system loaders, installers, adapter ROMs from boot devices, pre-OS diagnostics, utilities, and OS runtimes (for the small set of UEFI runtime services). In general, though, UEFI is about *booting*, or passing control to a successive layer of control, namely an operating system loader, as shown in Figure 1. UEFI offers many interesting capabilities and can exist as a limited runtime for some application set, in lieu of loading a full, shrink-wrapped multi-address space operating system like Microsoft Windows*, Apple OS X*, HP-UX*, or Linux, but that is not the primary design goal.

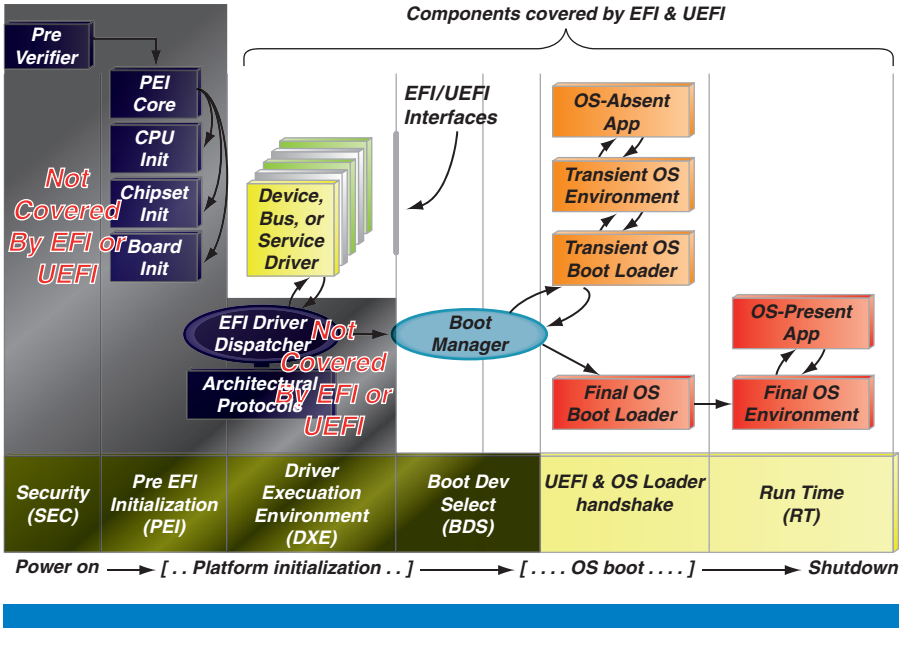

Figure 1: Where EFI and UEFI Fit into the Platform Boot Flow (Source: Intel Corporation, 2010)

PI, on the other hand, should be largely opaque to the pre-OS boot devices, operating systems, and their loaders since it covers many software aspects of platform construction that are irrelevant to those consumers. PI instead describes the phases of control from the platform reset and into the success phase of operation, including an environment compatible with UEFI, as shown in Figure 2. In fact, the PI DXE component is the preferred UEFI core implementation.

