What They Never Taught You In UEFI 101

Tim Lewis, Chief BIOS Architect

17 September 2007
Overview

• UEFI and PI specifications create a driver model for the firmware, but…
• What else do I do to get my platform working?
  • I plug the drivers in, but it doesn’t boot…
  • I plug the drivers in, but I’m not even sure they are being executed…
  • I want to run my own utilities by I can’t figure out how…
  • I plug the drivers in, but they can’t fit in my flash part
How-To #1: Configure PI Drivers Using Platform Drivers

Driver for each of the components

But drivers need to be configured

How to do it?

Driver for Driver for each of the components

But drivers need to be configured

How to do it?

How-To #1: Configure PI Drivers Using Platform Drivers

Driver for each of the components

But drivers need to be configured

How to do it?

How-To #1: Configure PI Drivers Using Platform Drivers

Driver for each of the components

But drivers need to be configured

How to do it?

How-To #1: Configure PI Drivers Using Platform Drivers

Driver for each of the components

But drivers need to be configured

How to do it?

How-To #1: Configure PI Drivers Using Platform Drivers

Driver for each of the components

But drivers need to be configured

How to do it?
How-To #1: Configure PI Drivers Using Platform Drivers

UEFI variables are set either at build-time by build tools or by a platform setup utility.

Drivers set these options based on platform protocols (defined by driver provider).

Drivers have runtime configuration settings.

Q: Why Not Just Skip The Platform Driver Step?

A: Because Setup Screens Don’t Always Match Configuration Settings Options 1:1!
What Do Platform Drivers Do?

• The PEI platform driver **must**:
  • Detect the boot mode. Prioritize the boot modes and install the `EFI_PEI_MASTER_BOOT_MODE_PPI` and, if necessary, the `EFI_PEI_BOOT_IN_RECOVERY_MODE_PPI`.
  • Create the CPU HOB
  • Handle ROM cache settings (prior to memory discovery) and default RAM cache settings (after memory discovery)
  • Create resource HOBs for devices with fixed I/O and memory requirements
    • Flash, HPET, APIC, SIO, etc.
What Do Platform Drivers Do?

• The PEI platform driver Usually:
  • Configures GPIOs
  • Performs early chipset initialization missed by chipset drivers
  • Set up required BARs for memory controller registers, ACPI power management registers and PCI Express memory-mapped I/O.
What Do Platform Drivers Do?

• The PEI platform driver may:
  • Configure the clock generator
  • Increase the size of the boot block
    • Programming flash-device-specific registers to lock the additional sections of the flash until (AT LEAST) the next platform reset or power-on.
  • Create policy PPIs for other PEI drivers.
    • Policy PPIs are defined by the driver author, NOT the PI or UEFI specifications.
    • Hard-coded values or read from UEFI variables
    • Other PEI drivers include PPI GUID in dependency expression
What Do Platform Drivers Do?

• The DXE platform drivers may:
  • Create policy protocols for other DXE drivers.
    • Policy protocols are defined by the driver author, NOT the PI or UEFI specifications.
    • Hard-coded values or converted from UEFI variables
    • Other DXE drivers include protocol GUID in dependency expression
  • Save settings needed for S3 resume
    • What to save? Anything that’s not restored by the device’s driver.
      • For multi-mode drivers (such as SATA controllers), this is often the mode settings.
      • For host controllers (USB, PCIe) this is usually some host controller settings.
      • For devices with no specific drivers (SIOs) this is usually the SIO configuration.
    • Where to save? UEFI variables or DRAM (if initialized after the memory controller)
How-To #2: Boot menu apps are disabled boot options

- Info about apps stored in UEFI global variables with the name `Boot###` (###=hex number)
- ### must be listed in `BootOrder` global variable
- Format of the `Boot###` variable:

<table>
<thead>
<tr>
<th>App Type</th>
<th>Rsvd</th>
<th>Hidden</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rsvd</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  **EFI 1.10+**
  - For applications that are not boot options, set to 0 (Inactive)

  **UEFI 2.1**
  - 1 = Application
How-To #3: Report Status Via ReportStatusCode

- PI Specification Has `ReportStatusCode` PPI/Protocol
- Allows Different Plug-Ins for Progress/Error Reporting
  - 8-bit Port 0x80, 16-bit Port 0x80, Serial Port, Debugger, etc.

```c
    TypeSeverity,
    ClassSubclassOperation,
    Instance,
    CallerId,
    AdditionalData
);
```
ReportStatusCode: Type & Severity

