BKK16-309B Enterprise Firmware - The gold standard and how to get there

Jeff Underhill
Why We Need Server Standards?

[Image of a box with the text "What's in the Box"]

[Image of an instruction manual with the text:
1. Installing Linux in 27 Easy Steps
2. OS / Platform Support Matrix
3. UEFI + ACPI Appendix
4. U-Boot Appendix
5. Who to call when it still doesn’t boot?]
Why We Need Server Standards?

1. Installing Linux in 27 Easy Steps
2. OS/Platform Support Matrix
3. UEFI + ACPI
4. U-Boot Appendix
5. Who to call when it still doesn't boot?
The Bar Has Been Set

"...and jeez these machines are proving difficult to use."
- Seasoned engineer at large enterprise software company

"Why are you exposing us to all this $h*t..."
- Software architect at large enterprise software company
Even Greenfield Installations...
...aren’t!
Heterogeneous Datacenters

**Requirement:** Homogeneous Platform Management and OS Deployment

### Platform Deployment Standards – ARM and x86 need to be the Same

- ACPI, UEFI Bootloader – standard server OS images
- PXE, HTTPS – standard remote boot options
- DMI – Platform inventory
- IPMI – Remote ID and control

### Specifications to explicitly define platform hardware and firmware standards

- Server Base System Architecture (SBSA) Spec. – Platform SOC requirements
- Server Base Boot Requirements (SBBR) Spec. – Platform Firmware requirements
- SBSA and SBBR enable standard server OS support

### Forums enabling organizations to collaborate on standards

- ARM Server Advisory Committee spans all ecosystem partners for standards content review
- Linaro providing Linux kernel for ARM standard deployment and UEFI Firmware Test Suite for compliance verification
- 3rd Party Firmware providers aligned Phoenix, AMI, Insyde

---

Requirement: Homogeneous Platform Management and OS Deployment
Server Base System Architecture

Server Base Boot Requirements
System Software on ARM® Platforms

Document number: ARM DEN 0044A
Copyright ARM Limited 2014

new Edition
Standards Framework

ARM Standard Server Specs
OS Specific Design Requirements
Industry Standard Server Specs
Firmware Standard Reference Code
Upstream Linux Kernel Support
SoC Specific Firmware Reference Code
Firmware Compliance Validation/Test Suite
OEM/ODM Server Development
OEM/ODM OS Platform Certification Testing
OEM/ODM Server Compliance Testing

PLATFORM DESIGN REQUIREMENTS
PLATFORM ENABLEMENT
PLATFORM IMPLEMENTATION & VALIDATION
Requirement: Homogeneous Platform Management and OS Deployment

Overarching goals:

1) Platforms ‘just work’ as received – good QE / QA
   1) Remote management minimum: On, Off, Reset & Boot order

2) Installation of alternate OS is simple and ‘just works’
   1) PXE & USB boot present and functional
   2) Don’t have to ask which OS works on which platform – single image

3) Platforms integrate seamlessly with existing environment
   1) SMBIOS present & populated
   2) No re-write of installation scripts (No “IF ARM THEN...”)
Do You Want to Build a Snowman?

A Good Snowman Is Hard to Build
BKK16-309A Enterprise Firmware – Open Platform support with UEFI

Leif Lindholm
Remember last Connect?

I held a presentation at SFO15, explaining in detail everything ARM had done wrong about UEFI since we started our involvement.

So they made ARM’s UEFI strategy my responsibility.

This is my plan for how to improve things.
What has ARM ever done for you?

So...
• ARM worked within the UEFI forum to define the bindings for AArch32 and AArch64 in the UEFI specification, guaranteeing interoperability across platforms from different vendors (and architectural licensees).
• ARM provided the support in TianoCore EDK2 for the ARM architectures.

But...
• ARM didn’t provide reference platform ports, and did not put in the effort required to help existing partners migrate to a new firmware architecture, or enable new partners to migrate to ARM.
History repeating itself

We’ve been providing ports implemented with mostly hard-wired ARM-specific platform drivers.

Sound familiar?

Basically, we are in the same situation as the ARM Linux ecosystem before the creation of Linaro. And we need the same kind of effort, to enable sensible code reuse in UEFI firmware.

Only in many cases we do not even have the existing code to refactor.
Show us the code

In order to reduce the amount of duplication between platforms, and reduce the number of different types of wheels in use, we need to have access to real platform support code.

