

The Universal Boot Loader ("Das U-Boot")

Table of contents:

- 1. Abstract
 - ◆ 1.1. Introduction
 - ◆ 1.2. History
 - ◆ 1.3. Supported Hardware
 - ◆ 1.4. Design Principles
 - ◆ 1.5. User Interface
 - ◆ 1.6. Basic Command Set
 - ◆ 1.7. Advanced Commands
 - ◇ 1.7.1. Logbuffer Manipulation Commands
 - ◇ 1.7.2. Bedbug Embedded Debugger Commands
 - ◇ 1.7.3. POST – Hardware Diagnose Commands
 - ◆ 1.8. Environment Variables
 - ◆ 1.9. Boot Options
 - ◆ 1.10. Command Interpreters
 - ◆ 1.11. Standalone Programs
 - ◆ 1.12. Special Features
 - ◇ 1.12.1. Bitmap and Splash Screen Support
 - ◇ 1.12.2. Boot Count Limit
 - ◇ 1.12.3. Keyboard Support
 - ◇ 1.12.4. Automatic Updates
 - ◆ 1.13. Resources, Summary

1. Abstract

This is a presentation about U-Boot, the universal boot loader.

The online version of this presentation is available at <http://www.denx.de/twiki/bin/view/UBootdoc/Presentation> or, as a PDF file, at <http://www.denx.de/twiki/publish/UBootdoc/UBootdoc.pdf>

This document was generated at Thu Jun 3 15:25:09 2004.

1.1. Introduction

- The "Universal Bootloader" ("*Das U-Boot*") is a monitor program.
- Free Software: full source code under GPL
- hosted on SourceForge: <http://sourceforge.net/projects/u-boot>
- production quality: used as default boot loader by several board vendors
- portable and easy to port and to debug
- many supported architectures: PPC, ARM, MIPS, x86, m68k, NIOS, Microblaze
- more than 216 boards supported by public source tree
- many, many features

1.2. History

- Oct 22, 1999: fadsrom – Dan Malek => PPCBoot rev. 1.1
- Dec 18, 1999: 8xxrom-0.3.0 – Magnus Damm, Raphael Bossek => PPCBoot rev. 1.2
- Jul 07, 2000: Wolfgang Denk => PPCBoot rev. 1.3
- Jul 19, 2000: Wolfgang Denk => PPCBoot-0.4.1
first public version of PPCBoot
- Siemens PSE, Vienna: Development of a Bluetooth LAN Access Point with a MPC850 Processor that needed to be able to boot over Ethernet => first commercial sponsor
- Aug 08, 2000: PPCBoot rev. 1.4 = **PPCBoot-0.4.2**
(only PPC, only MPC8xx, 4 boards)
- Oct 01, 2000: added network support => PPCBoot-0.4.4
- Oct 01, 2000: Stefan Roese: add support for IBM PPC401/403/405GP processors => PPCBoot-0.5.1
- Nov 16, 2000: Murray Jensen: add support for MPC8260 => PPCBoot-0.6.2
- Nov 20, 2000: Rob Taylor: add support for MPC8240 => PPCBoot-0.6.3
- End 2000: PPCBoot-0.7.1 (MPC8xx, MPC8240, MPC8260, PPC401/403/405GP; 27 boards)
- End 2001: PPCBoot-1.1.3 (MPC8xx, MPC8240, MPC8260, 7xx, 74xx, IBM 4xx, 63 boards)
- Mar 2002: SYSGO: split ARMBoot project, separate (incompatible) source tree
- Jul 2002: begin merging with ARMBoot tree
- Nov 2002: PPCBoot-2.0.0 (last release of PPCBoot)
(PPC: 8xx, 824x, 826x, 7xx, 74xx, 4xx; ARM: StrongARM, ARM7, ARM9, XScale; >106 boards)
=> Start **U-Boot** project: PPCBoot-2.0.0 = U-Boot-0.1.0
- Nov 2002: x86 support
- Mar 2003: MIPS32
- Apr 2003: MIPS64
- Oct 2003: Altera NIOS-32
- Dec 2003: Coldfire
- Apr 2004: Microblaze
- today (31 May 2004): U-Boot-1.1.2
(PPC: 5xx, 5xxx, 8xx, 824x, 826x, 85xx, 7xx, 74xx, 4xx;
ARM: StrongARM, ARM720T, ARM92xT, S3C44B0, AT91RM9200, XScale;
x86: SC520; m68k: Coldfire; MIPS32: 4Kc, Au1x00; MIPS64: 5Kc; NIOS32; Microblaze;
>216 boards in public tree; many more not submitted back)
- several board manufacturers use U-Boot as default firmware on some or all of their boards

