

# U-Boot-v2

Sascha Hauer <s.hauer@pengutronix.de>  
Marc Kleine-Budde <m.kleine-budde@pengutronix.de>



CELF Embedded Linux Conference Europe  
Grenoble, 2009-10-16

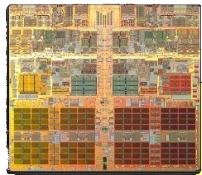


# U-Boot-v2: Agenda

- Bootloaders: What they do and why we (still) need them
- The U-Boot-v2 Project: Motivation for a fork
- Design Decisions: A Bootloader for Kernel Hackers
- Flow of Execution: From Power-On to the Kernel
- Sugar and Candies: Some Highlights
- Future: Perspective & Discussion



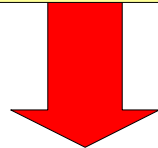
# Bootloaders: What they do ...



## Low Level Hardware Init

RAM, Flash, PLLs + Clocks, ...

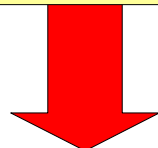
- BIOS (PC)
- Bootloader (SoC)



## Fetch Kernel(s) from Boot Medium

NOR-Flash, NAND-Flash, SD, USB, SATA, Network...

- BIOS (PC)
- Bootloader (SoC, PC)



## Start Kernel

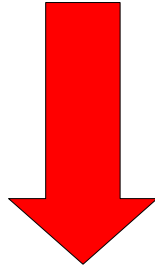
with kernel command line parameters

- Bootloader (SoC, PC)



# Requirements for Production Systems

- No interaction: power-on and boot
- No delays by the bootloader!



- The bootloader shall stay out of the way!
  - no selection screen
  - no nothing



# Requirements: Development & Maintenance

- **Stop** boot process in the bootloader:
  - key press on keyboard (PC)
  - serial console key or hardware button (embedded)
- **Choose** between pre-existing **kernels**
- Be able to edit kernel **location** - where to boot from (flash partition, disk partition, tftp location, ...)
- Change kernel **command line**
- Make changes **persistent** (change - store - boot with new config)

```
GNU GRUB version 0.97 (638K lower / 268832K upper memory)

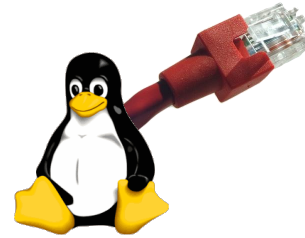
Debian GNU/Linux, kernel 2.6.26-2-686
Debian GNU/Linux, kernel 2.6.26-2-686 (single-user mode)

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, or 'c' for a command-line.
```



# Requirements on Embedded Systems

- TFTP booting the kernel  
(for quick development cycle)



- Redundancy Boot  
(start watchdog, boot,  
boot other kernel on startup-failure)



- Hardware testing environment:
  - have register access from a commandline while kernel was not ported to a new platform yet
  - have a non-complex environment for hardware people to test prototypes



# Why do we need a Bootloader at all?

- Alternative: use Linux to boot Linux
- Booting from NAND: we need at least a pre-loader
- Boot Time
  - kexec needs about 7 s add-on-time
  - can be avoided by directly starting the production kernel
- U-Boot-v2 is minimal porting effort until the developer is able to see something on the commandline and has fancy debug possibilities
- high scalability: even if we have linux-only booting in the future, U-Boot-v2 can be scaled down to the minimum



# U-Boot-v2: Agenda

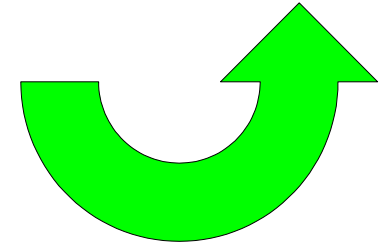
- Bootloaders: What they do and why we (still) need them
- **The U-Boot-v2 Project: Motivation for a fork**
- Design Decisions: A Bootloader for Kernel Hackers
- Flow of Execution: From Power-On to the Kernel
- Sugar and Candies: Some Highlights
- Future: Perspective & Discussion





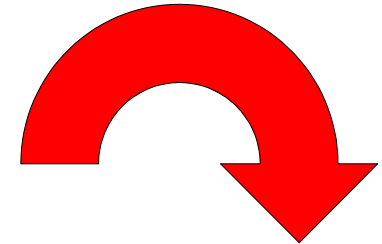
# The U-Boot-v2 Project: Motivation for a fork

