RESOURCE GUIDE

Overcoming BIOS development challenges in embedded systems

By Ed Brohm and Mohamad Saleh

Developers designing Intel Architecture-based systems running Windows XP, XPe, CE, or Linux usually can expect a BIOS requirement at some point during the development process. Ed and Mohamad explore the difficulties associated with BIOS development in embedded systems and how the Intel Platform Innovation Framework for Unified Extensible Firmware Interface (UEFI) can help surmount these difficulties.

BIOS is the necessary evil of embedded systems. BIOS developers know too well that while most everyone needs a BIOS, few have the patience, expertise, or budget to develop one. How are your assembly language skills? Are you familiar with the proprietary driver calls for each BIOS developer? Are you spending days debugging your board?

One solution within developers' grasp is Intel's Platform Innovation Framework for Unified Extensible Firmware Interface (UEFI), aka *the Framework* (www.intel.com/technology/framework/). How can the Framework make life easier? How about these perks for starters: faster time to market, less frustration, easier-to-use code, open source drivers, simplified debugging, write once and use many times, modularity, and more.

Transition to the Framework

In the last 25 years, platforms have expanded from 8-bit architectures to 32-bit to 64-bit with multiple cores. Video has progressed from text to color/VGA/SVGA/XVGA to SXVGA, while buses migrated from 8-bit ISA to XP or EISA to PCI Express. And Operating Systems (OSs) have evolved through four generations. While advances have been made in almost every area of embedded systems, the BIOS remained pretty much the same.

The Framework is a mature, reliable architecture available since 2002. Today, millions of units shipped worldwide contain the Framework code. A modular, platform-independent architecture that can perform boot and other BIOS functions, the Framework is driver based, so developers can perform binary linking. It is also written in C, scalable, and modular across different Intel platforms.

In the traditional firmware model, the BIOS discovers and initializes the hardware. The OS relies on the BIOS to discover the hardware and is typically tightly coupled to BIOS boot services. In this model, manufacturers provide hardware-specific reference code that BIOS vendors embed into their products.

In the model utilizing UEFI and Intel's Framework, a layered architecture is implemented with the OS and OS loaders dealing directly with the UEFI, containing a set of standard modular interfaces that replace traditional proprietary BIOS interfaces. The Framework provides BIOS functionality, preboot services in a standards-based environment, and a stage for loading the UEFI drivers in an orderly manner.

Challenges

Three specific BIOS challenges can create perplexing problems that lead to missed deadlines, poor embedded designs, budget overruns, and general emotional distress. These development challenges are in the areas of:

- Thermal management
- Firmware application development (including lack of portable and modular applications)
- Board debugging

Thermal management

As processor technology advances, thermal management for embedded systems becomes essential to ensure that temperatures are maintained within a specified range. When processor power supplies exceed their specified tolerance range, permanent damage to the processor or other components may occur.

The Framework initializes the Intel Thermal Monitor enabled registers

and provides the control methods for Advanced Configuration and Power Interface (ACPI). This in turn enables the processor to support thermal monitoring and performance control. Multiple thermal monitoring capabilities are supported through proper configuration to reduce internal clock frequency via internal clock modulation or perform a transition to lower voltage/bus ratio.

When thermal management is initiated on a platform, the processor's performance decreases as the processor operates in a degraded state. A good thermal design limits the amount of time the processor spends under thermal management, quickly returning to a normal state and maximum processor performance.

The two broad categories of thermal management are active and passive cooling. Active cooling, typically used in PCs, is provided via a fan(s) to lower internal ambient temperature. A system's BIOS can play a role in active cooling and is less of a challenge than passive cooling.

Passive cooling typically utilizes one of three Intel processor-cooling methods:

- Thermal Diode
- Thermal Monitor
- Digital Thermal Sensor (DTS)

The Thermal Diode method simply monitors the processor temperature. Upon reaching the maximum processor temperature, the Thermal Diode can be read by an analog/digital converter. The firmware can read this temperature and set the trip point for an external device to generate an ACPI interrupt that will notify the ACPI-aware OS to take action or, in some cases, shut down. Processor cooling using the Thermal Diode method is considered an unreliable indication of when the maximum processor operating temperature is met because it may not reflect the hottest location on the processor die.