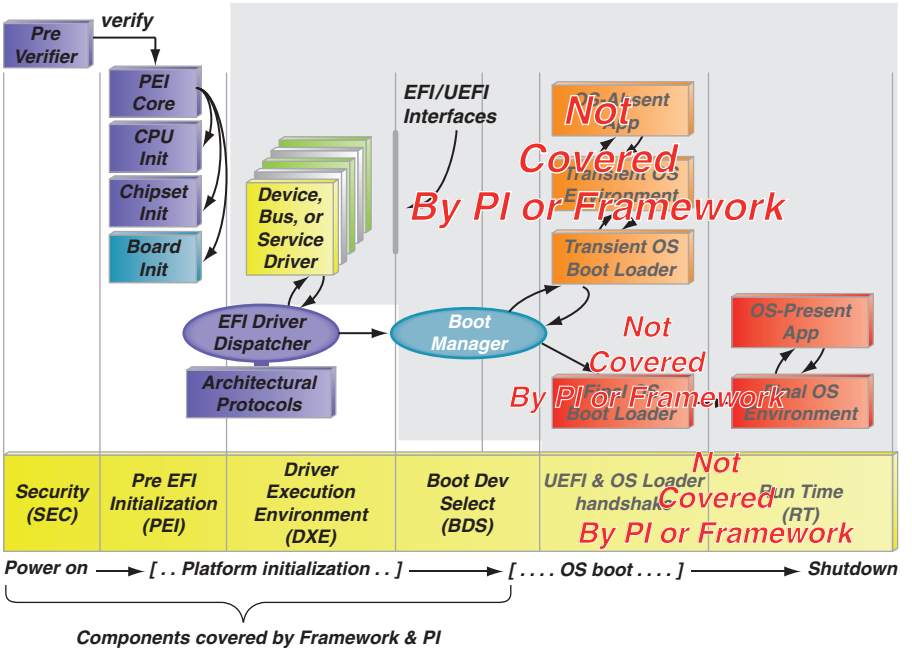

Figure 2: Where PI and Framework Fit into the Platform Boot Flow (Source: Intel Corporation, 2010)

Within the evolution of Framework to PI, some things were omitted from inclusion in the PI specifications. Specifically, the CSM specification abstracted booting on a PC/AT system. This requires an x86 processor, PC/AT hardware complex (for example, 8254, 8259, RTC). The CSM also inherited other conventional BIOS boot limitations, such as the 2.2-TB disk limit of Master Boot Record (MBR) partition tables. For a world of PI and UEFI, you get all of the x86 capabilities (IA-32 and x64, respectively), ARM*, Itanium®, and future CPU bindings. Also, via the polled driver model design, UEFI APIs, and the PI DXE architectural protocols, the platform and component hardware details are abstracted from all consumer software. Other minor omissions also include data hub support. The latter has been replaced by purpose-built infrastructure to fill the role of data hub in Framework-based implementations, such as SMBIOS table creation and agents to log report status code actions.

What has happened in PI beyond Framework, though, includes the addition of a multiprocessor protocol, Itanium E-SAL and MCA support, the above-listed report-status code listener and SMBIOS protocol, an ACPI editing protocol, and an SIO protocol. With Framework collateral that moved to PI, a significant update was made to the System Management Mode (SMM) protocol and infrastructure to abstract out various CPU and chipset implementations from the more generic components. On the DXE front,

"Some additions occurred in the PEI foundation for the latest evolution in buses, such as PCI Express."*

small cleanup was added in consideration of UEFI 2.3 incompatibility. Some additions occurred in the PEI foundation for the latest evolution in buses, such as PCI Express*. In all of these cases, the revisions of the SMM, PEI, and DXE service tables were adjusted to ease migration of any SMM drivers, DXE drivers, and PEI module (PEIM) sources to PI. In the case of the firmware file system and volumes, the headers were expanded to comprehend larger file and alternate file system encodings, respectively. Unlike the case for SMM drivers, PEIMs, and DXE drivers, these present a new binary encoding that isn't compatible with a pure Framework implementation.

The notable aspect of the PI is the participation of the various members of the UEFI Forum, which will be described below. These participants represent the consumers and producers of PI technology. The ultimate consumer of a PI component is the vendor shipping a system board, including multinational companies such as Apple, Dell, HP, IBM, Lenovo, and many others. The producers of PI components include generic infrastructure producers such as the independent BIOS vendors (IBVs) like AMI, Insyde, Phoenix, and others. And finally, the vendors producing chipsets, CPUs, and other hardware devices like AMD, ARM, and Intel would produce drivers for their respective hardware. The IBVs and the OEMs would use the silicon drivers, for example. If it were not for this business-to-business transaction, the discoverable binary interfaces and separate executable modules (such as PEIMs and DXE drivers) would not be of interest. This is especially true since publishing GUID-based APIs, marshalling interfaces, discovering and dispatching code, and so on take some overhead in system board ROM storage and boot time. Given that there's never enough ROM space, and also in light of the customer requirements for boot time such as the need to be "instantly on," this overhead must be balanced by the business value of PI module enabling. If only one vendor had access to all of the source and intellectual property to construct a platform, a statically bound implementation would be more efficient, for example. But in the twenty-first century with the various hardware and software participants in the computing industry, software technology such as PI is key to getting business done in light of the ever-shrinking resource and time-to-market constraints facing all of the UEFI forum members.