- Pro:
  - „Das U-Boot“ is a great bootloader for SoC type Linux systems!
  - **Multi Platform** design, runs on ARM, MIPS, PowerPC, Blackfin etc.
  - Good **user experience** (for us embedded people)
  - High level of **configurability** (environment + saveenv)
  - “True” **Open Source** project (no hidden development, public git + list)



# The U-Boot-v2 Project: Motivation for a fork

- Contra:
  - Usage of U-Boot requires deep board knowledge
  - abuse of the **environment** for **scripting**
  - **no clean driver model** and multi instance concepts
  - **filesystem support** is quite inconvenient (**fatfs**, **ext2load**, ...)
  - **Hard to configure**: many macros have to be edited by hand to configure the features of U-Boot
  - **“Must not break existing boards”** policy (makes it hard to change designs)



# The U-Boot-v2 Project: Motivation for a fork

- U-Boot-v2 was started as a technology study
- Idea: How can the U-Boot principle be improved when...
  - ... we take **proven concepts** from the Linux kernel
  - ... we **"think POSIX"**
  - ... we **don't** have to **care about stability** of 10 year old platforms
- Why can't a bootloader feel more like Linux?



# U-Boot-v2: Agenda

- Bootloaders: What they do and why we (still) need them
- The U-Boot-v2 Project: Motivation for a fork
- **Design Decisions: A Bootloader for Kernel Hackers**
- Flow of Execution: From Power-On to the Kernel
- Sugar and Candies: Some Highlights
- Future: Perspective & Discussion



# Design Decisions: A Bootloader for Kernel Hackers

- Some design decisions we made for U-Boot-v2:
  - more abstraction  
(devices instead of direct memory access + special knowledge)
  - multi instance, driver model from kernel  
(no global variables, “`ethaddr`”, “`eth1addr`” etc. any more)
  - file system abstraction (but still simple)
  - frameworks instead of multiple-drivers-with-(almost-)same-api
  - “usual” commands: `rm`, `cp`, `mount`, ...



# Design Decisions: A Bootloader for Kernel Hackers

- Some design decisions we made for U-Boot-v2:
  - **scripts are scripts**, no “runnable environment variables”
  - **environment** consists of ramdisk + tar archive for persistence
  - make **local variables** possible (saveenv saves everything in v1)
  - **KBuild + Kconfig** (configuration, parallel build)
  - **clocksource** model taken from Linux
  - kernel **coding style**
  - “**best of U-Boot and Linux**”



# Design Decisions: A Bootloader for Kernel Hackers

- But:
  - U-Boot should **not increase in binary size**
  - Code **size** is still **more important** than performance and feature completeness.



# U-Boot-v2: Agenda

- Bootloaders: What they do and why we (still) need them
- The U-Boot-v2 Project: Motivation for a fork
- Design Decisions: A Bootloader for Kernel Hackers
- **Flow of Execution: From Power-On to the Kernel**
- Sugar and Candies: Some Highlights
- Future: Perspective & Discussion





# U-Boot-v2: Startup sequence

- Board specific lowlevel init
- Relocation to RAM if necessary
- Common entry point start\_uboot()
- Initcall sequence: First subsystems, then drivers
- Mount ramfs to /
- Mount devfs to /dev
- Load environment to /env
- Execute /env/bin/init script

# “Hello World” in U-Boot-v2

- Here is a typical startup from U-Boot-v2:

```
U-Boot 2.0.0-rc10-00262-g7c693db-dirty (Sep 10 2009 - 11:16:13)
```

```
Board: Phytex phyCard-i.MX27
```

```
NAND device: Manufacturer ID: 0x20, Chip ID: 0x36 (ST Micro NAND 64MiB  
1,8V 8-bit)
```

```
Malloc space: 0xa7a00000 -> 0xa7f00000 (size 5 MB)
```

```
Stack space : 0xa79f8000 -> 0xa7a00000 (size 32 kB)
```

```
running /env/bin/init...
```

```
Hit any key to stop autoboot: 3
```

```
type update_kernel nand|nor [<imagenam>] to update kernel into flash
```

```
type update_root nand|nor [<imagenam>] to update rootfs into flash
```

```
uboot: /
```



# File System

- During startup, a RAM filesystem is mounted to /
- A device filesystem is mounted to /dev
- The environment is copied to /env
- At the prompt, the well known commands like 'ls', 'rm', 'cp' work the way we are used to:

```
uboot:/ ls  
.  
..      dev      env
```