1.3. Supported Hardware

Architecture	Processor	Number of Boards
PPC	5xx	2
	5xxx	6
	8xx	71
	824x	15
	826x	26
	85xx	3

	7xx/74xx	11
	4xx	38
ARM	StrongARM	5
	ARM720T	3
	ARM92xT	11
	S3C44B0	1
	AT91RM9200	1
	XScale	8
x86	SC520	2
m68k	Coldfire	2
MIPS32	4Kc	2
	Au1x00	3
MIPS64	5Kc	1
NIOS32		3
Microblaze		1

1.4. Design Principles

- easy to port to new architectures, new processors, and new boards
- easy to debug: serial console output as soon as possible
- features and commands configurable
- as small as possible
- as reliable as possible

1.5. User Interface

U-Boot uses a simple command line interface (CLI), usually over a serial console port.

Two different command interpreters are available:

- simple CLI
- Bourne compatible shell (HUSH shell from Busybox)

Configuration parameters and commands / command sequences (scripts !) can be stored in "environment variables" which can be saved to non-volatile storage (flash, EEPROM, NVRAM, etc.)

1.6. Basic Command Set

- **Information Commands**
 - ◆ bdfinfo – print Board Info structure
 - ◆ coninfo – print console devices and informations
 - ◆ flinfo – print FLASH memory information
 - ◆ iminfo – print header information for application image
 - ◆ imls – list all images found in flash
 - ◆ help – print online help
- **Memory Commands**
 - ◆ base – print or set address offset
 - ◆ crc32 – checksum calculation
 - ◆ cmp – memory compare
 - ◆ cp – memory copy
 - ◆ md – memory display
 - ◆ mm – memory modify (auto-incrementing)
 - ◆ mtest – simple RAM test
 - ◆ mw – memory write (fill)
 - ◆ nm – memory modify (constant address)
 - ◆ loop – infinite loop on address range
- **Flash Memory Commands**
 - ◆ cp – memory copy (program flash)
 - ◆ flinfo – print FLASH memory information
 - ◆ erase – erase FLASH memory
 - ◆ protect – enable or disable FLASH write protection
- **Execution Control Commands**
 - ◆ autoscr – run script from memory
 - ◆ bootm – boot application image from memory
 - ◆ bootelf – Boot from an ELF image in memory
 - ◆ bootvx – Boot vxWorks from an ELF image
 - ◆ go – start application at address 'addr'
- **Network Commands**
 - ◆ bootp – boot image via network using BOOTP/TFTP protocol
 - ◆ cdpc – Perform Cisco Discovery Protocol network configuration
 - ◆ dhcp – invoke DHCP client to obtain IP/boot params
 - ◆ loadb – load binary file over serial line (kermit mode)
 - ◆ loads – load S-Record file over serial line
 - ◆ nfs – boot image via network using NFS protocol
 - ◆ ping – send ICMP ECHO_REQUEST to network host
 - ◆ rarpboot – boot image via network using RARP/TFTP protocol
 - ◆ tftpboot – boot image via network using TFTP protocol
- **Environment Variables Commands**
 - ◆ printenv – print environment variables
 - ◆ saveenv – save environment variables to persistent storage
 - ◆ askenv – get environment variables from stdin
 - ◆ setenv – set environment variables
 - ◆ run – run commands in an environment variable
 - ◆ bootd – boot default, i.e., run 'bootcmd'
- **Filesystem Support (FAT, cramfs, JFFS2, Reiser)**
 - ◆ chpart – change active partition