"Given that there's never enough ROM space, and also in light of the customer requirements for boot time such as the need to be "instantly on," this overhead must be balanced by the business value of PI module enabling."

"There is a large body of Framework-based source-code implementations, such as those derived or dependent upon EDK I (EFI Developer Kit version 1), which can be found on www.tianocore.org."

There is a large body of Framework-based source-code implementations, such as those derived or dependent upon EDK I (EFI Developer Kit version 1), which can be found on www.tianocore.org. These software artifacts can be recompiled into a UEFI 2.3, PI 1.2-compliant core, such as UDK2010 (the UEFI Developer Kit revision 2010), via the EDK Compatibility Package (ECP). For new development, though, the recommendation is to build native PI 1.2, UEFI 2.3 modules in the UDK2010 since these are the specifications against which long-term silicon enabling and operating system support will occur, respectively.

Terminology

The following list provides a quick overview of some of the terms that have existed in the industry associated with the BIOS standardization efforts.

- *UEFI Forum.* The industry body which produces UEFI, Platform Initialization (PI), and other specifications.
- *UEFI Specification.* The firmware-OS interface specification.
- *EDK.* The EFI Development Kit, an open sourced project that provides a basic implementation of UEFI, Framework, and other industry standards. It, is not however, a complete BIOS solution. An example of this can be found at www.tianocore.org.
- *UDK.* The UEFI Development Kit is the second generation of the EDK (EDK II), which has added a variety of codebase-related capabilities and enhancements. The inaugural UDK is UDK2010, with the number designating the instance of the release.
- *Framework.* A deprecated term for a set of specifications that define interfaces and how various platform components work together. What this term referred to is now effectively replaced by the PI specifications.
- *Tiano.* An obsolete codename for an Intel codebase that implemented the Framework specifications.

Managing the Specifications in UEFI

Regarding the UEFI Forum, there are various aspects to how it manages both the UEFI and PI specifications. Specifically, the UEFI forum is responsible for creating the UEFI and PI specifications.

"the UEFI forum is responsible for creating the UEFI and PI specifications."

When the UEFI Forum first formed, a variety of factors and steps were part of the creation process of the first specification:

- The UEFI forum stakeholders agree on EFI direction
- Industry commitment drives need for broader governance on specification
- Intel and Microsoft contribute seed material for updated specification
- EFI 1.10 components provide starting drafts
- Intel agrees to contribute EFI test suite

As this had established the framework of the specification material that was produced and that the industry used, the forum itself was formed.

The UEFI Forum was established as a Washington nonprofit corporation. It develops, promotes, and manages evolution of Unified EFI Specification and continues to drive low barrier for adoption.

The UEFI Forum has a form of tiered membership: Promoters, Contributors, and Adopters. More information on the membership tiers can be found at www.uefi.org. The Promoter members for the UEFI forum are AMD, AMI, Apple, Dell, HP, IBM, Insyde, Intel, Lenovo, Microsoft, and Phoenix.

"The UEFI Forum has several work groups."

The UEFI Forum has several work groups. Figure 3 illustrates the basic makeup of the forum and the corresponding roles.

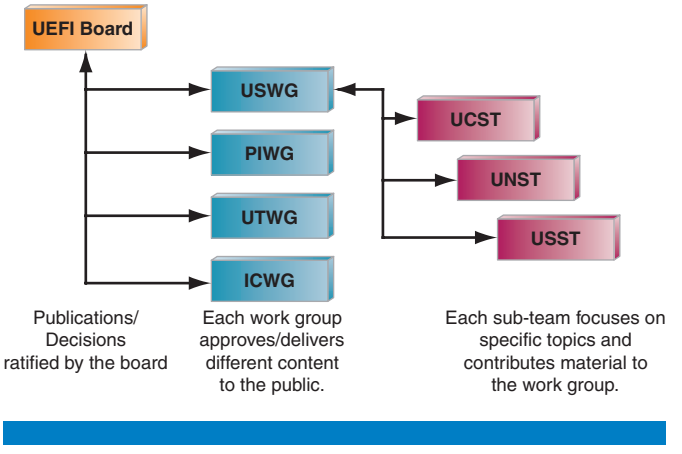

Figure 3: Forum Group Hierarchy (Source: Intel Corporation, 2011)

