Open Source Firmware @ Facebook
Design, deployment, and demo

David Hendricks: Firmware Engineer
Andrea Barberio: Production Engineer
It started with a quick hack back in ‘17...

OCP Yosemite V1/Mono Lake
Vendor firmware

coreboot + LinuxBoot
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

\[\text{Initialize Hardware} \quad \text{Load target OS kernel} \]

\[\text{coreboot} \]
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- **Bootblock**

Diagram:

- **Initialize Hardware**
- **Load target OS kernel**

- `coreboot`
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage

Initialize Hardware

Load target OS kernel

coreboot
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage
- Ramstage

Initialize Hardware

Load target OS kernel

coreboot
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage
- Ramstage
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

![Diagram of boot process]

- Bootblock
- Romstage
- Ramstage

**Initilaize Hardware**

PayLoad

Load target
OS kernel

coreboot

LinuxBoot

LinuxBoot logo by Susanne Nähler
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage
- Ramstage

Payload

Load target OS kernel

kexec

Target OS
System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage
- Ramstage

Initialize
Hardware

Payload

Load target OS kernel

Target OS

coreboot

kexec

LinuxBoot
Problem: Local booting is more complex

<table>
<thead>
<tr>
<th></th>
<th>Then</th>
<th>Now</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Few, simple interfaces</td>
<td>Many interfaces and protocols</td>
</tr>
</tbody>
</table>
### Problem: Local booting is more complex

*Then*
- Few, simple interfaces
- Simple, low-speed links

*Now*
- Many interfaces and protocols
- High-speed links

---

SSD: By Dmitry Nosachev [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0), from Wikimedia Commons

eMMC: By Toniperis [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0), from Wikimedia Commons
Problem: Local booting is more complex

<table>
<thead>
<tr>
<th>Then</th>
<th>Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>Few, simple interfaces</td>
<td>Many interfaces and protocols</td>
</tr>
<tr>
<td>Simple, low-speed links</td>
<td>High-speed links</td>
</tr>
<tr>
<td>Blindly execute MBR (CHS 0/0/1)</td>
<td>Scan, mount+decrypt, verify</td>
</tr>
</tbody>
</table>

SSD: By Dmitry Nosachev [CC BY-SA 4.0], from Wikimedia Commons
eMMC: By Toniperis [CC BY-SA 4.0], from Wikimedia Commons
Problem: Network booting is more complex

<table>
<thead>
<tr>
<th>Then</th>
<th>Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small, trusted networks</td>
<td>Global, untrusted networks</td>
</tr>
</tbody>
</table>
Problem: Network booting is more complex

<table>
<thead>
<tr>
<th>Then</th>
<th>Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small, trusted networks</td>
<td>Global, untrusted networks</td>
</tr>
<tr>
<td>Few, simple interfaces and protocols</td>
<td>Many interfaces and protocols</td>
</tr>
</tbody>
</table>
**Problem: Network booting is more complex**

<table>
<thead>
<tr>
<th>Then</th>
<th>Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small, trusted networks</td>
<td>Global, untrusted networks</td>
</tr>
<tr>
<td>Few, simple interfaces and protocols</td>
<td>Many interfaces and protocols</td>
</tr>
<tr>
<td>TFTP/PXE, security an afterthought</td>
<td>TLS/HTTPS, designed for security</td>
</tr>
<tr>
<td></td>
<td>BitTorrent for scalability</td>
</tr>
</tbody>
</table>
## Another way of looking at it...

<table>
<thead>
<tr>
<th>5-year growth of certain subsystems in Linux</th>
<th>3.15 (June 2014)</th>
<th>5.1 (May 2019)</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>sloccount net/core net/ipv{4,6}</td>
<td>131,550</td>
<td>174,195</td>
<td>+32%</td>
</tr>
<tr>
<td>git log --oneline net/core net/ipv{4,6}</td>
<td>wc -l</td>
<td>10,645</td>
<td>18,004</td>
</tr>
<tr>
<td>sloccount drivers/net/ethernet</td>
<td>729,678</td>
<td>1,295,169</td>
<td>+77%</td>
</tr>
<tr>
<td>git log --oneline drivers/net/ethernet</td>
<td>wc -l</td>
<td>7,320</td>
<td>27,244</td>
</tr>
<tr>
<td>sloccount drivers/block</td>
<td>90,740</td>
<td>74,149</td>
<td>-18%</td>
</tr>
<tr>
<td>git log --oneline drivers/block</td>
<td>wc -l</td>
<td>4,238</td>
<td>6,299</td>
</tr>
</tbody>
</table>
### TL;DR: Booting is complicated

