Best Practices for UEFI Driver & Option ROM Developers

UEFI Summer Plugfest – July 6-9, 2011
Presented by Brian Richardson (AMI)
Senior Technical Marketing Engineer
Agenda

• From BIOS to UEFI
• Best Practices for UEFI
• Common Driver Issues
• Summary
• Question & Answer
From BIOS to UEFI

• UEFI solves many problems for the IHV
  – Remove legacy memory & I/O limits
  – Clean driver/protocol model
  – Designed to reduce code size

• **Move from 16-bit legacy to UEFI Driver Model for add-on drivers & OpROM**
  – Generic model for multiple architectures
  – Designed to solve OEM, IBV & IHV problems
Why Standards Matter ...

Don’t let this happen to your product
UEFI – Technical Merits

Industry Standard
180+ members

C-based Coding
modern tools

Removes Legacy Limits
no dependency on 16-bit x86 design

HII User Interface
separates firmware data from interface
Agenda

• From BIOS to UEFI
• Best Practices for UEFI
• Common Driver Issues
• Summary
• Question & Answer
Best Practices for UEFI

• Simple version ... use the UEFI spec!
Best Practices for UEFI

• *Simple version ... use the UEFI spec!*

• Slightly longer version ...  
  – Only use UEFI protocols  
  – Make proper use of HII  
  – Don’t make legacy assumptions  
  – Test against multiple UEFI platforms
Let’s go to the UEFI Spec ...

UEFI 2.3.1, Pg. 17

“When UEFI drivers and UEFI applications are loaded they have access to all UEFI-defined runtime and boot services. See *Figure 2*.”

[Diagram of UEFI boot process]
UEFI Driver Model

• Notice the UEFI Driver doesn’t have arrows going back to “platform init”
• UEFI Drivers only make use of Boot Services and Runtime Services
  – No PI protocols
  – No EDK protocols
  – No CSM callbacks

Calling these from UEFI drivers can be unpredictable …
Sticking to the Spec ...

This is not “UEFI”. BIOS could be based on UEFI/PI, UEFI/EDK, UEFI/EDKII, DUET, …

This is the standardized layer in the UEFI Spec (Boot/Runtime)
Use HII for User Interface

• **Human Interface Infrastructure (HII)**
  – Firmware & Drivers publish to a “database”
  – System firmware uses a common “browser”

• Drivers don’t have to carry their own UI

• OEMs get a consistent user experience
  – No switching between multiple menus

• OEM & ODM branding happens in setup
Use HII for User Interface

Questions, Data & Strings

Localization, Input & Display
Mixing Legacy/UEFI OpROM

• Many UEFI drivers are packaged as an OpROM
• PCI spec allows multiple OpROM images on a device – Includes Legacy x86 & UEFI
• UEFI firmware sets platform policy for running OpROM
Common OpROM Combos

- Legacy ROM Only
- UEFI “native” OpROM
- Legacy ROM + UEFI EBC OpROM
- Legacy ROM + UEFI x64 OpROM
- Legacy ROM + UEFI x64 + UEFI IA32
OpROM “Awareness”

• UEFI firmware “policy” can change
  – Example: Run legacy OpROM or UEFI first?
  – IBV/OEM/ODM policy may be different

• Developers cannot assume that the UEFI firmware will run OpROM a certain way

• Make OpROM & driver code as platform independent as possible
Driver/OpROM Execution

• UEFI Drivers & UEFI OpROMs will only be executed for devices in the boot path
• Different from legacy BIOS, where all OpROMs are executed on every boot
  – This is a huge advantage for the boot time
• The OS driver cannot assume the UEFI driver/OpROM has been executed!

Make sure your OS driver team understands this fact...
Check the specs ...

• New Driver Model protocols in UEFI 2.2
  – Driver Family Override (optional)
  – Driver Supported EFI Version (required)
• Device config uses Driver Health protocol
  – In UEFI 2.1+ `DriverConfiguration` and `DriverConfiguration2` are depreciated
• All this and more can be discovered in the `UEFI Specification` at uefi.org

`UEFI Specification` at uefi.org

`click me ... click me ...`
Agenda

• From BIOS to UEFI
• Best Practices for UEFI
• Common Driver Issues
• Summary
• Question & Answer
Common Driver Issues

• Calling non-UEFI protocols (PI & EDK)
• Poor handling of function returns
  – Error returns & unsupported functions
• Misusing HII Database Protocols
  – Managing HII packs, generating UI elements
• “Inappropriate Touching” 😱
  – Trying to configure other platform hardware
Focus: non-UEFI Protocols

• Some protocols come from specific implementations, so don’t rely on them
  – UEFI != EDK ... UEFI != PI ... UEFI != CSM
  – Code to the spec, not an implementation