Historically, EDK2 has never had any concept of containing full platform support, apart from a few software models. The traditional workflow was:

• BIOS vendor/team gets tasked with providing firmware for a system
• They grab a version of EDK2 they find suitable as a starting point
• They add existing in-house components for non-upstream features as well as a customized GUI
• They add support for processor and other on-board components – generally in the form of .zip files extracted in the top-level directory
Moving a planet

The TianoCore project does not have a head maintainer. It has a bunch of people acting as maintainers for individual packages within EDK2. And potentially for different repositories owned by TianoCore.

So historically, making changes to the TianoCore project (as opposed to the code) has been nigh-on impossible. It has been either:

• Driven by consensus (with no one to tell when consensus has been reached)
• Changed by maintainers making the call that it is better to ask forgiveness than permission
• Decided internally by Intel with little external discussion
A new hope

... but recently, things have been changing.

In the beginning of this year, EDK2 finally migrated from SVN to git. This had been coming for a long time, but greatly hindered by the lack of anyone to actually make the call on when the migration plan was complete.

A TianoCore community manager (Tony Mangelfeste) has been appointed (by Intel, unilaterally, but this is still a good thing).

Further positive changes are coming. With a bit of luck I’ll have something more to share before the end of this week.
A way forward

What we want is more open platform code in TianoCore, but getting that in any sensible form is a longer play.

In the meantime, I have set up my own separate platforms tree to use for reducing duplication of platform bringup boilerplate and provide example code of best practices. **We’re not there yet – don’t assume this is a showcase!**

We have now migrated all of the ARM Ltd. Platform configurations there – only ArmVirtPkg remains in the EDK2 tree. ArmPlatformPkg still exists in EDK2, holding platform drivers and ARM PrimeCell drivers. Long-term, expect this to disappear.
OpenPlatformPkg

OpenPlatformPkg is just a git repository to hold platform code until there is a way to resolve this in TianoCore.

It is *really* not a power play, trying to take things over from TianoCore. And I have a reasonable level of confidence that this can feed directly into a future solution for all of TianoCore.

On the practical level, it is simply a tree holding device drivers, platform support code, component support code, and platform build configuration files. To maximize the number of platforms I can get access to ports for, I *have* compromised and permitted binary-only modules to form a non-majority part of a port.
Reference platform

OK, so that is the ability to share code between platforms sorted(ish) – but what about this gold standard thing?

We need to improve the quality of the code available in this tree.

One way to drive this is by providing a fully open-source platform port that we actively clean up to the point where we can direct people to it and tell them to use this as a reference.

But what platform would we base this on?
Over the past few years, a lot of good work has gone into creating/improving QEMU support for the “virt” platform. But the virt platform is really about running software on AArch64 – not providing an emulated version of a realistic server system.

What would be really useful to have is a software model looking like a real system – enabling a full reference firmware/software implementation available to anyone without having to wait for a physical system to be delivered:

• ARM Trusted Firmware
• UEFI

Whilst looking more like a real server than many of the more available development boards.
What I think that should be

QEMU
• Making use of the improved EL3 emulation support to provide an ARM Trusted Firmware port
• Improving on non-upstream EL2 support, to enable verification of virtualization support
• Implementing a platform that
  • Always tracks latest upstream available ARMv8-A server processor
  • Emulates a PCI controller (rather than virtio-pci) controlling
    • NVMe device(s)
    • AHCI disk(s)
    • A standard Ethernet controller
  • Keyboard/video/mouse
And what will we do

Existing ARM platform ports, and drivers for those, have way to many global predefines. Things like hard-coded base addresses for devices. (What do you do when your system has more than one?) These need to be converted into the UEFI driver model.

Some bits reappear implemented (differently) in many new platform ports.
Non-discoverable buses

The scourge of the SoC.

UEFI was defined in a world where this simply wasn’t an issue – core platform components were compatible between vendors and “everything” else was discoverable, and on PCI.

As a result, many core bits of EDK2 completely expects the world to look like this. And because of this, we see many new platform ports implemented with some form of PCI-emulation for pure MMIO peripherals.

There are plans underway to implement a Fake Peripheral Interconnect to resolve this in one place for all platforms.
Option ROMs

One of the more powerful aspects of UEFI is the ability to add peripherals via plug-in cards and have them automatically supported to use for booting via a driver provided in flash on the card itself.

Part of the PCI specification(s), and supports providing drivers for multiple architectures. But, the hw vendors want to keep the board cost down as much as possible – so in practice they will only provide cards supplied with the X64 version.

UEFI defines a portable bytecode executable format called EBC. The majority of the interpreter code is platform independent, but the ARM architectures do not yet have their bit implemented.
Resources

https://git.linaro.org/uefi/OpenPlatformPkg.git

https://wiki.linaro.org/LEG/Engineering/Kernel/UEFI/build

https://wiki.linaro.org/LEG/Engineering/Kernel/UEFI/CommonPlatformTree