- **TYPE**: 0 = Progress Code, 2 = Error Code, 3 = Debug Code

- **SEVERITY**: 0x40 = Minor, 0x80 = Major, 0x90 = Unrecovered, 0xA0 = Uncontained

- **RESERVED**: 0

- Bit positions:
  - 0-7: TYPE
  - 8-23: RESERVED
  - 24-31: SEVERITY
ReportStatusCode: Class/Subclass/Operation

For Computing:
- 0x00 = Unspecified
- 0x01 = Host Processor
- 0x02 = Firmware Processor
- 0x03 = I/O Processor
- 0x04 = Cache
- 0x05 = Memory
- 0x06 = Chipset

For Peripheral:
- 0x00 = Unspecified
- 0x01 = Keyboard
- 0x02 = Mouse
- 0x03 = Local Console
- 0x04 = Remote Console
- 0x05 = Serial Port
- 0x06 = Parallel Port
- 0x07 = Fixed Media
- 0x08 = Removable Media
- 0x09 = Audio Input
- 0x0A = Audio Output
- 0x0B = LCD Device
- 0x0C = Network

For I/O Bus:
- 0x00 = Unspecified
- 0x01 = PCI
- 0x02 = USB
- 0x06 = LPC
- 0x07 = SCSI
- 0x08 = ATAPI
- 0x0B = SMBUS
- 0x0C = I2C

For Software:
- 0x00 = Unspecified
- 0x01 = SEC
- 0x02 = PEI Core
- 0x03 = PEI Driver
- 0x04 = DXE Core
- 0x05 = DXE Boot Driver
- 0x06 = DXE Runtime Driver
- 0x07 = SMM Driver
- 0x08 = EFI Application
- 0x09 = OS Loader
- 0x0C = EBC Exception
- 0x0D = X86 Exception
- 0x0F = PEI Service
- 0x10 = UEFI Boot Service
- 0x11 = UEFI Runtime Service
- 0x12 = DXE Service
Translating `ReportStatusCode` To Port 80 (8-bit)

- Boards Still Have Port 80 LEDs For Progress
  - Class/Subclass Translated To Upper 5 Bits
  - Progress/Error Code Translated To Lower 3 Bits

<table>
<thead>
<tr>
<th>CLASS:</th>
<th>Port80 (7:3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Host Processor (0x00)</td>
</tr>
<tr>
<td>0002</td>
<td>Firmware Processor (0x01)</td>
</tr>
<tr>
<td>0003</td>
<td>I/O Processor (0x02)</td>
</tr>
<tr>
<td>0004</td>
<td>Cache (0x03)</td>
</tr>
<tr>
<td>0005</td>
<td>Memory (0x04)</td>
</tr>
<tr>
<td>0006</td>
<td>Chipset (0x05)</td>
</tr>
<tr>
<td>0101</td>
<td>Keyboard (0x06)</td>
</tr>
<tr>
<td>0102</td>
<td>Mouse (0x07)</td>
</tr>
<tr>
<td>0105</td>
<td>Serial Port (0x08)</td>
</tr>
<tr>
<td>0106</td>
<td>Parallel Port (0x09)</td>
</tr>
<tr>
<td>0107</td>
<td>Fixed Media (0x0A)</td>
</tr>
<tr>
<td>0108</td>
<td>Removable Media (0x0B)</td>
</tr>
<tr>
<td>0109</td>
<td>Audio Input (0x0C)</td>
</tr>
<tr>
<td>010A</td>
<td>Audio Output (0x0D)</td>
</tr>
<tr>
<td>010B</td>
<td>LCD (0x0E)</td>
</tr>
<tr>
<td>010C</td>
<td>Network (0x0F)</td>
</tr>
</tbody>
</table>

| 0201   | PCI (0x10) |
| 0202   | USB (0x11) |
| 0205   | PC/CARD (0x12) |
| 0206   | LPC (0x13) |
| 0208   | ATA/ATAPI (0x14) |
| 020B   | SMBUS (0x15) |
| 0301   | SEC (0x16) |
| 0302   | PEI Core (0x17) |
| 0303   | PEI Module (0x18) |
| 0304   | DXE Core (0x19) |
| 0305   | DXE Boot Service Driver (0x1A) |
| 0306   | DXE Runtime Service Driver (0x1B) |
| 0307   | SMM (0x1C) |
| 0308   | Application (0x1D) |
| 0309   | Boot Loader (0x1E) |
| xxxx   | Other (0x1F) |
Translating ReportStatusCode To Port 80 (16-bit)

