Boot Loader Choices for Small and Fast System Initialization Requirements

ABSTRACT: There are several system initialization solutions for Intel architecture. Choosing the solution that is best for a given architecture migration depends on requirements such as boot speed, boot loader size, and the amount of configurability that the system needs to support. With these factors in mind, boot loaders either target closed box designs or open box designs. Other architectures, such as PowerPC† or ARM†, are accustomed to getting the boot logic solution through open source or from their silicon vendor. This article provides an overview of the Intel architecture system initialization solutions, which include custom boot loaders and Basic Input/Output System (BIOS).

System Initialization Roles and Responsibilities

The system firmware is a layer between the hardware and the operating system that maintains platform hardware data for the operating system. The system firmware is customized for the specific hardware requirements of the platform and perhaps for a given application. Traditionally, platforms based on Intel architecture boot in three steps:

1. System firmware
2. Operating system loader
3. Operating system

As part of the power on self test (POST), the system firmware begins to execute out of flash memory to initialize all the necessary silicon components including the CPU itself and the memory subsystem. Once main memory is initialized, the system firmware is shadowed from ROM into RAM and the initialization continues. As part of the advanced initialization stages the system firmware creates tables of hardware information in main memory for the operating system to utilize during its installation, loading, and runtime execution. Hardware workarounds are often implemented during the power-on self-test (POST) to avoid changing silicon or hardware during later design phases. There may be an element of the system firmware that remains active during later stages to allow for responses to various operating system function calls.

The last task that the system firmware performs is a handoff of control to the operating system loader. The operating system loader does exactly what its name implies. It is customized with knowledge about the specific operating system, how it is ordered, and which blocks of the operating system to pull from the OS storage location. The operating system loader may be configured to extend the platform initialization beyond the system firmware's scope in order to allow for additional boot options. Depending on the system architecture and the firmware solutions that are adopted, the operating system loader and the system firmware could be part of the same binary.

The operating system completes the initialization of the hardware as it executes the software stack and device drivers. It potentially loads the human/machine interface and finally begins the applications. Care should be taken when considering combining elements of various components together as licenses may prohibit linking the objects together in various ways.

Boot Loaders for Closed Box Designs

Some embedded systems use minimized specialized (custom) firmware stacks created for fast speed, small size, and specific system requirements. These boot loaders perform static hardware configurations and only initialize critical hardware features prior to handoff to an operating system. They are tuned to a targeted OS, specific application, or function set, and support minimal upgrade and expansion capabilities.

QNX† Fastboot Technology for Intel® Atom™ Processors

QNX fastboot technology integrates system initialization into the QNX Neutrino† RTOS, eliminating the need for BIOS or other boot loader. It was developed specifically for use in the QNX Neutrino RTOS, for Intel® Atom™ processor Z5xx series platforms. Systems using QNX fastboot can achieve boot times of milliseconds while eliminating the BIOS royalty from their bill of materials. More information about QNX fastboot technology may be found at http://www.qnx.com/news/pr_3024_1.html.

Develop a Custom Boot Loader for Intel® Architecture

A custom boot loader may be developed for the Intel platform for architecture migrations where more time and effort is available and a do-it-yourself model is preferred. Developing your own boot loader requires a special set of software and hardware knowledge, and you'll need certain documents respective to the Intel architecture processor, chipset, motherboard, and other platform hardware. Additional information that will be needed includes operating system requirements, industry standards and exceptions, silicon-specific eccentricities beyond the standards, basic configuration, along with compiler and linker details, and software debug tools. Gather the appropriate documents at the start of the project.

Motherboard schematics are an absolute must. If the design is reusing an off-the-shelf solution from a vendor it could be more difficult to obtain the required information. In some cases confidential nondisclosure agreements (CNDAs) and perhaps restricted secret nondisclosure agreements (RSNDAs) with the various silicon vendors or motherboard vendors must be signed. The nondisclosure agreements (NDAs) will require some level of legal advice. Further, Memory Reference Code (MRC) requires an RSNDA agreement with Intel.

Refer to the EDC software Web site for other boot loader technology options.

