A Tale of Two Standards

Abstract

It is well understood that UEFI and the PI (Platform Initialization) [3] environment are designed to replace BIOS. However, there are still some firmware architects, engineers and managers uncertain of what UEFI is and why it came about.

In the face of incumbent capabilities the evolution of a new technology, faces many challenges. This paper catalogs the evolution of a standards-based solution in the face of an incumbent infrastructure. The specific example given in this paper is the boot firmware's transition from BIOS to UEFI (Unified Extensible Firmware Interface) [2][9]. Comparisons are also made with other transitions, such as the transition from a proprietary networking solution to an IP-based networking solution, and from IPv4 [7] to IPv6 [8].

Examining two technologies in this paper, we compare and contrast the evolutions of the boot firmware technology and the networking solution. Viewing these transitions in technology it clarifies that not all transitions happen at a similar pace. While the transition of the networking technologies went through a fairly long period of evolution, the relatively rapid transition underway in the BIOS industry is quite remarkable. For example, the initial IPV6 roll-out began in 1995, with greater adoption toward the end of this decade. On the other hand, UEFI adoption started with the UEFI 2.0 specification in 2006 and will have volume OS shipments that support this technology at the end of 2007.

INTRODUCTION – TALE OF TWO TECHNOLOGIES

Of the two technologies in this article, one of them includes the evolution of internet protocol (IP) based networking. This evolution spans the requests-for-comment (RFC's) within the Internet Engineering Task Force (IETF). The other evolutions include the analogous growth of boot firmware through the Unified Extensible Firmware Interface (UEFI) specifications and the associated UEFI Forum.

UEFI is several things. It is the moniker for the industry standards group, the "UEFI Forum," that is responsible for the governance of UEFI specifications. Furthermore, within the UEFI organization, there are several groups. First, the UEFI Specification Working Group (USWG) manages the evolution of the main interface document. Today, the UEFI 2.1 specification represents the latest version. Next there is the Platform Initialization Working Group (PIWG). Because the UEFI Specification is the interface between the operating system and option ROM's, the PIWG's Platform Initialization (PI) specifications describe a means by which to construct an initialization environment. This PI environment is one means by which to provide the main UEFI interfaces, and the PI infrastructure can also publish other boot interfaces, such as BIOS runtime.

NETWORKING AS A TECHNOLOGY

Life in today's world would be very different without the concept of connected devices. If companies had not developed solutions leveraging networking concepts, we would be without the ability to watch digital cable, browse the web, exchange e-mail, use our cell-phones, to name a few such common solutions.

Over time, the concept of a network has evolved from some of the simplest peer-to-peer interactions to those offering very complex transactions.

History of Networking

In this article the term "networking" refers to the ability to receive and send data across long distances. This article skips past some of the early non-electronic based methods of communications such as semaphore, smoke signals, and the like. The first examples of devices exhibiting the traits that we might call networking using a signal (albeit analog) were created as far back as the 19th century. In 1840 Samuel Morse was award US patent #1647 for "Improvement in the mode of communicating information signals by the application of electro-magnetism". This is the first example where two distinct entities could communicate with each other through an electronic signal.

The telegraph illustrated how an invention could allow for various means of communication in a peer-to-peer type of connection. This type of peer-to-peer communication evolved into usage models for such mundane devices as the telephone.

By the 1960's, digital communications could ensue on a large proliferation of phone lines. In 1960 some of the first practical MODEM (**mod**ulator-**dem**odulator) devices were introduced. In this same decade the Massachusetts Institute of Technology (MIT) started research on connecting mainframes on a phone-based network. The main sponsor of this effort was the Advanced Research Projects Agency (ARPA). By 1969 ARPAnet was commissioned by the US Department of Defense for further research. These actions might be considered the beginning of the present-day Internet.

During the 1970's and 1980's several parallel courses of communication evolution occurred in the mainframe and PC world. With the ARPAnet project, more dedicated nodes were added for communicating between major sites (e.g. several government agencies, several universities, etc.) New communication protocols such as TCP/IP were introduced to more easily identify targets and facilitate the efficient transmission of data. This was the first time that the "Internet" term came into common usage.

With the PC, the 1980's were dominated by the use of MODEM-based communication technology. The usage models were that of a PC connecting to a bulletin board to establish a 1:1 communication with another PC, or that of a PC connecting through a dial-up connection to a mainframe that would allow for multiple PC's to dial in as a client. Common examples of the latter were dial-up services such as CompuServe and AOL.