# Devices

- Drivers can create device nodes under `/dev` which can be accessed like normal files:

```
uboot:/ ls /dev/  
zero          defaultenv    mem           nand0         ram0  
phy0          self_raw      self0         env_raw       env0
```



# Accessing Memory

- `/dev/mem` is a special file which represents the whole address space and is the device the memory display (`md`) command normally works on
- `md` internally works by opening `/dev/mem` and lseeking to the desired position and reading the contents:

```
uboot:/ md 0x0
00000000: e59ff00c e51ff11c e51ff11c e51ff11c .....
00000010: e51ff11c d8000000 e51ff120 e51ff120 .....
00000020: 000000a0 00000000 00000000 00000000 .....
00000030: e51ff018 00000000 00000000 00000000 .....
00000040: 0000000a 584d2e69 43003732 7279706f ....i.MX27.Copyr
00000050: 74686769 29632820 30303220 72462035 ight (c) 2005 Fr
00000060: 63736565 20656c61 2e636e49 6c6c4120 eescale Inc. All
00000070: 67697220 20737468 65736572 64657672 rights reserved
00000080: 0000002e 00000000 00000000 00000000 .....
00000090: 00000317 000004b9 000004e5 00000000 .....
000000a0: e321f0d3 e59fd008 e59f0008 e12fff10 ..!...../
000000b0: ea000001 fffffb79c 000000f1 e321f0d3 .....!
000000c0: e59fd014 e59f0014 e3a0106c e5c01000 .....1.....
000000d0: e59f000c e12fff10 eafffff7 fffffb79c ...../.....
```



# Accessing Other Devices

- While `/dev/mem` is the default “file” for the memory commands, it can be changed:

```
uboot:/ md -w -s /dev/phy0
00000000: 1000 786d 0022 1613 01e1 45e1 0007 2001 ..mx".....E...
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

- This displays the contents of `/dev/phy0` (-s) in 16 bit wordsize (-w)



# Copying Things

- Not only the memory display/write (**md**, **mw**) commands work this way, but also **memcpy** and friends
- Copy memory at **<src>** of **<count>** bytes to **<dst>**:

```
uboot:/ memcpy
Usage: memcpy [OPTIONS] <src> <dst> <count>

options:
  -b, -w, -l    use byte, halfword, or word accesses
  -s <file>     source file (default /dev/mem)
  -d <file>     destination file (default /dev/mem)
```

- This would copy a 1k chunk from **/dev/nand0** from offset 1 M to the file 'nand':

```
uboot:/ memcpy -s /dev/nand0 -d nand 1M 0x0 1k
```



# Partitioning

- Device files can be partitioned to get a convenient access to flashes and to get a consistent partition layout between U-Boot and Linux:

```
uboot:/ addpart nor0 256k(u-boot) ,128k(u-boot-env) ,2M(kernel) ,-(rootfs)
uboot:/ ls -l /dev/nor0*
crw----- 31064064 /dev/nor0.root
crw----- 2097152 /dev/nor0.kernel
crw----- 131072 /dev/nor0.ubootenv
crw----- 262144 /dev/nor0.uboot
crw----- 33554432 /dev/nor0
```

- The partition description for the “addpart” command is the same as the Linux mtd layer uses for command line partitioning, so this string can be directly given to the Kernel





# Environment

- In U-Boot-v1: environment can be made persistent with “saveenv”
- In U-Boot-v2, “saveenv” is different:
  - Save the contents of `/env` to `/dev/env0`
  - You can think of saveenv/loadenv as a simple tar command
  - `/env` and `/dev/env0` are only defaults which can be changed on the command line so that it is possible to have multiple environment sectors.
  - The environment can be changed by editing the files under `/env`, using the internal editor (edit)
  - The changes are then made persistent with saveenv
- Due to the file nature of the environment it is of course possible to store arbitrary files in the environment, for example splash images.



# Device Variables

- Design criterium: "avoid magic variables"
- Introduction of "device variables": `eth0.ipaddr`
- Device variables can be displayed using the `devinfo` command:

```
uboot:/ devinfo eth0
base   : 0x00000000
size   : 0x00000000
driver: none

Parameters:
        ipaddr = 192.168.24.26
        ethaddr = 00:50:c2:a5:bb:87
        gateway = 192.168.1.1
        netmask = 255.255.0.0
        serverip = 192.168.23.2
```

- The device variables can be used like any other variable:

```
uboot:/ eth0.serverip=192.168.23.123
uboot:/ echo $eth0.serverip
192.168.23.123
```



# Getting the Kernel via TFTP

- Most important networking commands:
  - dhcp (configure the network adapter)
  - tftp (transfer files via tftp)
- tftp normally writes to files but can be used to write directly to flash:

```
uboot:/ erase /dev/nand0.kernel.bb
uboot:/ dhcp
phy0: Link is up - 100/Full
BOOTP broadcast 1
DHCP client bound to address 192.168.24.26
uboot:/ tftp zImage-pca100 /dev/nand0.kernel.bb
phy0: Link is up - 100/Full
TFTP from server 192.168.23.2; our IP address is 192.168.24.26
Filename 'zImage-pca100'.
Loading: #####
          #####
          #####
done
Bytes transferred = 1815016 (1bb1e8 hex)
```



# Build System

- Building should be familiar to kernel hackers:

```
# export ARCH=arm
# export CROSS_COMPILE=arm-linux-
# make pcm038_defconfig
# make menuconfig
# make
```



# U-Boot-v2: Agenda

- Bootloaders: What they do and why we (still) need them
- The U-Boot-v2 Project: Motivation for a fork
- Design Decisions: A Bootloader for Kernel Hackers
- Flow of Execution: From Power-On to the Kernel
- **Sugar and Candies: Some Highlights**
- Future: Perspective & Discussion



# Sandbox

- U-Boot-v2 can be built as a normal Linux binary (inspired by user mode linux)
- Features:
  - Working on U-Boot-v2 without real hardware
  - Run U-Boot under gdb
  - Networking using a tap device
- To compile the sandbox: pass **ARCH=sandbox** while compiling



# Sugar & Candies: Highlights

- Minimum port (2nd stage):
  - v2 can be used as a payload of an existing bootloader
  - getting the features without having to do the hardware init part first
- Quickboot: Booting Linux Fast & Fancy:
  - U-Boot-v2 brings up splash screen in < 0.5 s
  - Kernel starts in < 3 s (i.MX27 @ 400 MHz)
  - Soft fading into Qt 4.5 application
- Integrated editor:
  - “`edit /path/to/file`” opens a full screen editor (even on serial line)
  - `ctrl-d` (save) / `ctrl-c` (cancel)



# Sugar & Candies: Highlights

- “1-image-starts-everywhere”
- MMU support
- USB host/device support
- DFU (Device firmware update) support
- Module support