The Thermal Monitor method helps protect the processor when it reaches its maximum operating temperature by controlling temperature via the Thermal Control Circuit (TCC). The challenge facing embedded systems designers is to limit TCC activation, which may decrease the processor's long-term reliability and cause excessive processor performance degradation. Typically, with the help of the BIOS, the Thermal Monitor activates the processor clock throttle and/or processor speed step. Multiple Thermal Monitor capabilities are supported through proper configuration to reduce internal clock frequency via internal clock modulation or perform a transition to a lower voltage/bus ratio. CPU temperature, chassis temperature, fan speed, system voltage, and so on can be monitored and displayed simultaneously in the Setup Configuration Utility.

The last method of monitoring Intel processor temperature, the DTS, is more accurate than the Thermal Diode because it can be located closer to the hottest portions of the processor, providing improved monitoring capabilities and clock modulation activation via the Thermal Monitor. The differences between the DTS and Thermal Diode can be significant because of software, processor power, and mechanical factors. It is also important to note that the BIOS firmware calibrates the DTS temperature reading relevant to the CPU Thermal Diode. This calibration is a major factor in meeting hardware thermal specifications.

Firmware application development

Before the Framework for EFI became available, most BIOS coding was done in a proprietary environment particular to a BIOS vendor using assembly language. An agreed-upon set of APIs, well-defined functions for preboot and runtime services, and modularity to allow simultaneous development among software engineers did not exist because each BIOS vendor had a proprietary system.

The UEFI specification defines a standard interface between the OS and platform firmware. Data tables within the interface contain platform-related information plus boot and runtime service calls available to the OS and its loader. Together, these features provide a standard environment for booting an OS and running preboot applications.

The Intel Platform Innovation Framework for EFI Architecture (Figure 1) describes the building blocks needed to develop firmware for platforms based on Intel architecture. The design is characterized by a central framework (the green box shown in Figure 1) that provides services and infrastructure in combination with modular software building block elements. The Framework is designed to allow embedded system developers to use software modules from different vendors and share those modules across platforms and projects.

Taken as a whole, the infrastructure code (known as the Foundations) and an appropriate number of modular building blocks comprise a complete platform software implementation designed to initialize the platform and boot shrink-wrap OSs or other custom application environments. The Foundations consist of the Pre-EFI Initialization (PEI) and Driver Execution Environment (DXE) phases. In addition, the Framework specification describes a number of additional design elements to provide a complete set of standard design and services support for auxiliary building blocks that will customize a particular platform firmware image to the hardware and its intended purpose. These additional elements include security services, boot device policy management, runtime services, user interface, firmware integrity services, manageability support, and legacy compatibility.

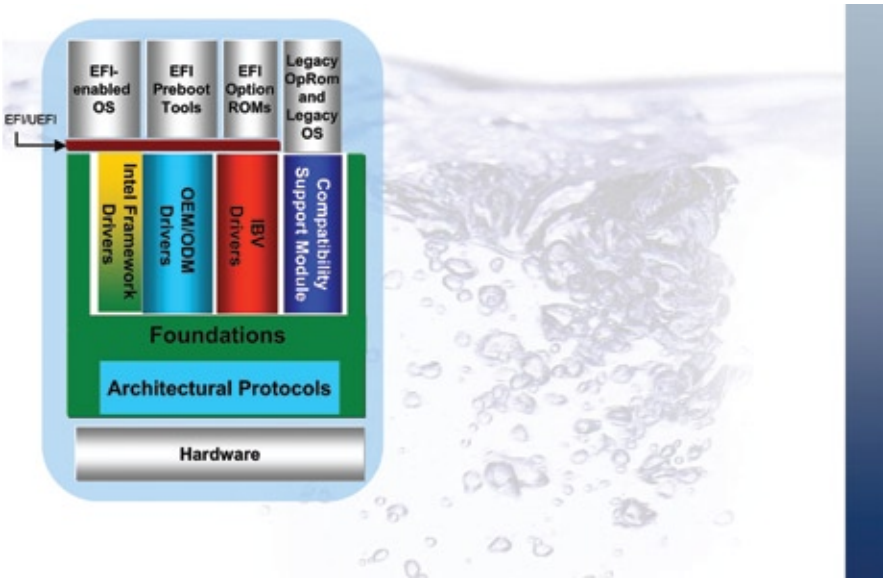

Figure 1

BIOS based on the Framework provides the following major benefits:

- The BIOS can be built to facilitate the developer's firmware application by incorporating built-in tests, diagnostics, or editing environments – all through the UEFI shell
- Validate the hardware and modular applications from the UEFI shell and use those applications where needed (such as in the preboot sequence)
- Code modularity
- Implemented in C language
- Address space management and protected-mode memory
- Automatic ordering of driver execution
- Preboot applications and EFI drivers enable efficient manufacturing test automation and platform management

Board debugging

A board debugging utility is a powerful tool that enables firmware engineers to easily find bugs, speed development time and board bring-up, and simplify application transfer between boards and project work groups. BIOS debugging options can be classified into two broad categories: firmware status and hardware and firmware debugging.

Firmware status options consist of output codes only and are not considered true debugging. The codes include:

- EFI status codes through the serial console port
- Post codes (from port 80h)
- Beep code (from audio device)

True hardware and firmware debugging options consist of:

- Hardware Integrated Circuit Emulator such as American Arium, providing source-level debugging during the PEI, DXE, and OS boot phases for EFI and native debug support for EFI Framework platforms; users can single step, set break points, see the call stack, set variables, and perform other real-time debugging functions
- Intel In Target Probe/eXtended Debug Port
- InsydeH2O Diagnostic and Debugging Tool (H2ODDT) using the LPT parallel port and USB 2.0

The H2ODDT is a powerful utility enabling firmware engineers to easily trace InsydeH2O code in the target platform. This debugging tool provides a stable connection, flexible debugging method, and easy user interface and can debug at the source code level before system memory is made available.

H2ODDT capabilities include:

- Users can define a project that represents the target platform and/or the firmware image
- Individual windows can be easily docked/tabbed to enhance usability
- Individual windows can be easily invoked to access I/O registers directly or indirectly through I/O ports, PCI register space, base I/O, read/write memory region, show CPU registers, manipulate hardware/software breakpoints, dump postcode, and so on
- Address values of a memory region or register value of an I/O space can be read, written (some registers are write protected and read only), and saved to file or buffer; values stored to buffer or file can then be compared against newly entered values
- Users can view and trace in C source code and assembly code concurrently

The future of BIOS

The Framework for UEFI is the future of BIOS development in embedded systems, helping developers overcome major firmware development challenges within thermal management, firmware application development, board debugging, and other areas not mentioned in this article.

The Framework architecture enables ODMs and OEMs to improve efficiency, validate hardware, lower costs to support new hardware designs, and speed up system debugging. Its modularity allows faster migration from platform to platform with less engineering resources and provides development flexibility because work can be done in parallel. Overall, the Framework for UEFI is an EFI that can be used across many platforms for years to come. ECD

Ed Brohm *is director of sales for Insyde Software based in the Southborough, Massachusetts office and is responsible for Insyde's channel partners, managing European sales, and working with major OEMs in the United States. He has pre-vious experience managing large OEMs such as Nortel and Motorola after serving as director of OEM sales at InfoLibria and maintaining sales positions at Harris & Jeffries and FTP Software. Ed has a BS in Marketing from The State University of New York and an MBA in Quantitative Marketing from Boston College.*

Mohamad Saleh *is a senior principal engineer for Insyde Software based in the Southborough, Massachusetts office and is responsible for developing and maintaining InsydeH2O firmware and supporting customers worldwide. Having worked as a principal engineer at SystemSoft Corp. and a senior hardware engineer at Wang Laboratories, Inc., he has experience in system and video firmware/hardware development.*
Mohamad has a BS in Computer Engineering from the University of Massachusetts Dartmouth and a graduate degree in Computer Architecture Engineering from the University of Massachusetts Lowell.

Insyde Software
508-983-0983
ed.brohm@insydesw.com
mohamad.saleh@insydesw.com
www.insydesw.com