Sub-teams are created in the main owning workgroup when a topic of sufficient depth requires a lot of discussion with interested parties or experts in a particular domain. These teams are collaborations amongst many companies who are responsible for addressing the topic in question and bringing back to the workgroup either a response or material for purposes of inclusion in the main working specification. Some examples of sub-teams that have been created are as follows as of this writing:

- UCST – UEFI Configuration Sub-team. Chaired by Michael Rothman (Intel), this sub-team is responsible for all configuration related material and the team has been responsible for the creation of the UEFI configuration infrastructure commonly known as HII, which is in the UEFI Specification.
- UNST – UEFI Networking Sub-team. Chaired by Vincent Zimmer (Intel), this sub-team is responsible for all network related material and the team has been responsible for the update/inclusion of the network related material in the UEFI specification, most notably the IPv6 network infrastructure.
- USST – UEFI Security Sub-team. Chaired by Tim Lewis (Phoenix), this sub-team is responsible for all security related material and the team has been responsible for the added security infrastructure in the UEFI specification.

PIWG and USWG

"The Platform Initialization Working Group (PIWG) is the portion of the UEFI forum that defines the various specifications in the PI corpus."

The Platform Initialization Working Group (PIWG) is the portion of the UEFI forum that defines the various specifications in the PI corpus. The UEFI Specification Working Group (USWG) is the group that evolves the main UEFI specification. Figure 4 illustrates the layers of the platform and shows the scope for the USWG and PIWG, respectively.

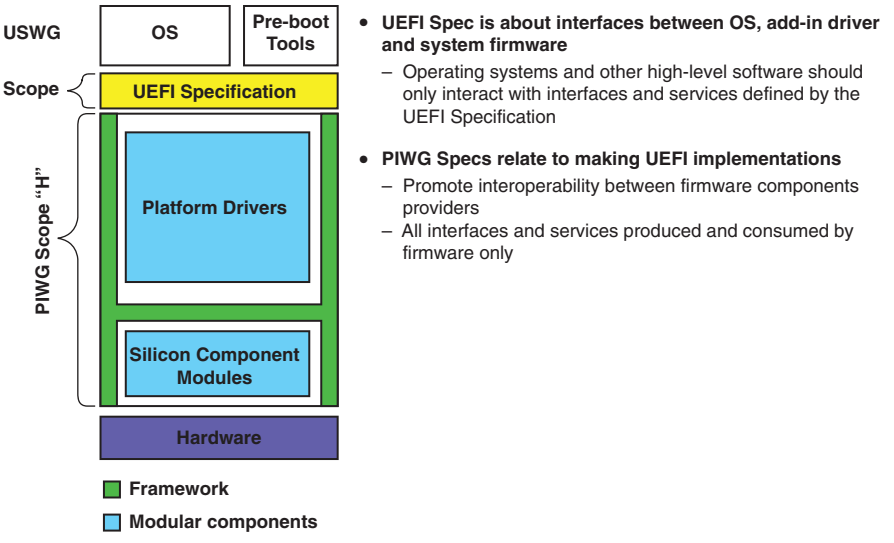

Figure 4: PI/UEFI Layering
(Source: Intel Corporation, 2011)

Figure 5 shows how the PI elements evolve into UEFI. The left half of the diagram with SEC, PEI, and DXE are described by the PI specifications. BDS, UEFI+OS Loader handshake, and RT are the province of the UEFI specification.