<table>
<thead>
<tr>
<th>Then/Now</th>
<th>Now/Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysFW/BIOS is an OS</td>
<td>Let Linux Do It</td>
</tr>
</tbody>
</table>
The Solution: Let Linux Do It

<table>
<thead>
<tr>
<th>Then/Now</th>
<th>Now/Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysFW/BIOS is an OS</td>
<td>Let Linux Do It</td>
</tr>
<tr>
<td>Opaque</td>
<td>Open, well-understood at FB</td>
</tr>
</tbody>
</table>
The Solution: Let Linux Do It

<table>
<thead>
<tr>
<th>Then/Now</th>
<th>Now/Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysFW/BIOS is an OS</td>
<td>Let Linux Do It</td>
</tr>
<tr>
<td>Opaque</td>
<td>Open, well-understood at FB</td>
</tr>
<tr>
<td>Proprietary ecosystem</td>
<td>Auditable, debuggable</td>
</tr>
</tbody>
</table>
The Solution: Let Linux Do It

<table>
<thead>
<tr>
<th>Then/Now</th>
<th>Now/Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysFW/BIOS is an OS</td>
<td>Let Linux Do It</td>
</tr>
<tr>
<td>Opaque</td>
<td>Open, well-understood at FB</td>
</tr>
<tr>
<td>Proprietary ecosystem</td>
<td>Auditable, debuggable</td>
</tr>
<tr>
<td>Vendor-specific tooling</td>
<td>Open-source tools</td>
</tr>
</tbody>
</table>
## The Solution: Let Linux Do It

<table>
<thead>
<tr>
<th>Then/Now</th>
<th>Now/Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysFW/BIOS is an OS</td>
<td>Let Linux Do It</td>
</tr>
<tr>
<td>Opaque</td>
<td>Open, well-understood at FB</td>
</tr>
<tr>
<td>Proprietary ecosystem</td>
<td>Auditable, debuggable</td>
</tr>
<tr>
<td>Vendor-specific tooling</td>
<td>Open-source tools</td>
</tr>
<tr>
<td>Product-specific</td>
<td>Portable, re-usable</td>
</tr>
</tbody>
</table>
FW-based runtime services (thar be dragons...)

- Run at highest privilege level
- Can access some (or all) resources
FW-based runtime services (thar be dragons...)

- Run at highest privilege level
- Can access some (or all) resources
- Run outside your OS/VM/container security model
  - x86: System Management Mode (SMM), UEFI Runtime
  - Non-x86: ARM Trusted Firmware, RISC-V M-Mode
  - Example: “Firmware First” HW error handlers
Example: HW Error Handling ("Firmware First")

- Code running merrily within the confines of OS/VM
Example: HW Error Handling (‘‘Firmware First’’)

• Uh oh, something just made a bit flip!
Example: HW Error Handling ("Firmware First")

- Uh oh, something just made a bit flip!
- Error detected by DRAM controller, CPU interrupted
Example: HW Error Handling ("Firmware First")

- Uh oh, something just made a bit flip!
- Error detected by DRAM controller, CPU interrupted
- CPU jumps to FW error handler
FW-based runtime services (thar be dragons...)

- You will (and may already) have a webserver in your FW
  - RESTful services used for remote configuration
  - Standard in UEFI spec as of June 2019

“New UEFI Specification, Version 2.8, captures evolution of platform firmware technology”

“The new specification adds support for REST. REST is a software architectural style that defines a set of constraints to be used for creating Web services. RESTful Web services provide interoperability between computer systems on the internet.”

FW-based runtime services (thar be dragons...)