- ◆ fsinfo – print information about filesystems
- ◆ fsload – load binary file from a filesystem image
- ◆ ls – list files in a directory (default /)
- ◆ fatinfo – print information about filesystem
- ◆ fatls – list files in a directory (default /)
- ◆ fatload – load binary file from a dos filesystem
- ◆ nand – NAND flash sub-system
- ◆ reiserls – list files in a directory (default /)
- ◆ reiserload – load binary file from a Reiser filesystem
- **Special Commands**
 - ◆ i2c – I2C sub-system
 - ◆ doc – Disk-On-Chip sub-system
 - ◆ dtt – Digital Thermometer and Thermostat
 - ◆ eeprom – EEPROM sub-system
 - ◆ fpga – FPGA sub-system
 - ◆ ide – IDE sub-system
 - ◆ kgdb – enter gdb remote debug mode
 - ◆ diskboot – boot from IDE device
 - ◆ icache – enable or disable instruction cache
 - ◆ dcache – enable or disable data cache
 - ◆ diag – perform board diagnostics (*POST* code)
 - ◆ log – manipulate logbuffer
 - ◆ pci – list and access PCI Configuration Space
 - ◆ regdump – register dump commands
 - ◆ usb – USB sub-system
 - ◆ sspi – SPI utility commands
- **Miscellaneous Commands**
 - ◆ bmp – manipulate BMP image data
 - ◆ date – get/set/reset date & time
 - ◆ echo – echo args to console
 - ◆ exit – exit script
 - ◆ kbd – read keyboard status
 - ◆ in – read data from an IO port
 - ◆ out – write datum to IO port
 - ◆ reset – Perform RESET of the CPU
 - ◆ sleep – delay execution for some time
 - ◆ test – minimal test like /bin/sh
 - ◆ version – print monitor version
 - ◆ wd – check and set watchdog
 - ◆ ? – alias for 'help'

1.7. Advanced Commands

Some of the following commands depend on certain hardware features and may not be available on all boards.

1.7.1. Logbuffer Manipulation Commands

Use a reserved area of system memory as log buffer;
can be re-used by syslogd in Linux;
content will survive reset / warm boot

- log info – show pointer details
- log log reset – clear contents
- log log show – show contents
- log log append – append to the logbuffer
- setenv stdout log – redirect standard output to log buffer

Used for example to pass POST results to Linux application code or for post-mortem checking of the Linux system logs.

1.7.2. Bedbug Embedded Debugger Commands

- ds – disassemble memory
- as – assemble memory
- break – set or clear a breakpoint
- continue – continue from a breakpoint
- step – single step execution.
- next – single step execution, stepping over subroutines.
- where – Print the running stack.
- rdump – Show registers.

1.7.3. POST – Hardware Diagnose Commands

- cache – Cache test
- watchdog – Watchdog timer test
- i2c – I2C test
- rtc – RTC test
- memory – Memory test
- cpu – CPU test
- uart – UART test
- ethernet – ETHERNET test
- spi – SPI test
- usb – USB test
- spr – Special register test
- sysmon – SYSMON test
- dsp – DSP test

1.8. Environment Variables

Environment Variables (EV) can be used to configure the system and to store parameters for commands, and even commands and sequences of commands (simple scripts).

- *Board Configuration*

- baudrate, ethaddr, serial#, cpuclock
- **Startup Behaviour**
 - bootdelay, bootcmd
- **Network Parameters**
 - ipaddr, serverip, gatewayip, dnsip, netmask, hostname, rootpath, bootfile
- **Misc**
 - autoload (network lookup without download); autostart (start downloaded image); pram (reserve "protected RAM" area); silent (suppress console messages); verify (disable CRC32 checks); ...
- **User Defined**
 - => setenv name word1 word2 word3
 - => saveenv

The real power of EVs results from the fact that Unix shell like *variable expansion* is available. For example:

```
=> setenv ipaddr 192.168.3.71
=> setenv serverip 192.168.3.1
=> setenv netdev eth0
=> setenv hostname testbox
=> setenv rootpath /opt/eldk/ppc_8xx
=> setenv ramargs setenv bootargs root=/dev/ram rw
=> setenv nfsargs 'setenv bootargs root=/dev/nfs rw nfsroot=${serverip}:${rootpath}'
=> setenv addip 'setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:::${rootpath}'
=> setenv kernel_addr 40040000
=> setenv ramdisk_addr 40100000
=> setenv flash_ram 'run ramargs addip;bootm ${kernel_addr} ${ramdisk_addr}'
=> setenv flash_nfs 'run nfsargs addip;bootm ${kernel_addr}'
=> setenv net_nfs 'tftp 200000 ${bootfile};run nfsargs addip;bootm'
=> setenv net_ram 'tftp 200000 ${bootfile};run ramargs addip;bootm 200000 ${ramdisk_addr}'
```

Boot Kernel Image in flash with ramdisk in flash:

```
=> run flash_ram
```

Boot Kernel Image in flash with root filesystem over NFS:

```
=> run flash_nfs
```

Download Kernel Image over network and use root filesystem over NFS:

```
=> run flash_ram
```

Download Kernel Image over network with ramdisk in flash:

```
=> run flash_ram
```