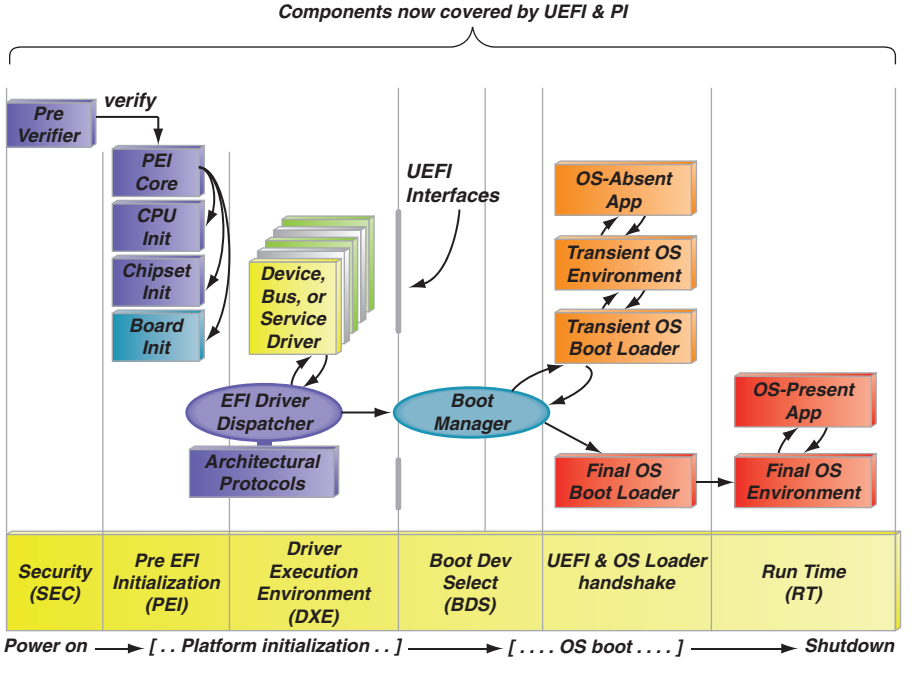

Figure 5: Where PI and Framework Fit into the Platform Boot Flow
(Source: Intel Corporation, 2011)

In addition, as time has elapsed, the specifications have evolved. Figure 6 is a timeline for the specifications and the implementations associated with them.

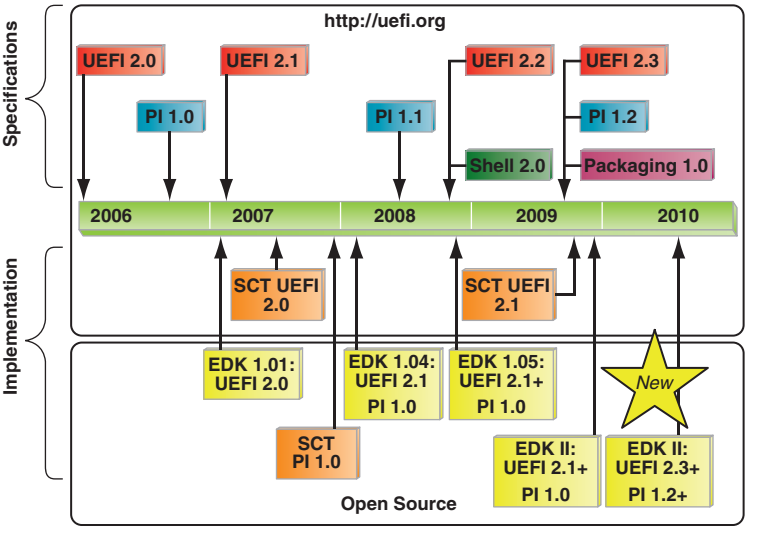

Figure 6: Specification and Codebase Timeline (Source: Intel Corporation, 2011)

Platform Trust/Security

Recall that PI allowed for business-to-business engagements between component providers and system builders. UEFI, on the other hand, has a broader set of participants. These include the operating system vendors that built the OS installers and UEFI-based runtimes; BIOS vendors who provide UEFI implementations; platform manufacturers, such as multinational corporations who ship UEFI-compliant boards; independent software vendors who create UEFI applications and diagnostics; independent hardware vendors who create drivers for their adapter cards; and platform owners, whether a home PC user or corporate IT, who must administer the UEFI-based system.

"PI differs from UEFI in the sense that the PI components are delivered under the authority of the platform manufacturer and are not typically extensible by third parties."

