EBBR: Standard Boot for Embedded Systems
Brief History of Booting

**Server/Desktop/Laptop:**
- Many Binary OSes
- End-users OS installation
- Horizontal integration
- Combinatorial explosion of supportable options

Lots of economic incentive for interoperability
- UEFI, ACPI, Plugfests

**Embedded/Mobile**
- OS built from source
- Vertically Integrated
- OS image tied to hardware

Interoperability useful to developers, but little economic incentive to make sure it happens
- Custom boot flow, little interoperability testing
Problems for Distributions (Embedded)

- Many interesting SBCs, but
  - Every SBC behaves subtly differently
  - Impossible to support generically
- Board-specific images abound
  - DT shipped with OS does not scale
- Painful to upgrade kernel and/or firmware
- Domain specific behaviour
  - Android firmware optimized to fastboot model
  - Custom-specific scripts in U-Boot
  - ChromeOS firmware architecture is different still
- No pre-boot execution
- The world isn’t all Linux (though we pretend it is)

“Exciting” is not a good firmware feature
Solvable Problems

- No technical barriers to boot standardization
- Boot flow is more than just a distro problem
  - Even vertical software stacks benefit from standardization
  - Easier to integrate standard features (PXE, Secure Boot, firmware drivers, etc.)
- Platform specific features aren’t impeded by standards
- Stop duplicating the same work!

EBBR was conceived to create a firmware standard suitable for embedded, but aligned with server/desktop.
Case Study: Cable Set-top box

Characteristics:
- Minimal local storage
- Check network for updates on each boot
- Needs remote pre-boot agent for management
- Multiple hardware variants
- Single OS image

Benefits of standardization
- Pre-boot agent can be built against common firmware API
  - Custom applications can be separate from firmware stack
- Kernel doesn’t need to carry every hardware variant when firmware provides HW description
- Update and Security model already proven in other domains
Survey of Embedded Firmware

Projects
- U-Boot
- Little Kernel (Android)
- Core Boot (ChromeOS)
- Tianocore (UEFI EDKII)

Related Projects
- Trusted Firmware A
- OP-TEE
- SoC Proprietary
Survey of Embedded Firmware

Projects
- U-Boot
- Little Kernel (Android)
- Core Boot (ChromeOS)
- Tianocore (UEFI EDKII)

What do we start with?
- Work with existing ecosystem
- Must be implemented in upstream firmware projects
- Start with achievable goals

Most SBC supported by distros are running U-Boot, so that’s a good place to start.

Related Projects
- Trusted Firmware A
- OP-TEE
- SoC Proprietary
Overview of EBBR

Platform requirements document
- Points to other specifications
- Interprets and provides design guidance

Intended to be baseline that distros can require for support

Open process
- CC-BY-SA license
- Drafts in Github
- Conversations on mailing list
- Contributions under DCO
- Anyone can participate

Architecture independent
Overview of EBBR

Implementation independent
- Implemented in both Tianocore and U-Boot
- Other firmware implementations possible (i.e. Linuxboot)

UEFI ABI
- Sufficient subset to boot Linux, FreeBSD as a minimum

Additional architecture specific guidance
- Similar to SBBR, EBBR requires PSCI

Embedded System constraints
- Firmware & OS on same media
  - e.x., eMMC
- Limited hardware access at runtime
- Limited variable access
- Device partitioning limitations
  - E.x. firmware loaded from fixed offset into block storage
EBBR Project

Hosted on Github
  - https://github.com/Arm-Software/ebbr
Mailing list
  - boot-architecture@lists.linaro.org
Weekly Conference Call
Open Source Project
  - Creative Commons BY-SA license
  - DCO contributions
  - Anyone welcome to participate
  - Releases managed by EBBR committee
EBBR Project Status & Roadmap

- Current Release: v0.7 pre-release
- Will seek feedback at ELC-E and Linux Plumbers
- v1.0 projected for early December

Fedora, SUSE, Debian and FreeBSD server images will boot unmodified on early EBBR U-Boot Platforms

v1.0 Contents:
- AArch32 & AArch64
- Basic UEFI ABI
- Hardware Description from platform
  - Both ACPI and DT allowed, but firmware must choose one or the other
- Shared storage guidance
- PSCI required on AArch64
- Runtime Services guidance
- Exceptions to UEFI spec
EBBR Next Steps

Post v1.0:

Secure Boot
Capsule updates
More embedded use cases
  • A/B upgrades
  • Preboot requirements
Better UEFI compliance
Non-Linux representation
Thank You!

Questions?