As requirements evolved and the amount of transmitted data increased, some of the inherent data throughput limitations became more evident. In various compute-based environments, networking computers capable of transmitting data in a relatively quick manner became very important.

Evolving the Network standards

With basic PC Networking coming to the foreground in the 1980's, various network-related protocols evolved. One such protocol, the Internet Protocol (IP), is used so that machines can uniquely identify each other across the Internet. The fourth iteration of the IP protocol defined in RFC 791 used a 32-bit address to identify a participant on the network. This limits it to roughly four billion unique identities.

In the 1990's the number of addresses was becoming an issue. There were various strategies to try an alleviate some of the address limitations, such as the adoption of network address translation (NAT) support to allow networks to access the internet through a single IP address. This introduced complications in how hosts communicate with each other.

Introduced in 1995, IPv6 is an attempt to alleviate the address limitation issues. With 128 bit address values, the unique address issues can be resolved, and IPv6 is currently a work-in-progress for adoption in the industry.

From the introduction of IPv6 in 1996 as a draft standard [8] to the DoD mandate for a transition plan by FY 2008 [13][14], there was a thirteen-year progression from the first draft of the IPv6 standard to the first deadline for a plan to be ready to use it.

PLATFORM FIRMWARE AS A TECHNOLOGY

A surprising amount of activity occurs between switching on a machine and the OS logon prompt for a modern computer. This is a long, tedious process to-date on PC/AT architecture platforms because the BIOS does not have a driver model or policy mechanism to know the hardware needing to be initialized for a given hardware device. As such, the BIOS initializes any possible console or storage device in anticipation of the hardware being used.

Pre-OS boot firmware complexity also increased in conjunction with the associated platform complexity. In the early days of the PC, the Industry Standard Architecture [10] memory was attached to the same bus as the devices. The I/O devices were just "there", and the user was only able to elide functionality via insertion or removal of jumpers. The memory just "worked" and was attached to the bus.

Over time, memory complexity increased. Fast page-mode DRAM and its memory controller had a few possible settings. The boot firmware could attempt a few settings, check memory, and see which worked. Modern memory controllers with Double-Data Rate (DDR) are capable of memory initialization routines running into thousands of lines of C code.

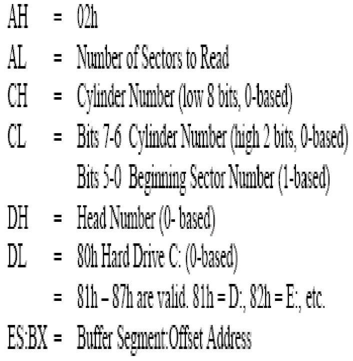
For the I/O bus, the venerable ISA bus with the fixed ISA I/O decodes went to plug-and-play ISA with some programmatic configurability up to today's Peripheral Component Interconnect (PCI) [12], which entails a full bus enumeration with bridge devices, etc. In contrast to the networking example, these platform hardware limitations are hard, rock walls auguring the UEFI transition.

The same holds true for the initialization of the central processing unit (CPU). The 286 in the PC/AT needed little initialization by the original BIOS, and there was only one in the system. On the other hand, a modern CPU, can have several physical sockets, each of which has several cores and possibly several hardware threads. Each of these sockets can have tens of model-specific registers (MSR's), in addition to other control and status registers (CSR's), that need to be initialized in synchrony with all of the other CPUs.

The original PC/AT BIOS from 1979 wasn't designed to cope with the massively different hardware designs seen today—the original designers just didn't foresee its long term use and hence didn't think about long-term extensibility. In contrast, for modern I/O buses, CPUs, and memory controllers, the ability to distribute a modular driver for each capability, such as one enabled by the UEFI Platform Initialization Architecture, is key.

Similarly, the operating system interface has evolved over time. The original interface between the BIOS and the operating system was a series of 16-bit callable interfaces invoked through software interrupts. Interrupt 13-hex, or “Int13h”, was the set of services to read or write a block device. Similarly, Int16h was to access an input console, such as a keyboard.

Initially, with 5mbyte disks, an API such as that illustrated in Figure 1 may have sufficed.