PI differs from UEFI in the sense that the PI components are delivered under the authority of the platform manufacturer and are not typically extensible by third parties. UEFI, on the other hand, has a mutable file system partition, boot variables, a driver load list, support of discoverable option ROMs in host-bus adapters (HBAs), and so on. As such, PI and UEFI present different issues with respect to security. Chapter 10 treats this topic in more detail, but in general, the security dimension of the respective domains include the following: PI must ensure that the PI elements are only updateable by the platform manufacturer, recovery, and that PI is a secure implementation of UEFI features, including security; UEFI provides infrastructure to authenticate the user, validate the source and integrity of UEFI executables, network authentication and transport security, audit (including hardware-based measured boot), and administrative controls across UEFI policy objects, including write-protected UEFI variables.

A fusion of these security elements in a PI implementation is shown in Figure 7.

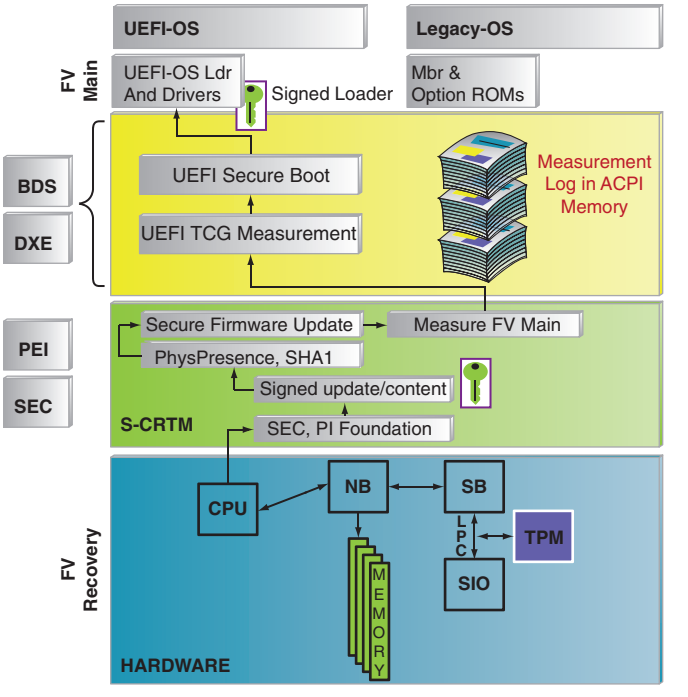

Figure 7: Trusted UEFI/PI stack
(Source: Intel Corporation, 2011)

Embedded Systems: The New Challenge

As UEFI took off and became pervasive, a new challenge has been taking shape in the form of the PC platform evolution to take on the embedded devices, more specifically the consumer electronic devices, which have a completely different set of requirements driven by user experience factors like instant power-on for various embedded operating systems. Many of these operating systems required customized firmware with OS-specific firmware interfaces and did not fit well into the PC firmware ecosystem model.

The challenge now is to make the embedded platform firmware have similar capabilities to the traditional model such as being OS-agnostic, being scalable across different platform hardware, and being able to lessen the development time to port and to leverage the UEFI standards.

"The challenge now is to make the embedded platform firmware have similar capabilities to the traditional model such as being OS-agnostic, being scalable across different platform hardware, and being able to lessen the development time to port and to leverage the UEFI standards."

How the Boot Process Differs between a Normal Boot and an Optimized/Embedded Boot

Figure 8 illustrates that, from the point of view of UEFI architecture, there are no design differences between the normal boot and an optimized boot. Optimizing a platform's performance does *not* mean that one has to violate any of the design specifications. It should also be noted that to comply with UEFI, one does not

need to encompass all of the standard PC architecture, but instead the design can limit itself to the components that are necessary for the initialization of the platform itself. Chapter 2 in the *UEFI 2.3 Specification* does enumerate the various components and conditions that comprise UEFI compliance.