Intel® Architecture System BIOS for Open Box Designs

A common requirement for open, expandable system designs is to provide the broadest possible system initialization solution, allowing the flexibility to load a wide range of off -the-shelf operating systems and methodical, dynamic hardware configurations. These designs will support multiple standard interfaces and expansion slots, and host mainstream operating systems with a broad set of pre-OS features and are ready to run multiple applications. On Intel architecture designs that require the flexibility, developers can choose from vendor-provided firmware.

Legacy Basic Input/Output System

The legacy Basic Input/Output System (BIOS) initializes the hardware and boots it to a point where the operating system can load, and it also abstracts the hardware from the operating system through various industry standard tables (ACPI, SMBIOS, IRQ routing, memory maps, and so on). Access to the hardware is directly made through silicon-specific BIOS commands or industry standards interfaces. Intel architecture has commonly used BIOS for over twenty years to support designs with multiple use cases, customizable services, multiple boot paths, native operating systems, or that are rich in features. BIOS is a common choice for legacy Intel architecture software design support. Major BIOS vendors include:

- American Megatrends Inc.†
- Insyde Software Corp.†
- Nanjing Byosoft Co., Ltd.†
- Phoenix Technologies, Ltd.†

Talking to a BIOS vendor is a great idea when the situation demands ready solutions and the return on investment merits the costs. The BIOS solution provides everything needed to get the system initialized and to a successful production cycle. Obtaining starter code from a BIOS vendor normally requires various levels of licenses and agreements for evaluation, production, and follow-on support. Additionally, a commercial BIOS usually includes a varying amount of nonrecurring engineering (NRE), and/or royalties per unit or subscription costs.

Many successful and established computer OEM development teams utilize BIOS vendors to provide a base level of software core competency, basic OS support, tools, and on-call support. Smaller companies can take advantage of BIOS starter kits, which consists of a lesser number of features and limited support.

Unified Extensible Firmware Interface

Unified Extensible Firmware Interface (UEFI) specifications define an interface layer between the operating system and the platform firmware. Intel developed the original Extensible Firmware Interface (EFI1) as a C language based firmware alternative to BIOS, and donated it to the UEFI forum as a starting point for the creation of the industry specifications, including UEFI and Platform Interface (PI). The interface and all of the platform-related information provide a standard environment for booting an operating system and running pre-boot applications. Additionally, UEFI addresses the limitations inherit with BIOS implementations such as 16-bit addressing mode, 1 MB addressable space, PC AT hardware dependencies and upper memory block (UMB) dependencies.

In 2005, the Unified EFI Forum, Inc. was formed as a nonprofit corporation whose goal is to manage and promote a set of UEFI standard specifications. The UEFI Forum is governed by a board of directors from eleven promoter companies including AMD, AMI, Apple, Dell, HP, IBM, Insyde, Intel, Lenovo, Microsoft and Phoenix, and 120 contributor and adopter member companies (MacInnis, 20092). The UEFI Forum is responsible for two specifications:

1. Unified Extensible Firmware Interface specification
   The UEFI specification defines interfaces between OS, add-in firmware drivers, and system firmware where the OS and other highlevel software should *only* interact with exposed interfaces and services defined by the UEFI specification. It includes the EFI Byte Code (EBC) specification, which defines an interpretive layer for portable component drivers.

2. Platform Initialization Interface specifications
   The PI specification defines the core code and services that are required for an implementation of the PI specifications, hereafter referred to as the PI architecture. These are the interoperability standards between firmware phases and pre-OS components from different providers.

Figure 1 is a block diagram that illustrates the UEFI software and specification interfaces.