AH	=	02h
AL	=	Number of Sectors to Read
CH	=	Cylinder Number (low 8 bits, 0-based)
CL	=	Bits 7-6 Cylinder Number (high 2 bits, 0-based)
		Bits 5-0 Beginning Sector Number (1-based)
DH	=	Head Number (0- based)
DL	=	80h Hard Drive C: (0-based)
	=	81h – 87h are valid. 81h = D:, 82h = E:, etc.
ES:BX	=	Buffer Segment:Offset Address

Figure 1. INT 13h Interface Definition

This initial API had an 8Gbyte limit. Over time, as disks became larger, the Int13h API had to be expanded.

Just as NaT’s helped to cover the shortage of IP addresses, these Int13h extensions tracked the size of disks over time.

Other artifacts of BIOS and boot, namely the partition table is in the Master Boot Record, limit partitions to two-Terabyte disks. This is an on-disk encoding that cannot be evolved via BIOS APIs.

To provide arbitrarily large partitions, the UEFI Specification defines a Guided Partition Table (GPT)[2].

History of “Bootstrap”/PC Booting

Ultimately, the main point of a BIOS, and therefore the boot process of a machine, is to launch a target piece of software, typically an operating system. In the earliest systems, booting was accomplished by entering boot instructions through toggle switches. In fact, the target of such instructions at the time was paper tape/punch cards for their “operating system”. These initial machine instructions were intended to start the machine.

Through the years, the targets have changed:

In the PC of the late 1970’s and early 1980’s, the target was a ROM-based item such as a BASIC interpreter built into the system. Originally, when the typical PC started, the user was launched into something akin to a BASIC prompt, and the only programs capable of being loaded were from a cassette tape. This was a common user interface in the earliest PCs. As time evolved, Floppies, Hard Disks (MBR), CD’s, networks, etc., also became possible targets.

Over the years, BIOS too has evolved:

The original PC BIOS was intended for 250,000 units on a production run over two years. It has ended up being used for the last 25 years. IBM’s publication of the schematics, and most importantly, the BIOS source code for the PC/AT led to many clones and imitators.

In order to attempt to fend off imitations, in 1987 IBM created the PS/2. This machine had a proprietary bus referred to as "Microchannel" (MCA). IBM also attempted to move away from the original BIOS with "ABIOS" or "Advanced BIOS." ABIOS was a protected-mode BIOS that ran in 32-bit to match the newly-introduced Intel ® 386 CPU.

For the RISC machines of the 1990's, there was an effort to rally around a common firmware standard. An industry group referred to as Advanced RISC Computing (ARC) [6] aimed to unify the boot environment for MIPS CPU and DEC Alpha. ARC shared some features with UEFI, including the support of a FAT file system in the firmware, firmware boot variables, and a C-callable interface. But ARC lacked, GPT, the extensibility of UEFI, and clear legal specification governance or an evolution path like the UEFI Forum and UEFI specification. As a design, another issue with ARC was that it specified platform design definitions. This design mandated that various components must exist in a platform, thereby limiting diversity of platform designs. Also, ARC was never ported to a volume architecture such as IA-32® or x64.

Another effort that is similar in its role to the PC/AT BIOS-based platform occurred with Open Firmware (OF) [4] and the Common Hardware Reference Platform (CHRP). CHRP was a set of common platform designs and hardware specifications meant to allow for interoperable designs among PowerPC (PowerPC) platform builders. Open Firmware, also known as IEEE-1275 [4], has issues in that it is an interpreted, stack-based byte-code. Very few programmers are facile in Forth programming, and it is renowned as being "write-once/understand-never", and having poor performance, and non-standard tools. Also, Open Firmware has a device tree essentially duplicating ACPI static tables. As such, the lack of Forth programmers, prevalence of ACPI, and the fact that UEFI uses standard tools and works alongside ACPI—versus instead-of—helped spell Open Firmware's lack of growth. In fact, it is used on SPARC and PowerPC, but it is unsuitable for high-volume machines and thus prevent it from making the leap from niche server & workstation markets.

Another effort at the evolution of BIOS is LinuxBIOS [5]. Like the Linux operating system, though, any LinuxBIOS success is based more upon the collaboration model than the technology. LinuxBIOS provides a way to implement some of the platform initialization (e.g., something akin to PI or BIOS POST) via the open source development model. But LinuxBIOS does not innovate beyond today's BIOS or UEFI. It emulates BIOS in a software emulator (e.g., BOCHS) and in its native operation. LinuxBIOS does not innovate on the PI portion. Vendors have to link all of the source code together, and because it's a static link, not binary, the GPL[17] binary exemption doesn't inhere. That is, hardware vendors must open source *all of their code/design specifications* if they link against LinuxBIOS.