• Some FW runs out-of-band on a separate microcontroller
  • Intel Management Engine (ME)
  • AMD Platform Security Processor (PSP)
  • Qualcomm Integrated Management Controller (IMC)
  • ... various other management controllers

• They often have direct access to HW resources
  • Including DRAM controller, PCIe bus, etc.
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- Add features + tools as needed
- Debug, build, deploy on our schedule
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- Add features + tools as needed
- Debug, build, deploy on our schedule
- Flexible security architecture
To recap

• Put a kernel+initramfs in boot ROM
• Do minimal silicon init and jump to Linux as soon as possible
• Use Linux to boot Linux
• Production-quality drivers, networking
• Add features + tools as needed
• Debug, build, deploy on our schedule
• Flexible security architecture
• Boot in seconds, not minutes
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- Add features + tools as needed
- Debug, build, deploy on our schedule
- Flexible security architecture
- Boot in seconds, not minutes
- Bring modern, open-source development to the firmware
Present

- Coreboot Project ([coreboot.org](http://coreboot.org))
  - Celebrating its 20th anniversary in 2019
  - Member of Software Freedom Conservancy
- LinuxBoot Project ([linuxboot.org](http://linuxboot.org))
  - Linux Foundation member project as of Jan. 2018
- Open Compute Project ([opencompute.org](http://opencompute.org))
  - Industry effort to develop open HW designs for hyperscale deployment
  - Founded in 2011
    - 100+ members including FB, Google, Tencent, Alibaba, MS, Intel, etc.
  - “Open System Firmware” (OSF) initiative started in March 2018
Present

- **Deployment**
  - In progress at Facebook
    - “Reinventing Facebook’s data center network with F16, Minipack”
  - Google also working on deployment
    - Google around for their LinuxBoot talks
  - Sesame Open Hardware
    - Selling refurbished OCP servers preloaded with LinuxBoot
    - Great for SMBs, universities, etc.
  - Probably more that we’re not aware of...
Future

- Continue development on existing and upcoming OCP platforms
- **March 2021**
  - Tentative cutover date for “OCP Accepted” criteria

OCP Open System Firmware

*Because if you don't own your firmware your firmware owns you!*
Hands on with LinuxBoot

Andrea Barberio
Host Provisioning / LinuxBoot team @ Facebook infrastructure

• Host provisioning infrastructure
• LinuxBoot development and production
LinuxBoot architecture @ FB

System Firmware on boot ROM

- **coreboot**: silicon and DRAM initialization
- **Linux**: device drivers, network stack, multi-user multi-tasking environment
- **u-root/systemboot**: userspace tools and bootloader
- **Target OS**: CentOS

source: systemboot.org [CC BY-SA 4.0], the CentOS project
Boot entries

- Boot entries and their order are stored in VPD variables
- Value in JSON format. Example:

  • **Boot0000**={
      "type": "netboot",
      "method": "dhcpv6",
      "mac": "00:fa:ce:b0:0c:00"
      "debug_on_failure": true,
    }

  • **Boot0002**={
      "type": "localboot",
      "method": "path",
      "device_guid": "abcdef...",
      "kernel": "/path/to/kernel",
      "ramfs": "/boot/ramfs.cpio.xz",
    }
In production: testing (1)

- We run LinuxBoot in production[1]. How to test it before hitting prod?
- **Unit tests:** coreboot, u-root, systemboot have unit and integration tests running in public CIs
- but this is not enough

In production: testing (2)

- Continuous end-to-end testing: reflash and run continuous tests (e.g. stress testing, dependencies)
- On-demand end-to-end testing: every diff triggers a test run, on real hardware and VMs. Results are reported back on the code review tool
- We created an open source system testing framework called ConTest (for continuous testing), soon on GitHub
In production: troubleshooting

• Manual troubleshooting using Linux programs
• Automated troubleshooting and remediations with the checker framework
  • runs through a list of checks and remediations at the time of the failure
• Example: fixmynetboot
  • it runs when netboot fails. It checks interface status, addresses, link speed, DHCP, HTTP, and so on.
  • it can try to remediate
  • if everything fails, it can drop into an emergency shell for manual troubleshooting
• Imagine doing the same for disks, etc
Live demo

- Using a virtual machine, for simplicity
  - built using github.com/insomniacslk/osf-build
- Showing boot flow (coreboot, Linux, u-root, systemboot)
- Boot sequence and boot variables
- Network booting via DHCPv6 and HTTPS
- Troubleshooting stories and automated troubleshooting
- Disk booting with either Grub or local boot parameters
Thanks!

Resources
• coreboot.org
• linuxboot.org
• u-root.tk
• systemboot.org
• opencompute.org

Contact
• Slack: http://slack.u-root.com
• IRC: #coreboot on freenode

Participate