

## Course 209

# Tiano & UEFI Architecture

## A 2-day Course

Microsoft's announcement that all systems shipping with a desktop version of Windows 8 have to use UEFI (*Unified Extensible Firmware Interface*) and *Secure Boot* is speeding up the industry's transition from legacy BIOS to firmware producing the UEFI.

This change will impact you, whether you run Windows or not: Depending on the security policies implemented by the OEM, you may have difficulty installing the OS of your choice, even if it is UEFI based.

Most implementations of UEFI follow the highly flexible and modular open architecture defined by the UEFI Forum's *PI (Platform Initialization) Specification*, based on Intel's *Platform Innovation Framework for EFI*, better known by its 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 jobs.

This course walks you through *PI/Tiano's* seven phases and explains how modules are discovered and dependencies are resolved as system hardware is initialized. *Secure Boot*, 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 UEFI firmware prepares a system to load and give control to an operating system (**Note:** The course can be presented with the focus on either x86, Itanium, or ARM platforms)
- 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 UEFI *PI/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 system

### Prerequisites

Attendees are expected to have a technical background. For x86 architecture audiences 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

### Firmware fundamentals

- What, why, and where is “firmware?”
- BIOS – The x86 legacy firmware architecture
- Limitations BIOS places on modern systems
- UEFI Forum’s *Platform Initialization*, Intel’s *Framework* and UEFI (*Unified Extensible Firmware Interface*)
- Compatibility with legacy BIOS
- x86 platform essentials
  - CPU capabilities in real- and protected-mode
  - Real- and protected mode memory maps

### Tiano’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
  - BDS 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

### ***Secure Boot***

- Exactly what is *Secure Boot*?
- How *Secure Boot* works
- OS vendors and *Secure Boot*
- Industry concerns

### ***ACPI Implementation in Tiano***

- ACPI Support driver
- ACPI Platform driver
- ACPI table storage

### ***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