Additionally, the boot environment/hand-off into the operating system is either 1) emulate PC/AT and 16-bit BIOS with something like BOCHS [18], 2) reproduce the UEFI interfaces using GNUFI [16], or 3) hand-off directly into the Linux kernel using kexec. The first is just a "BIOS" as we know it today, the second is an alternate PI implementation, and the third is a one-off, embedded OS launch strategy of which the many vendors already support (i.e., embedded recovery OS in platform firmware). And with the GPL issues around intellectual property, hardware vendors and OEM's will be unlikely to adopt LinuxBIOS going forward.

Creation and Evolving of PC Platform Standards

The PC industry has evolved over time to encompass various disparate technologies that inevitably have to interact with each other. During the normal platform initialization process there are many different components involved during system initialization. In turn, each of these components must pass information or communicate with the other somehow.

This communication is further complicated, considering that many of these components are built or delivered by different companies. As the PC industry has evolved, the need for a standard method for these components to interact with each other has become clearly evident.

There are varying opinions on the definition of a “standard”, but there are two major uses of this concept. The first use is that of a private organization (e.g. operating system vendor X) defining a standard set of APIs for the use of its operating system. This is often represented by a software development kit (SDK) provided by a particular vendor to enable customers to properly use their product. This concept also extends to add-in hardware devices, for instance when a given hardware device which might be put into a PC Platform advertises a Data book on how to use this hardware. A specific example of this might be a CPU manual describing the items which a consumer of this CPU might need to understand, such as the available op-codes, pin-outs, etc. These vendor-specific type of standards were the norm for the PC industry components until the late 1980s.

It wasn’t until the introduction of the proprietary MCA bus in 1987 (which was an evolution from the previous ISA bus) that a strong push for industry cooperation ensued. This cooperation led to the common use of a second type of “standard” in various aspects of the PC industry. These standards were ones which are co-developed by multiple interested parties in the industry. For example, in 1988 a collection of promoting companies defined an extension to the original ISA bus, EISA. This ultimately evolved into what is commonly known as the PCI bus, and extensions to this are ongoing.

Since the late 80’s this type of collaboration has been ever-expanding in the PC industry. The standards evolve over time and are typically focused on addressing certain issues or topics that a portion of the industry feels is important to address yet does not significantly erode their ability to differentiate.

Figure 2 describes the boot process of UEFI PI components, including when the UEFI interfaces are available. The driver decomposition describes how components can be put together and dispatched in order to initialize the platform.

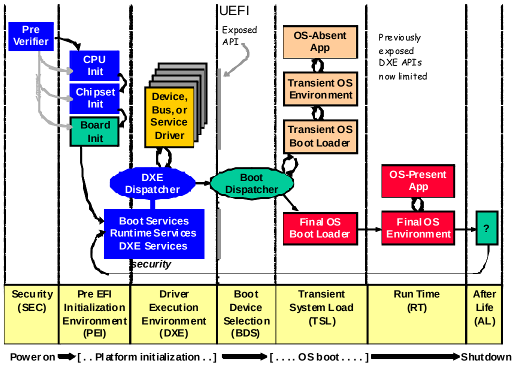


Figure 2. Flow of the PI modules

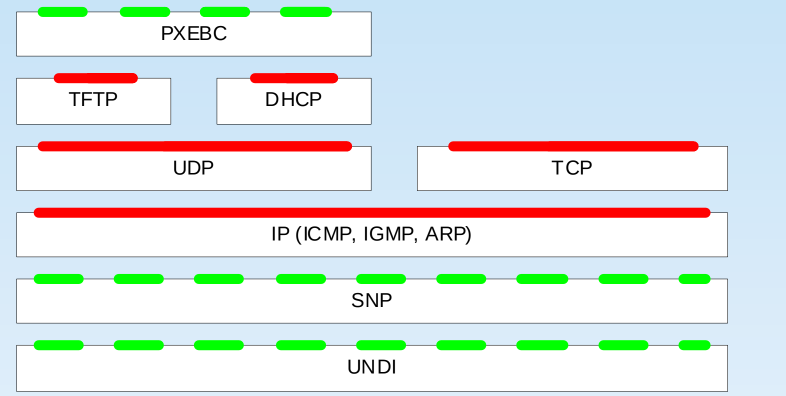
We have mentioned how some of the I/O buses, an obvious hardware component, have evolved. Other standards deal with various software and hardware combinations. For instance, standards associated with power management have evolved from the Advanced Power Management standard (APM 1.0), which was initiated in 1992, to the Advanced Configuration and Power Interface (ACPI) [11].