- Some boards have 4 LED digits
  - [15:11] Class/Subclass Translated To Upper 5 Bits
  - [10] Error Code(1)/Progress Code(0)
  - [9] Subclass Specific(1), General(0)
  - [8:0] Operation Lower 9 Bits

<table>
<thead>
<tr>
<th>CLASS:</th>
<th>Port80 (15:11)</th>
<th>OPERATION</th>
<th>SUB-CLASS</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Host Processor</td>
<td>(0x00)</td>
<td>0201</td>
<td>PCI</td>
</tr>
<tr>
<td>0002</td>
<td>Firmware Processor</td>
<td>(0x01)</td>
<td>0202</td>
<td>USB</td>
</tr>
<tr>
<td>0003</td>
<td>I/O Processor</td>
<td>(0x02)</td>
<td>0205</td>
<td>PC/CARD</td>
</tr>
<tr>
<td>0004</td>
<td>Cache</td>
<td>(0x03)</td>
<td>0206</td>
<td>LPC</td>
</tr>
<tr>
<td>0005</td>
<td>Memory</td>
<td>(0x04)</td>
<td>0208</td>
<td>ATA/ATAPI</td>
</tr>
<tr>
<td>0006</td>
<td>Chipset</td>
<td>(0x05)</td>
<td>020B</td>
<td>SMBUS</td>
</tr>
<tr>
<td>0101</td>
<td>Keyboard</td>
<td>(0x06)</td>
<td>0301</td>
<td>SEC</td>
</tr>
<tr>
<td>0102</td>
<td>Mouse</td>
<td>(0x07)</td>
<td>0302</td>
<td>PEI Core</td>
</tr>
<tr>
<td>0105</td>
<td>Serial Port</td>
<td>(0x08)</td>
<td>0303</td>
<td>PEI Module</td>
</tr>
<tr>
<td>0106</td>
<td>Parallel Port</td>
<td>(0x09)</td>
<td>0304</td>
<td>DXE Core</td>
</tr>
<tr>
<td>0107</td>
<td>Fixed Media</td>
<td>(0x0A)</td>
<td>0305</td>
<td>DXE Boot Service Driver</td>
</tr>
<tr>
<td>0108</td>
<td>Removable Media</td>
<td>(0x0B)</td>
<td>0306</td>
<td>DXE Runtime Service Driver</td>
</tr>
<tr>
<td>0109</td>
<td>Audio Input</td>
<td>(0x0C)</td>
<td>0307</td>
<td>SMM</td>
</tr>
<tr>
<td>010A</td>
<td>Audio Output</td>
<td>(0x0D)</td>
<td>0308</td>
<td>Application</td>
</tr>
<tr>
<td>010B</td>
<td>LCD</td>
<td>(0x0E)</td>
<td>0309</td>
<td>Boot Loader</td>
</tr>
<tr>
<td>010C</td>
<td>Network</td>
<td>(0x0F)</td>
<td>xxxx</td>
<td>Other</td>
</tr>
</tbody>
</table>

© Phoenix Technologies Ltd.
ReportStatusCode: Operation Values

- The operation values depend on class/subclass.
  - Ex: Memory Controller:
    - 0 = Reading configuration data (i.e. SPD) from memory devices.
    - 1 = Detecting presence of memory devices.
    - 2 = Determining optimum configuration (i.e. timing)
    - 3 = Initial configuration of memory devices/controller
    - 4 = Optimized settings for memory devices/controller
    - 5 = Memory initialization (ECC, etc.)
    - 6 = Memory test
  - OR with 0x20 (0x04 << 3) gives port 80 values of 0x20-0x26 for the memory controller.
Debugger Output

POST CODE: 0x80
POST CODE: 0x82
POST CODE: 0x84
POST CODE: 0x88
Executing PEIM at FFFFDB8A
  PEIM name: BasememoryTest
  Entry point: 0xFFFFD220
Installing PPI at 0xFFFFD2E4
  Flags: 0x80000010 (PPI + Terminate List)
  GUID: B6EC423C-21D2-490D-85C6-DD5864EAA674
  Entry pointer: 0xFFFFD2E0
POST CODE: 0x8A
POST CODE: 0x80
POST CODE: 0x82
POST CODE: 0x84
POST CODE: 0x88
Executing PEIM at FFFFDBC2
  PEIM name: Variable1
  Entry point: 0xFFFFC814
Installing PPI at 0xFFFFC940
  Flags: 0x80000010 (PPI + Terminate List)
  GUID: 3CDC90C6-13FB-4A75-9E79-59E9DD78B9FA
  Entry pointer: 0xFFFFC938
POST CODE: 0x8A
POST CODE: 0x80
Dumping Information From The Debugger
Example: \texttt{de:PEIDISP(“PPI”)}

