Booting Linux Fast & Fancy

The Barebox Bootloader

Barebox (formerly known as u-boot-v2) is a bootloader that inherits the best of U-Boot and the Linux kernel. The size and look-and-feel of u-boot, with driver model and lots of design concepts from the kernel.

You can find the latest of our timed releases in the download section. Check the repository for the most recent developments. Feel free to subscribe to the mailing list. Note also the slides from the ELCE09 talk by Sascha Hauer, creator of barebox.

If you search for a kernel hacker friendly bootloader, read on!

Last change: Mon Oct 25 11:09:50 2010
© barebox.org
http://www.barebox.org

Embedded Linux Conference Europe
Cambridge, 2010-10-28
Robert Schwebel <r.schwebel@pengutronix.de>

Slide 1 - http://www.pengutronix.de - 01/11/2010
Motivation: Booting Linux Fast & Fancy I

- User experience becomes more important:
Motivation: Booting Linux Fast & Fancy II

- Industrial devices don't look like computers

- And the should'nt boot slowly like computers...
Motivation: Booting Linux Fast & Fancy III

- Automotive devices have fast-boot requirements
- Example from a project:

  Answering CAN messages in < 200 ms after power-on!
Motivation: Booting Linux Fast & Fancy IV

- Video: Start sequence on Vortex DX (800 MHz x86)
What can we do to avoid this?
Barebox: Project History

- 2007 / u-boot-v2-rc1: Forked from U-Boot, as a technology study under the “U-Boot-v2” name

- 2009 / barebox-2009.12.0: Renamed to barebox, with its own infrastructure

- 2010 / barebox-2010.10.0 20 releases up to now

- Timed releases: about once per month

- Maintenance releases: on demand
Barebox: Development Resources

- Website:  
  http://www.barebox.org

- GIT Server:  
  http://git.pengutronix.de/?p=barebox.git

  next branch:  
  accumulates new features

  master branch:  
  next is merged into master after release

- Mailing List:  
  http://lists.infradead.org/mailman/listinfo/barebox/
Barebox: Development Speed

- Commit History:
  - 2008: 364
  - 2009: 583
  - 2010: 648 (until now)
## Barebox: CPU Architectures

- **Supported Hardware:**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm</td>
<td>at91, ep93, i.MX, netX, nomadik, omap, s3c24xx, stm</td>
</tr>
<tr>
<td>blackfin</td>
<td></td>
</tr>
<tr>
<td>m68k</td>
<td>mcfv4e</td>
</tr>
<tr>
<td>ppc</td>
<td>mpc5xxx</td>
</tr>
<tr>
<td>sandbox</td>
<td>linux</td>
</tr>
<tr>
<td>x86</td>
<td>bios based</td>
</tr>
</tbody>
</table>
Barebox - All Features on One Slide

- Build system: Kconfig, Kbuild
- Boot media: linux16, nand, ubi, sd
- Data Transport: DFU, Kermit, X,/Y/Z-Modem, tftp
- Graphics: Framebuffer, splash screen
- Filesystem: cd, ls, cp, saveenv/loadenv, mount, partitions
- Tools: crc, edit, gpio, unlzo
- User interaction: login, menu
- Drivers: i2c, mfd, flash, serial, spi, usb host+device
- Modules: insmod, lsmod
- Memory: meminfo, memtest, md, mw
- Network: ipv4, dhcp, netconsole, tftp, rarp, ping, nfs, dns
Booting Linux Fast

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
- /sbin/init

Optimize Hardware (Electronics)
Booting Linux Fast

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
  /sbin/init

Select CPUs optimized for fastboot (i.e. MX25)
Booting Linux Fast

1. Power-controller releases reset line
2. ROM bootloader starts running
3. Fetch boot block from NAND / SD card
4. Execute first boot code
5. Initialize hardware
6. Fetch Linux kernel from NAND / SD card
7. Execute Linux
8. Extract compressed image
9. Kernel boots, initializes hardware
10. /sbin/init

Done by firmware, cannot be tuned, usually.
Booting Linux Fast

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
- /sbin/init

First place we can do something in software
Booting Linux Fast

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
  - /sbin/init

Do only what's absolutely necessary.
Tune clocks & timings.
Booting Linux Fast

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
- /sbin/init

Async read and decompression tricks
Booting Linux Fast

Power-controller releases reset line
ROM bootloader starts running
Fetch boot block from NAND / SD card
Execute first boot code
Initialize hardware
Fetch Linux kernel from NAND / SD card
Execute Linux
Extract compressed image
Kernel boots, initializes hardware
/sbin/init

Depending on CPU, use uncompressed Image or zImage
Booting Linux Fast

Power-controller releases reset line
ROM bootloader starts running
 Fetch boot block from NAND / SD card
 Execute first boot code
 Initialize hardware
 Fetch Linux kernel from NAND / SD card
 Execute Linux
 Extract compressed image
 Kernel boots, initializes hardware
 /sbin/init

Minimized kernel, all the tricks from elinux.org wiki
Booting Linux Fast

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware

/sbin/init

Depending on use case, use initramfs or real rootfs (slower)
Booting Linux Fast

Reset released

First Bootloader Instruction

Bootloader starts kernel

Example: Freescale i.MX35, ARM1136EJ-S, 532 MHz

336 ms
Booting Linux Fancy

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
- /sbin/init

Backlight off
Load splash
Show splash
Backlight on
**Booting Linux Fancy**

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
  `/sbin/init`

---

*Make sure Framebuffer has fixed address between bootloader and kernel.

*No re-init!*
Booting Linux Fancy

- Power-controller releases reset line
- ROM bootloader starts running
- Fetch boot block from NAND / SD card
- Execute first boot code
- Initialize hardware
- Fetch Linux kernel from NAND / SD card
- Execute Linux
- Extract compressed image
- Kernel boots, initializes hardware
- /sbin/init

Cross fading with overlay
Booting Linux Fancy

- Video: Start sequence on phyCORE-i.MX27
  http://www.youtube.com/watch?v=F5Cbu1sO4D8

- Video: Start sequence on NESCO (MX27, ARM926, 400 MHz)
  http://www.youtube.com/watch?v=2FZl_7u9nBE

- Demo: Start sequence on CUPID (MX35, ARM1136, 532 MHz)
If fast is not fast enough ...

- We do not reach the 200 ms limit of the automotive guys

- BTCS: *Boot Time Critical Services*  
  (originally inspired by Freescale, now implemented with mainline focus)

- Idea:
  
  - Set aside some memory
  - Register a “poller” in Barebox
  - Make sure memory is handed over to Linux
  - Poller ends up as a normal interrupt service routine in Linux

- We estimate to have CAN ready after about < 100 ms

- No numbers yet, measurement hardware is still under construction...

- Downside: “bare metal” stack
Thanks for Listening - Questions?
Do we need a Bootloader at all?

- Alternative: use Linux to boot Linux (less code duplication) (See John's talk yesterday)

- Booting from NAND: we need at least a pre-loader dealing with Bad Blocks / UBI

- ROM access routines to NAND and SD are unoptimized

- Barebox offers a (to kernel developers) well-known structure, where to put code in *if* it is necessary.

- Minimal porting effort, no parallel running code

- Even if we have linux-only booting in the future, Barebox can be scaled down to the minimum, for the first stage