

Course 209

Tiano & UEFI Architecture

A 2-day Course

Legacy BIOS is being retired on most new x86-based 32- and 64-bit platforms. It is replaced by firmware producing the UEFI (Unified Extensible Firmware Interface). Most implementations of this new firmware follow the highly flexible and modular open architecture defined by the *Intel Platform Innovation Framework for EFI*, better known by its development code name – *Tiano*. UEFI and *Tiano* provide developers with powerful tools for integrating new features on a platform, and test- and validation-engineers with equally powerful capabilities to do their job.

This course walks you through *Tiano*'s seven phases and explains how modules are discovered and dependencies are resolved as system hardware is initialized. The UEFI compliant interface, CSM – the BIOS compatibility module, the pre-OS execution environment, and the hand-off to an operating system boot loader are all covered in detail.

You will benefit from this workshop if you

- Need a detailed understanding of how this next-generation firmware prepares a PC to load and give control to an operating system
- Design, configure, validate, or test hardware and low-level software/firmware that depends on UEFI data structures
- Are planning the migration from legacy BIOS to UEFI/*Tiano* based systems

You will learn

- The order and relevance of *Tiano*'s phases
- When and how key subsystems and devices are detected, initialized, and configured
- How backward compatibility can be implemented, making it possible to boot legacy, MBR-based, operating systems
- Where device configuration and management information is collected, stored, and communicated between the phases and to an operating system
- The firmware's ongoing participation in running the PC

Prerequisites

Attendees are expected to have a technical background. Our *PC Architecture Overview* is a good primer for this workshop. Basic knowledge of assembly language programming, microprocessor technology, memory, and standard peripherals is expected.

The training approach

- **Up to date information:** We update the materials before every event
- **Straightforward explanations:** Technical concepts and terms are explained in plain English

Workshop topics

PC firmware fundamentals

- Why firmware?
- BIOS – The legacy firmware architecture
- Limitations BIOS places on modern systems
- *The Framework* and UEFI (*Unified Extensible Firmware Interface*)
- Compatibility with legacy BIOS
- IA-32 platform essentials
 - CPU capabilities in real- and protected-mode
 - Real- and protected mode memory maps

The Framework's phases

- SEC – *Security* phase
 - Available system resources
 - Authenticating firmware boot code
 - Hand-off information
- PEI – *Pre-EFI Initialization* phase
 - Key PEI tasks
 - PEI Foundation code
 - PEIM – *PEI Modules*
 - Detection
 - Authentication
 - Dependency resolution
 - Execution
 - PPI – *The PEIM-to-PEIM Interface*
 - Creating HOBs – *Hand-Off Blocks*
- DXE – *Driver Execution Environment* phase
 - Key DXE tasks
 - DXE Foundation code
 - DXE Dispatcher
 - *The A Priori* file
 - Driver authentication
 - DXE and EFI drivers
 - DXE driver model
 - EFI driver model
 - Creation of the *EFI System Table* and other descriptor tables
- BDS – *Boot Device Selection* phase
 - BDS and UEFI
 - Console services
 - Running EFI applications
 - Boot paths
 - Selecting a boot option
 - Custom system configuration for selected operating system
 - Operating system hand-off

- TSL – *Transient System Load* phase
 - Transient OS boot loader
 - Transient OS environment
 - BSD re-entry
 - Terminating boot services
- RT – *Runtime* phase
 - Firmware services available to the operating system
 - Invoking DXE/EFI services
 - SMM – *System Management Mode*
 - *Fallback* mode
- AL – *Afterlife* phase
 - Purpose of the AL phase
 - Entering and exiting the AL phase
 - ACPI state transitions
 - Reset
 - OS panic or OS hang
 - Firmware update capsules

CSM – The BIOS *Compatibility Support Module*

- Creating BIOS-equivalent data structures and tables
- Required remnants of BIOS functionality
- Detecting and executing legacy option ROMs
- Implementation of legacy INT19 boot loader
- Hand-off to MBR based operating system loader
- Limitations of the *Compatibility Support Module* – CSM
- The *EfiCompatibility* and *Compatibility16* components

Firmware structure

- Firmware devices
- Firmware volumes
- FFS – the *Firmware File System*
- Different firmware file types
- Firmware file components
 - Header
 - Different sections

EFI disk organization

- GUID partition advantages
- MBR preservation
- GPT (GUID Partition Table) disk architecture
- The *System* partition