Step by step:

```
=> run flash_nfs
```

<u>Step</u>	<u>Action</u>	<u>Result in bootargs</u>
run	setenv bootargs root=/dev/nfs rw	root=/dev/nfs rw

nfsargs	nfsroot=\${serverip}:\${rootpath}	nfsroot=192.168.3.1:/opt/eldk/ppc_8
run addip	setenv bootargs \${bootargs} ip=\${ipaddr}:\${serverip}:\ \${gatewayip}:\${netmask}:\ \${hostname}:\${netdev}:off	root=/dev/nfs rw nfsroot=192.168.3.1:/opt/eldk/ppc_8 ip=192.168.3.71:192.168.3.1:::\ testbox:eth0:off
bootm 40040000	boot Linux kernel	

Example: simple recovery strategy:

```
=> setenv bootcmd 'run flash_ram; setenv kernel_addr ${alt_kernel}; run flash_ram; run net_ram'
```

1.9. Boot Options

U-Boot supports many different ways to load and boot an image.

Basic command:

"bootm" – Boot Image in memory (RAM, flash)

Image:

Header + Payload

Header:

- ◇ Creation Timestamp
- ◇ Data Load Address
- ◇ Entry Point Address
- ◇ Data CRC Checksum
- ◇ Operating System
- ◇ CPU architecture
- ◇ Image Type
- ◇ Compression Type
- ◇ Image Name

Actions:

- ◇ test CPU architecture and OS
- ◇ test checksum (optional)
- ◇ if compressed, uncompress
- ◇ copy to load address
- ◇ prepare boot arguments
- ◇ start at entry point

Load image in memory:

- ◇ Serial Port: "loads" (S-Record), "loadb" (Kermit binary protocol)
- ◇ Ethernet: "tftp", "bootp", "dhcp", "nfs", ...

- ◇ Harddisk, CDROM: "ide read"
- ◇ CompactFlash card etc.: "ide read"
- ◇ USB Mass Storage Device: "usb read"
- ◇ SCSI Disk and CDROM: "scsi read"
- ◇ NAND flash with JFFS2 filesystem: "nboot"
- ◇ Disk on Chip: "doc read"
- ◇ PCI Bus: copy
- ◇ ...

Supported Filesystems (read-only):

- ◇ FAT
- ◇ Reiser
- ◇ JFFS2

1.10. Command Interpreters

Two command line interfaces:

- Simple (old) command interpreter:
 - ◆ sequential statements
 - ◆ statements separated by newline or ';'
 - ◇ "run cmd1; run cmd2; run cmd3"
 - will always run all three commands
 - ◇ "run cmd1 cmd2 cmd3"
 - will stop when a command fails
 - ◆ "scripts" (canned sequences of commands) available using "autoscr" command
- Hush Shell (from Busybox, see <http://www.busybox.net/>):
 - ◆ Bourne Shell compatible
 - ◆ (local) shell variables ("name=val"), (global) environment variables ("setenv name val")
 - ◆ Conditionals: "if ... then ... else ... fi"
 - ◆ Control loops: "for ... do ... done", "while ... do done", "until ... do ... done"
 - ◆ Control operators: && and | | (AND and OR lists: "command1 && command2")
 - ◆ real shell scripts
 - ◆ no functions
 - ◆ no command substitution
 - ◆ no backquotes

1.11. Standalone Programs

U-Boot can dynamically load independent software modules, called "standalone programs". Standalone programs have a standard C calling environment, and can use standard services like `printf()`, `malloc()`,

install_hdlr().

Used for:

- special test software that is used in bring-up but shall not be included with customer release
- software that is needed only occasionally
- code that performs special actions that were not foreseen (software updates)
- code that shall not be made available under GPL

Example:

```
#include <common.h>
#include <exports.h>

int hello_world (int argc, char *argv[])
{
    int i;
    app_startup(argv);
    printf ("Example expects ABI version %d\n", XF_VERSION);
    printf ("Actual U-Boot ABI version %d\n", (int)get_version());
    printf ("Hello World\n");
    printf ("argc = %d\n", argc);
    for (i=0; i<=argc; ++i) {
        printf ("argv[%d] = \"%s\"\n",
                i,
                argv[i] ? argv[i] : "<NULL>");
    }
    return (0);
}
```

Run:

```
=> tftp 40000 /tftpboot/hello_world.bin
...
=> go 40004 Hello World! This is a test.
## Starting application at 0x00040004 ...
Hello World
argc = 7
argv[0] = "40004"
argv[1] = "Hello"
argv[2] = "World!"
argv[3] = "This"
argv[4] = "is"
argv[5] = "a"
argv[6] = "test."
argv[7] = ""
## Application terminated, rc = 0x0
```

1.12. Special Features

- **Bitmap and Splash Screen Support**
- **Boot Count Limit**
- **Keyboard Support**
- **Automatic Updates**

1.12.1. Bitmap and Splash Screen Support

Problem: booting Linux and starting a GUI takes a couple of seconds, but the user expects to see something "immediately" after power-on.