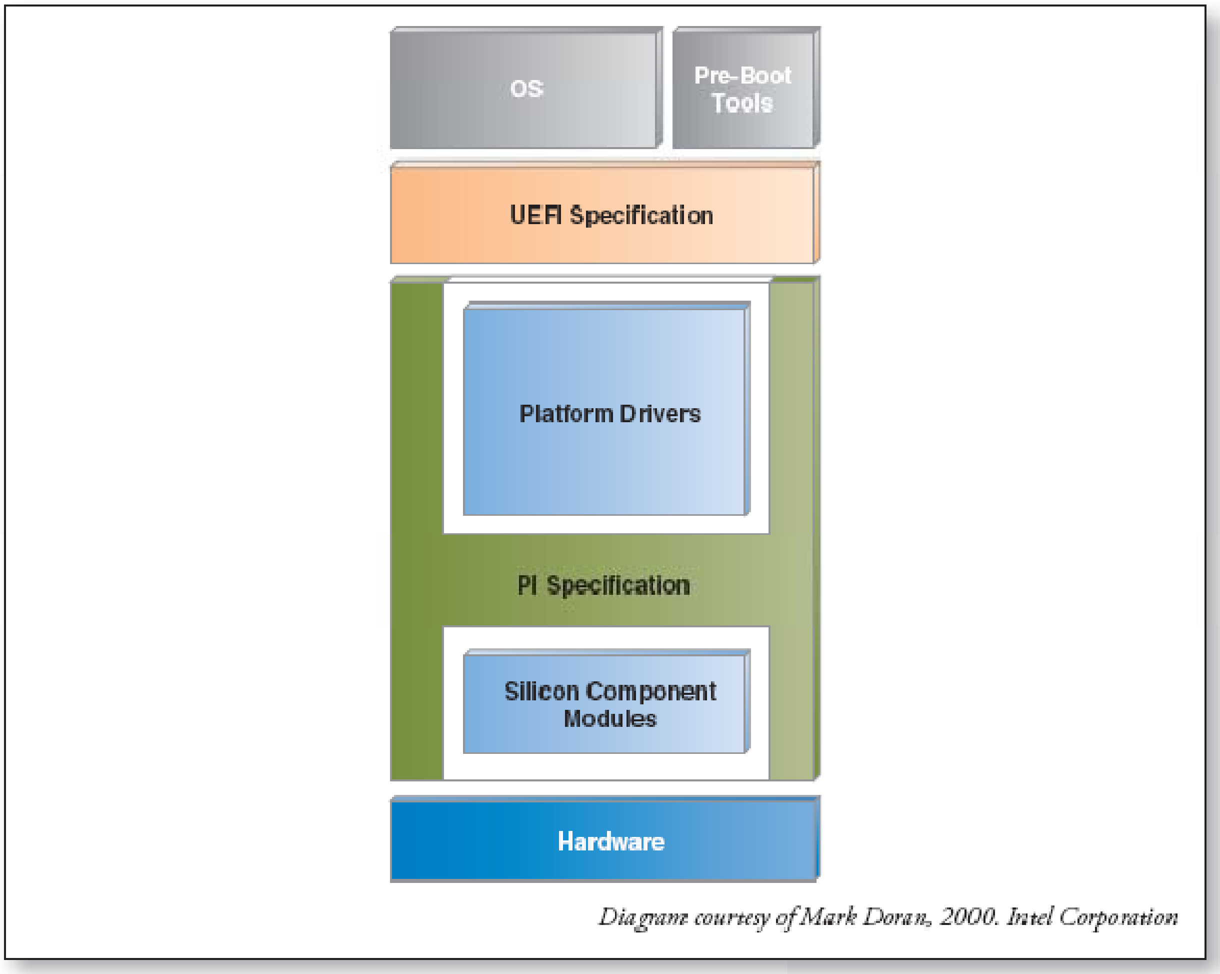
OS
Pre-Boot Tools
UEFI Specification
Platform Drivers
PI Specification
Silicon Component Modules
Hardware
Diagram courtesy of Mark Doran, 2000. Intel Corporation

Figure 1 Unified Extensible Firmware Interface

For more details about the UEFI specifications, writing UEFI drivers, and how to use the UEFI Sample Implementation and UEFI Application Toolkit, see the UEFI Web site at http://www.uefi .org/.

Intel® Platform Innovation Framework for EFI

The Intel® Platform Innovation Framework for EFI is referred to as the Framework, and previously code-named Tiano, is a reference code implementation of UEFI and PI specifications developed by Intel.