PPI #0 at 0x000DF99A -> 0xFFF053C8
  Flags: 0x80000010 (PPI + Terminate List)
  GUID: CA3B3A50-5698-4551-8B18-CEAEEF917D50
  Entry pointer: 0xFFF053C0

PPI #1 at 0x000DF99E -> 0xFFF0552C
  Flags: 0x80000010 (PPI + Terminate List)
  GUID: 229832D3-7A30-4B36-B827-F40CB7D45436
  Entry pointer: 0xFFF05528

PPI #2 at 0x000DF9A2 -> 0xFFF055E0
  Flags: 0x80000010 (PPI + Terminate List)
  GUID: 44010885-9F0B-4AA8-826F-B455958D1531
  Entry pointer: 0xFFF055D8

PPI #3 at 0x000DF9A6 -> 0x000DE078
  Flags: 0x80000010 (PPI + Terminate List)
  GUID: D03EC65A-C31E-4ABD-909C-8BBAA5DD4233
  Entry pointer: 0x000DE040

PPI #4 at 0x000DF9AA -> 0xFFFF6E58
  Flags: 0x80000010 (PPI + Terminate List)
  GUID: C9737920-C2AD-41C3-B133-0F9C251B6743
  Entry pointer: 0xFFFF6E40

Total 5 PPI function(s)
# How-To #4: Saving Space

<table>
<thead>
<tr>
<th>Driver Type</th>
<th>64-Bit DDK Compiler</th>
<th>64-Bit VS2005 SP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver A</td>
<td>Total Size: 11,616</td>
<td>Total Size: 7,552</td>
</tr>
<tr>
<td></td>
<td>Code: 8,416</td>
<td>Code: 6,416</td>
</tr>
<tr>
<td></td>
<td>Initialized Data: 2,592</td>
<td>Initialized Data: 528</td>
</tr>
<tr>
<td></td>
<td>Compressed: 6,395 (55%)</td>
<td>Compressed: 4,439 (58%)</td>
</tr>
<tr>
<td>Driver B</td>
<td>Total Size: 6,336</td>
<td>Total Size: 5,328</td>
</tr>
<tr>
<td></td>
<td>Code: 4,224</td>
<td>Code: 3,586</td>
</tr>
<tr>
<td></td>
<td>Initialized Data: 1,472</td>
<td>Initialized Data: 848</td>
</tr>
<tr>
<td></td>
<td>Compressed: 3,861 (61%)</td>
<td>Compressed: 3,328 (62%)</td>
</tr>
<tr>
<td>Driver C:</td>
<td>Total Size: 7,680</td>
<td>Total Size: 6,096</td>
</tr>
<tr>
<td></td>
<td>Code: 6,048</td>
<td>Code: 5,040</td>
</tr>
<tr>
<td></td>
<td>Initialized Data: 1,024</td>
<td>Initialized Data: 448</td>
</tr>
<tr>
<td></td>
<td>Compressed: 5,096 (66%)</td>
<td>Compressed: 4,121 (68%)</td>
</tr>
<tr>
<td>Driver D:</td>
<td>Total Size: 4,608</td>
<td>Total Size: 1,888</td>
</tr>
<tr>
<td></td>
<td>Code: 2,368</td>
<td>Code: 944</td>
</tr>
<tr>
<td></td>
<td>Initialized Data: 1,568</td>
<td>Initialized Data: 304</td>
</tr>
<tr>
<td></td>
<td>Compressed: 2,738 (59%)</td>
<td>Compressed: 1,193 (63%)</td>
</tr>
</tbody>
</table>
How-To #4: What Made The Difference?

• Code: Alignment of 16-bytes vs. 32-bits
• Code: Register usage
  • Better register usage (esp. pointers to interfaces)
  • Better instruction selection (e.g. AND x,0, not MOV x,0)
• Code: Link-Time Code Generation
  • Eliminates common subroutines
  • Calling conventions for static routines optimized
  • Constant folding for function parameters
• Data: Unintentional static data left in driver.
  • Usually debug strings and file names (even in release)
Summary

• How-To #1: Platform drivers customize other drivers for your platform.
• How-To #2: Use Boot Options to add your apps to the boot manager menu
• How-To #3: Use ReportStatusCode to track progress during POST
• How-To #4: Configure the right tools and the right flags to fit your drivers into the flash part.