# U-Boot-v2: Agenda

- Bootloaders: What they do and why we (still) need them
- The U-Boot-v2 Project: Motivation for a fork
- Design Decisions: A Bootloader for Kernel Hackers
- Flow of Execution: From Power-On to the Kernel
- Sugar and Candies: Some Highlights
- **Future: Perspective & Discussion**



# Perspective & Discussion

- Do we need yet-another-bootloader?



Log in / create account

page discussion view source history

eLinux.org - Embedded Linux Wiki

## Bootloader

Briefly, a bootloader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to the operating system kernel software (such as the Hurd or the Linux). The kernel, in turn, initializes the rest of the operating system (e.g. GNU).

### List of bootloaders

Bootloader	ARM	BFIN	MIPS	PPC	SH	x86	remarks
<a href="#">APEX</a>	y	-	?	?	?	?	
<a href="#">Blob</a>	y	-	-	-	-	-	
<a href="#">CFE</a>	-	-	y	-	-	-	for specific BroadCom chipsets
<a href="#">coreboot (LinuxBIOS)</a>	-	-	-	-	-	y	Conference talk by Peter Stuge at Embedded Linux Conference Europe 2008, <a href="#">video</a>
<a href="#">Das U-Boot</a>	y	y	y	y	y	y	allows networked setup
<a href="#">U-Boot-v2</a>	y	y	-	y	-	-	allows networked setup, integrated editor and scripting
<a href="#">Grub</a>	?	-	?	y	?	y	
<a href="#">Lilo</a>	-	-	-	-	-	y	x86 only, requires nasm to build
<a href="#">MicroMonitor</a>	y	y	?	y	y	?	
<a href="#">PMON 2000</a>	-	-	y	-	-	-	
<a href="#">RedBoot</a>	y	?	y	y	y	y	allows networked setup
<a href="#">Syslinux</a>	-	-	-	-	-	y	variants (including) isolinux are very flexible for booting x86
<a href="#">Yaboot</a>	-	-	-	y	-	-	
<a href="#">YAMON</a>	-	-	y	-	-	-	

Legend: ?:Unknown -:Not supported

navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Volunteering
- Popular Pages

search

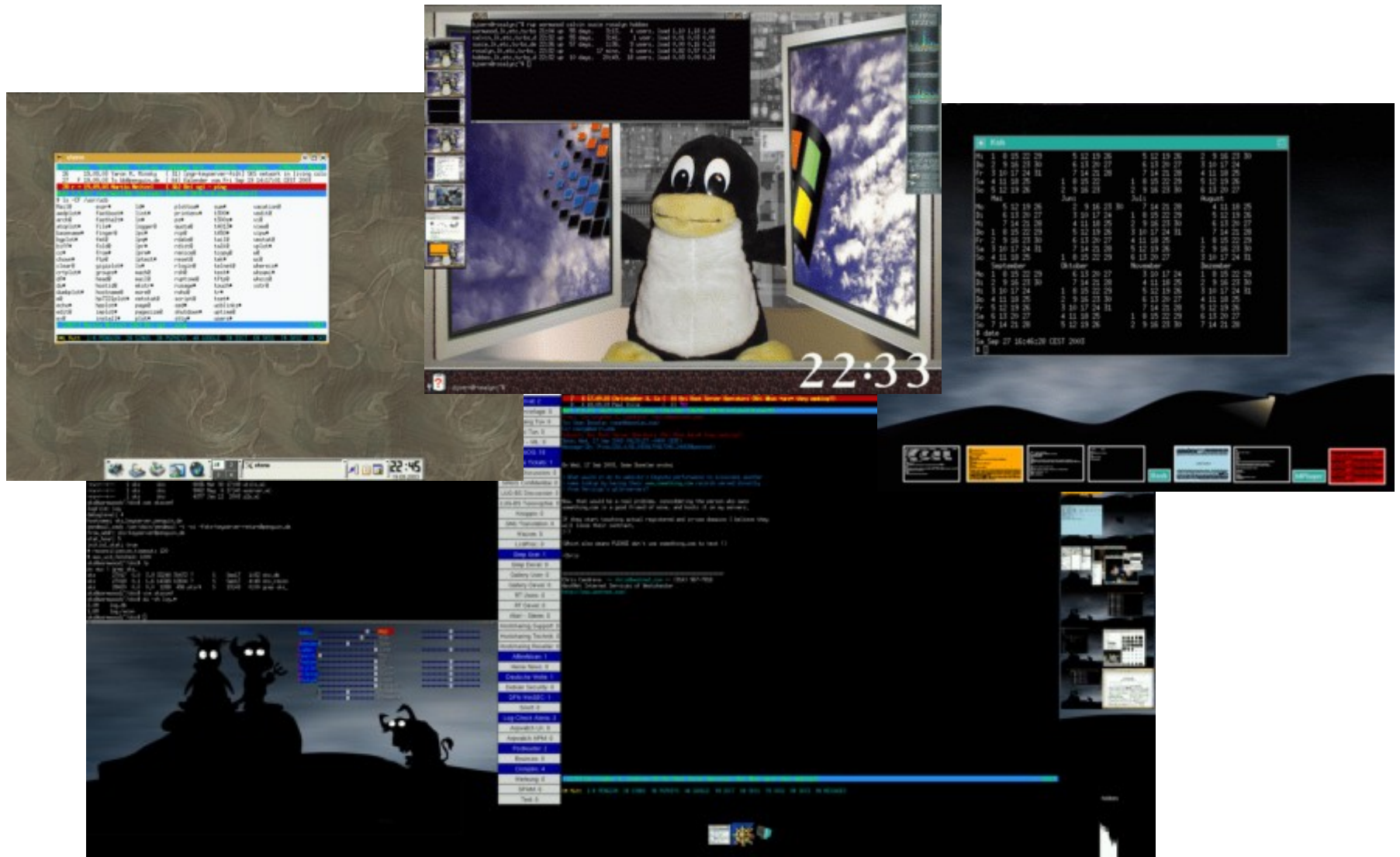
Go Search

toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link



# Thanks for Listening - Questions?



# Literature

- Web Site for U-Boot-v2:  
<http://www.pengutronix.de/software/u-boot/v2/>
- Source Code:  
`git clone git://www.denx.de/git/u-boot-v2.git`
- Mailing List:  
<http://lists.denx.de/mailman/listinfo/u-boot>