• Other UEFI protocols are optional
  – Check to make sure protocols are installed before calling and handle errors gracefully

And while we’re on the subject ...
Focus: Function Returns

• Per spec, if a service returns an error, the output parameters are undefined
• Check the return code instead of just checking the output parameters
• Use return codes to verify protocols are installed
Focus: Function Returns

• Invalid return values when calling `RouteConfig()` and `ExtractConfig()`
  – Example: routine returns `EFI_SUCCESS` when it is unsupported (incorrect)
  – Browser reads `EFI_SUCCESS` value and tries to interpret an invalid return string

• Result: *unknown behavior*
Focus: User Interface

• Check if the Console is installed *before* use
  – Check for NULL pointer in the system table
  – What if it’s a headless system (no console)?
  – Always consider possibility of NULL pointers

• Proper use of add/remove formset pack
  – Only update when something changes
  – The firmware won’t check if something is different (too much overhead)
Focus: User Interface

• Avoid direct user interaction
  – Publish protocols for firmware interaction
  – Use DriverHealth protocol for mandatory configuration or repair operations

• Don’t directly invoke popup windows
  – Formset elements such as InconsistentIF can create conditions to trigger a popup

• Remember ... drivers provide forms, the HII browser provides the user experience
  – Look & feel varies between platforms
Focus: Inappropriate Touching

- Configure your hardware ... don’t configure theirs
- Yes, this seems obvious ... but it can be a problem
- UEFI encourages portable code, so making platform assumptions doesn’t work
Debug Output: Old School

0x80? Byte? Word?

0x03F8? 0x02F8?
Debug Output: New School

Don’t assume legacy ports are available.
Example: Debug Output

• Some drivers try directly access hardware for debug output (USB, COM, Port 80)
  – Problem: *hardware is already in use*
  – Result: the driver breaks system output

• Solution: *call standard output protocols*
  – `GST->StdErr`
  – More flexible
  – Works with new tools
Example: PciIo Attributes

- Avoid enabling unsupported PCI attributes
  - PciIo->Attributes
  - Support is required from the PCI Controller, PCI-to-PCI Bridge and PCI Bus Controller for an attribute to properly take effect
- Check platform attributes before enabling EfiPciIoAttributeOperationSupported
- Avoid using EDK macros to enable devices
Example: PciIo Attributes

- `EFI_PCI_IO_ATTRIBUTE_ISA_IO_16`
- `EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO_16`
- `EFI_PCI_IO_ATTRIBUTE_VGA_IO_16`
- `EFI_PCI_IO_ATTRIBUTE_ISA_MOTHERBOARD_IO`
- `EFI_PCI_IO_ATTRIBUTE_ISA_IO`
- `EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO`
- `EFI_PCI_IO_ATTRIBUTE_VGA_MEMORY`
- `EFI_PCI_IO_ATTRIBUTE_VGA_IO`
- `EFI_PCI_IO_ATTRIBUTE_IDE_PRIMARY_IO`
- `EFI_PCI_IO_ATTRIBUTE_IDE_SECONDARY_IO`
- `EFI_PCI_IO_ATTRIBUTE_DUAL_ADDRESS_CYCLE`

Check platform attributes before enabling
Other Areas of Concern

• Hooking periodic timers
• MP Aware Code ... “Unless otherwise specified a protocol’s member function is not reentrant or MP safe.”
  – Many firmware implementations will block this type of call to avoid reentrance issues
• Using `BrowserCallback()` properly
  – This is driver function intended to be called by a callback handler ... weird things may happen if other functions call it
Agenda

• From BIOS to UEFI
• Best Practices for UEFI
• Common Driver Issues
• Summary
• Question & Answer
Summary

• The UEFI Driver Model has multiple benefits over Legacy BIOS Option ROMs
  – Removes legacy x86 hardware limitations
  – Based on well documented standards
  – Decoupling driver & OpROM from the UI
• Code to specification, not implementation
• Test against multiple UEFI implementations
• This presentation is only the beginning … check uefi.org for more information
Relevant UEFI Spec Sections

Based on UEFI 2.3.1 Specification

• 2.5.1 Legacy Option ROM Issues
• 10 Protocols – UEFI Driver Model
• 13.4.2 PCI Option ROMs
• 20 EFI Byte Code Virtual Machine
• 28 HII Overview
• 29 HII Protocols
• 30 HII Configuration Processing and Browser Protocol
Agenda

• From BIOS to UEFI
• Best Practices for UEFI
• Common Driver Issues
• Summary
• Question & Answer
Thanks for attending the UEFI Summer Plugfest 2011

For more information on the Unified EFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by
American Megatrends
But wait, there’s more ...

That’s it for now ...
Visit uefi.org to download presentations, specifications and other documentation.

Download presentations after the plugfest at www.uefi.org