In addition, the interfaces associated with the BIOS evolved over time, each PC manufacturer or BIOS vendor providing their own specialized extensions. At some level this caused difficulties in interoperability. There was no clear-cut “standard” associated with BIOS. For example, the clone market was largely enabled by Phoenix Technologies and their clean-room BIOS. They had a group of experts evaluate the PC BIOS and write a specification, while a group of development “virgins” who never saw an IBM PC or PC BIOS then developed the “clone” BIOS equivalent directly from the specification.

As time passed, the inherent assumptions of the underlying platform formed a set of design constraints that limited the flexibility available to system designers wanting to innovate in the platform. These design constraints made the BIOS difficult to use and adopt across many varying PC architectures (e.g. XScale, Itanium, x64, x86, etc). For example:

- BIOS is 16-bit based, while the processor architecture has mostly evolved to 64-bit
- BIOS presents a hard limit on the size of Option ROM execution space, severely restricting the support of bootable devices on servers, both in terms of the executable image size and the number of devices supported for booting
- BIOS depends on legacy PC-AT interrupt routing and timer hardware, severely limiting server hardware design freedom. By comparison, the mainstream OSes have moved to rely on advanced interrupt control and high precision timers
- BIOS interface extensions are ad hoc, prone to conflicts as well as interoperability issues
- Most BIOS implementations are not modular, making code decomposition and sharing very difficult

With this in mind, a new set of abstractions were designed for the pre-OS space and ultimately named the Extensible Firmware Interface (EFI). EFI attempted to address the issues associated with the existing design of the BIOS and lack of clear documentation. In retrospect the EFI effort could be considered a proof-of-concept to the industry, because at the time EFI was primarily being promoted by the Itanium Processor Family ® Architecture stakeholders in the industry. Over time, the stakeholders realized that these solutions could actually address many industry issues and concerns, and EFI eventually evolved from what might have been considered a somewhat proprietary set of definitions to one adopted by a wider group of industry promoters. The corresponding EFI specification was formally adopted by the Unified EFI Forum and is now formally known as the UEFI Specification. Figure 3 illustrates some of the network component access methods which are incorporated into the UEFI Specification, including a reference to EFI 1.10 [1].



Red (solid) lines are new (for UEFI **2.0**) network protocol APIs that can be accessed by multiple applications and drivers at the same time.

Green (broken) lines are existing network protocol APIs (from EFI **1.1** or earlier) that can only be accessed by one application or driver at a time.

Figure 3. Network Support Diagram for UEFI

Current standards activities indicate that the industry is moving to the *UEFI First[15]* approach: UEFI firmware will become the design center for new features even while the industry transitions from conventional BIOS to UEFI Specification based firmware designs. In some cases, vendors are starting to develop features for UEFI, but not necessarily for BIOS. Because the PI environment provides a Compatibility Support Module (CSM) to boot legacy OSes, the expectation is a swift transition to the UEFI/PI implementation with CSM, as vendors want to drop the dual investment. The full transition to a native UEFI-only approach will happen when the support for legacy OS and tools is no longer relevant.

There is a rather rapid adoption of UEFI as a technology taking place in the industry today. The first UEFI specification was based on an earlier specification introduced in 2003, and by May 2007 support for UEFI in a very widely deployed operating system was announced. [15]

Conclusion

Evolving platform firmware for the next twenty years of computers based on the Intel Architecture is a work in progress.

As with the migration from IPv4 to IPv6, the technical arguments for the move are manifestly obvious. Unfortunately, in both cases a “light switch” model of transition where overnight the incumbent is replaced simply isn’t viable. This conclusion is inescapable given the observation that there is considerable inertia in the marketplace around the incumbent solution.

Much as with the move to IPv6, the pressure is building to make the move to firmware based on the UEFI Specification constructed using PI Architecture for the implementation design. The industry faces continuously increasing complexity and the costs of living within the constraining envelope of the incumbent.