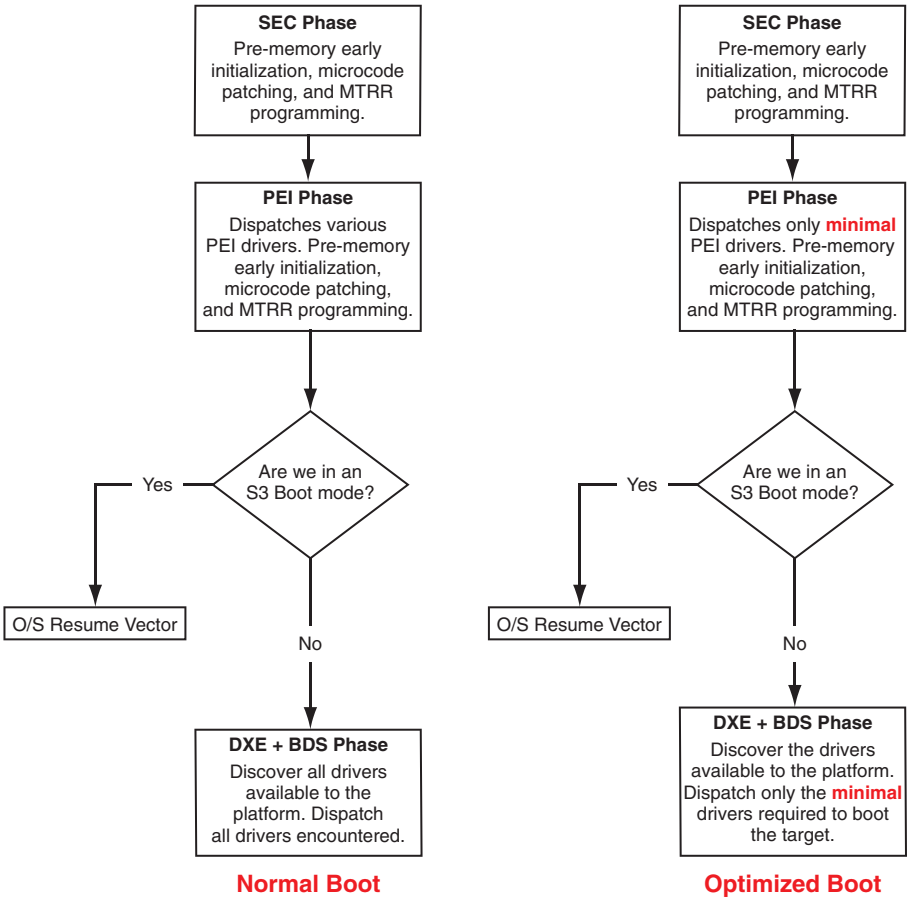

Figure 8: Architectural Boot Flow Comparison
(Source: Intel Corporation, 2011)

Summary

We have provided some background about the history that led to the creation of the BIOS standards that are developed today. In addition, we have hopefully provided some insight on how the UEFI forum operates and opened the door for people to understand how UEFI applies within their platform. Finally, we have given some pointers to the open source aspect of UEFI such that people can follow the evolution of the codebase technology to help realize implementations of this technology. As you read the other articles in this journal, you should see a very clear indication of some of the usage and capabilities exhibited by various members of the industry.

So fasten your seatbelt and dive into a journey through industry standard firmware.

Authors' Biographies

Mark Doran is a Senior Principal Engineer with Intel Corporation. He is Intel's lead architect for UEFI work. His prior work includes OS kernel development and IEEE POSIX standards content development. His first venture into standards for the firmware space was the Intel Multiprocessor Specification, the first recipe for building Intel Architecture symmetric multiprocessor computers that run shrink-wrap operating system binaries. Mark is originally from the UK and received a BSc in Computer Science with Electronic Engineering from University College, University of London.

Vincent J. Zimmer is a Principal Engineer in the Software and Services Group at Intel Corporation and has over 18 years experience in embedded software development and design, including BIOS, firmware, and RAID development. Vincent received an Intel Achievement Award and holds over 200 patents. He has a Bachelor of Science in Electrical Engineering degree from Cornell University, Ithaca, New York, and a Master of Science in Computer Science degree from the University of Washington, Seattle.

Can be contacted at http://www.twitter.com/VincentZimmer and vincent.zimmer@gmail.com

Michael A. Rothman is a Senior Staff Engineer in the Software and Services Group at Intel and has more than 20 years of operating system and embedded software development experience. Michael holds over 200 patents and was awarded an Intel Achievement Award for some of his systems work. He started his career with kernel and file system development in OS/2 and DOS and eventually migrating to embedded operating systems work and firmware development.

Can be contacted at http://www.twitter.com/MichaelARothman and michael.a.rothman@gmail.com