The Intel® Platform Innovation Framework for UEFI Web site describes the Framework as:

> "...a set of robust architectural interfaces, implemented in C, which has been designed to enable the firmware industry and our customers to accelerate the evolution of innovative, differentiated, platform designs. The framework is the Intel recommended implementation of the UEFI specification for platforms based on all members of the Intel® architecture family" (Intel Corporation Web site. 2010).

BIOS vendors provide a Compatibility Support Module (CSM), which is used to connect operating systems to the Framework that require legacy BIOS interfaces. The Framework firmware implementation includes support for UEFI without the CSM, but does provide interfaces that support adding a CSM supplied by a BIOS vendor. The Framework is a good solution for architecture conversions, since these designs would not already use legacy BIOS interfaces, but can take advantage of the benefits of UEFI, which include:

- Locate option ROMS above 1 MB – Legacy option ROMs have been constrained for many years by having to reside below the 1-MB boundary of 16-bit code, between C0000h and FFFFFh in system memory. In server platforms, this limited the number of add-in cards that could be plugged in. The ability to move the option ROMs above 1 MB enhances their capabilities and size.
- Faster boot – Initialize only the option ROMs needed to boot the OS and load the rest later through EFI function calls from the OS.
- Faster integration – The modularity of the PEI and DXE modules allow for faster integration of differing code modules. In some cases the faster adoption of the code bases' newer technologies into the platform.

The EFI Developer Kit is the open source portion of the Framework code base, referred to as the Foundation, and is available from the TianoCore project at http://www.tianocore.org/.

A complete Framework implementation is not generally available directly from Intel, but is offered by participating vendors as products and services based on the Framework for both Intel and non-Intel silicon. These Framework products and vendors include:

- Aptio† by American Megatrends Inc.
- InsydeH2O† by Insyde Software Corp.
- Nanjing Byosoft Co., Ltd.
- SecureCore Tiano† by Phoenix Technologies, Ltd.

For more information about implementing firmware for embedded Intel architecture systems see the Intel white paper titled “Implementing Firmware on Embedded Intel architecture Designs” at http://download.intel.com/design/intarch/papers/321072.pdf.

Intel Press has also published the book *Beyond BIOS: Implementing the Unified Extensible Firmware Interface with Intel’s Framework*, which contains examples for implementing the EFI specification

For more information about boot loader and architecture options, please refer to the book *Break Away with Intel® Atom™ Processors: A Guide to Architecture Migration* by Lori Matassa and Max Domeika.

About the Authors

Lori Matassa is a Sr. Staff Platform Software Architect in Intel’s Embedded and Communications Division and holds a BS in Information Technology. She has over 25 years experience as an embedded software engineer developing software for platforms including mainframe and midrange computer system peripherals, as well as security, storage, and embedded communication devices. In recent years at Intel she has contributed to driver hardening standards for Carrier Grade Linux, and has led the software enablement of multi-core adoption and architecture migration for embedded and communication applications. Lori is a key contributor to Intel’s Embedded Design Center, with numerous whitepapers, blogs, and industry contributions on a variety of topics critical to embedded migration.

Max Domeika is an embedded software technologist in the Developer Products Division at Intel, creating tools targeting the Intel architecture market. Over the past 14 years, Max has held several positions at Intel in compiler development, which include project lead for the C++ front end and developer on the optimizer and IA-32 code generator. Max currently provides embedded tools consulting for customers migrating to Intel architecture. In addition, he sets strategy and product plans for future embedded tools. Max earned a BS in Computer Science from the University of Puget Sound, an MS in Computer Science from Clemson University, and a

MS in Management in Science & Technology from Oregon Graduate Institute. Max is the author of *Software Development for Embedded Multi-core Systems* from Elsevier. In 2008, Max was awarded an Intel Achievement Award for innovative compiler technology that aids in architecture migrations.

Copyright © 2010 Intel Corporation. All rights reserved.

This article is based on material found in book *Break Away with Intel® Atom™ Processors: A Guide to Architecture Migration* by Lori Matassa and Max Domeika. Visit the Intel Press web site to learn more about this book: http://www.intel.com/intelpress/sum_ms2a.htm

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Publisher, Intel Press, Intel Corporation, 2111 NE 25 Avenue, JF3-330, Hillsboro, OR 97124-5961. E-mail: intelpress@intel.com .