For networking and firmware, multiple parties in the industry will need to do their part to deploy their piece of the replacement technology. For the firmware, this means BIOS vendors must supply products based on the UEFI Forum’s standards, the operating system vendors’ need to supply UEFI Specification boot code, and add-in card vendors need to supply UEFI Drivers for their cards.

Fortunately, the critical mass of the industry stakeholders in the firmware equation is included among the Promoters and Contributors of the UEFI Forum. As such, the industry is substantially aligned and is developing and deploying products that will make better firmware for Intel Architecture machines a reality.

Once the throes of transition are passed, firmware based on the UEFI Specification promises to remove many of the constraints implicit in the PC AT system design and the conventional BIOS. Freedom from these constraints represents a springboard for innovation. This will lead to an exciting expansion in the design envelope of future computers based on Intel Architecture.

References

[1] Intel. Extensible Firmware Interface (EFI) Specification, revision 1.10, 2002. http://www.intel.com/technology/efi/.

[2] UEFI 2.3 Specification. January, 2007. www.uefi.org [last accessed 8/24/2009]

[3] Platform Initialization Spec citations, Volume 1 -5, Version 1.2. www.uefi.org.

[4] Open Firmware, IEEE 1275. playground.sun.com/**1275**

[5] LinuxBIOS http://linuxbios.org/ [Last accessed 8/24/2009]

[6] Advanced RISC Computing (Arc) Specification, Version 1.2. 1992. http://www.netbsd.org/Documentation/Hardware/Machines/ARC/riscspec.pdf [last accessed 5/24/2007]

[7] Internet Protocol, Version 4 (IPV4) – RFC 791. http://www.faqs.org/rfcs/rfc791.html [last accessed 8/24/2009]

[8] Internet Protocol, Version 6 (IPV6) – RFC 1883. http://www.faqs.org/rfcs/rfc1883.html [last accessed 8/24/2009]

[9] UEFI Book http://www.intel.com/intelpress/sum_efi.htm [last accessed 8/24/2009]

[10] Don Anderson, ISA System Architecture. MindShare. April, 1995.

[11] ACPI specification Version 3.0b. http://www.acpi.info/spec.htm [last accessed 8/24/2009]

[12] PCI 3.0 Specification. http://www.pcisig.org [last accessed 8/24/2009]
Army Implementation of DoD Internet Protocol Version 6 (IPv6) Mandate.
http://jitc.fhu.disa.mil/apl/ipv6/pdf/disr_ipv6_product_profile_v3.pdf [last accessed 8/24/2009]

[13] Phase out of IPv4 in 2011.
Price Waterhouse Coopers, Technology Forecast: 2002-2004, Volume 2: Emerging Patterns of Internet Computing, Oct. 202. Pp. 541-542.

[14] WinHEC 2007 – UEFI Implementation Guidelines.
http://download.microsoft.com/download/a/f/d/afdfd50d-6eb9-425e-84e1-b4085a80e34e/SYS-T303_WH07.pptx [last accessed 8/24/2009]

[15] Adlo/LinuxBIOS http://linuxbios.org/Booting_Windows_using_LinuxBIOS [last accessed 8/24/2009]

[16] GNUFI http://www.gnu.org/software/gnufi/ [last accessed 8/24/2009]

[17] General Public License.
http://www.gnu.org/copyleft/gpl.html [last accessed 8/24/2009]

[18] BOCHS
http://bochs.sourceforge.net/ [last accessed 8/24/2009]

BIO

Vincent Zimmer is a Principal Engineer in the Software and Solutions Group at Intel Corporation. He has been with Intel for over 10 years. He has a Bacher of Science in Electrical Engineering from Cornell University and a Master of Science Degree in Computer Science from the University of Washington, Seattle. He is an IEEE Member. Contact him at vincent.zimmer@gmail.com

Michael Rothman is a Senior Staff Engineer in the Software and Solutions Group at Intel and has more than 20 years of operating system and embedded software development experience. 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. Mike has worked on many firmware products, including those with conventional code bases as well as various EFI and now UEFI deployments. Contact him at michael.a.rothman@gmail.com

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.

Dong Wei is a Distinguished Technologist at Hewlett-Packard Company. He is an IEEE Senior Member. He is the Vice President and Test Workgroup Chair of the Unified EFI Forum. He also represents HP in many industry standards groups such as ACPI and PCI. Wei received an Executive MBA degree from California State University and BSEE, MSEE and MS in Physics degrees from the University of Idaho. Contact him at dong.wei@ieee.org

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.