Solution: display a static splash screen as soon as possible; available commands:

- `bmp info` – print Bitmap info
- `bmp display` – display bitmap image on screen
- `setenv splashimage addr` – display splash screen image at address *addr*

1.12.2. Boot Count Limit

The Open Source Development Labs Carrier Grade Linux Requirements Definition says:

CGL shall provide support for detecting a repeating reboot cycle due to recurring failures and will go to an offline state if this occurs.

U-Boot allows to run an arbitrary command in such a case:

- `bootcount` (EV) – number of reboots since power-on
- `bootlimit` (EV) – maximum number of reboot cycles
- `altbootcmd` (EV) – alternate boot action

1.12.3. Keyboard Support

Problem: make system behaviour dependent on keys pressed at power-on

Solution: in U-Boot you can define one or more keys or key combinations and commands which are executed when these keys are pressed at power-on:

- `"magic_keys"` (EV) – List of characters for keys; for example:

```
=> setenv magic_keys 0123CB*
```

- `"key_magic_?_"` (EV) – Key code or list of key codes for this action; scanned in the order as listed in `magic_keys`; for example:

```
=> setenv key_magic0 3a+3b
=> setenv key_magic1 3c
=> setenv key_magic2 4a
=> setenv key_magic3 4c+51
=> setenv key_magicC 3a+4a+5a
=> setenv key_magicB 52+53
=> setenv key_magic* 55+56
```

- `"key_cmd?"` (EV) – Action to be performed when corresponding key(s) are pressed; for example:

```
=> setenv key_cmd0 setenv bootdelay 10
=> setenv key_cmd1 setenv addcons 'console=ttyS0,${baudrate}'
```

```
=> setenv key_cmd2 setenv memtest on
=> setenv key_cmd3 setenv bootcmd run old_version
=> setenv key_cmdC setenv bootcmd run recovery
=> setenv key_cmdB setenv bootcmd run usb_update
=> setenv key_cmd* setenv bootcmd run demo_mode
```

Example: after a software update, the user can select to boot the old software version ("run old_version") by holding the keys with keycodes "4c" and "51" at power-on.

1.12.4. Automatic Updates

Problem:

Distribute Software Updates to customers

Solution:

use cheap standard media like USB memory sticks

Implementation:

- Keep BOM (Versions, Timestamps etc.) in persistent memory (EEPROM etc.)
- When booting, check whether a USB memory stick is plugged in.
- If one is found:
 1. If `prepare.img` is found load it into memory.
If it is valid then run it (always).
 2. If `preinst.img` is found load it into memory.
If it is valid then run it. Update the EEPROM.
 3. If `firmware.img` is found load it into memory.
If it is valid, burn it into FLASH and update the EEPROM.
 4. If `kernel.img` is found load it into memory.
If it is valid, burn it into FLASH and update the EEPROM.
 5. If `app.img` is found load it into memory.
If it is valid, burn it into FLASH and update the EEPROM.
 6. If `disk.img` is found load it into memory.
If it is valid, burn it into FLASH and update the EEPROM.
 7. If `postinst.img` is found load it into memory.
If it is valid then run it. Update the EEPROM.

1.13. Resources, Summary

Resources:

- The U-Boot project is hosted at Sourceforge: <http://sourceforge.net/projects/u-boot>
- Tarballs can be found either at sourceforge.net or at the [DENX ftp server](http://denx.de).
- There is a pretty active [u-boot-users mailing list](http://sourceforge.net/mail/forum/?do=manage&group_id=1000000).
- The Mailing list archive can be viewed at sourceforge.net (if it works – it is regularly broken, so don't get discouraged and try again later).
- The *DENX U-Boot and Linux Guide* is a Wiki based documentation documenting U-Boot and its interaction with Linux. It can be viewed (and improved) at www.denx.de. The whole DULG web packed into a [single HTML page](#) or a [PDF file](#) is also available (TQM8xxL version, i.e. PowerPC based).

- The current README file can be viewed through `viewcvs` at sourceforge.net
- Sample configuration files for the Abatron BDI2000 debugger are available at the [FTP Server at DENX](#).
- Even more useful links can be found in the DENX Training wiki documentation at www.denx.de.

Summary:

- U-Boot is a very active community project.
- It lives from the spirit of Free Software. It would be impossible without the numberless contributions of other developers.