Interfacing FreeBSD with U-Boot

Rafał Jaworowski
raj@semihalf.com

BSDCan 2008, Ottawa

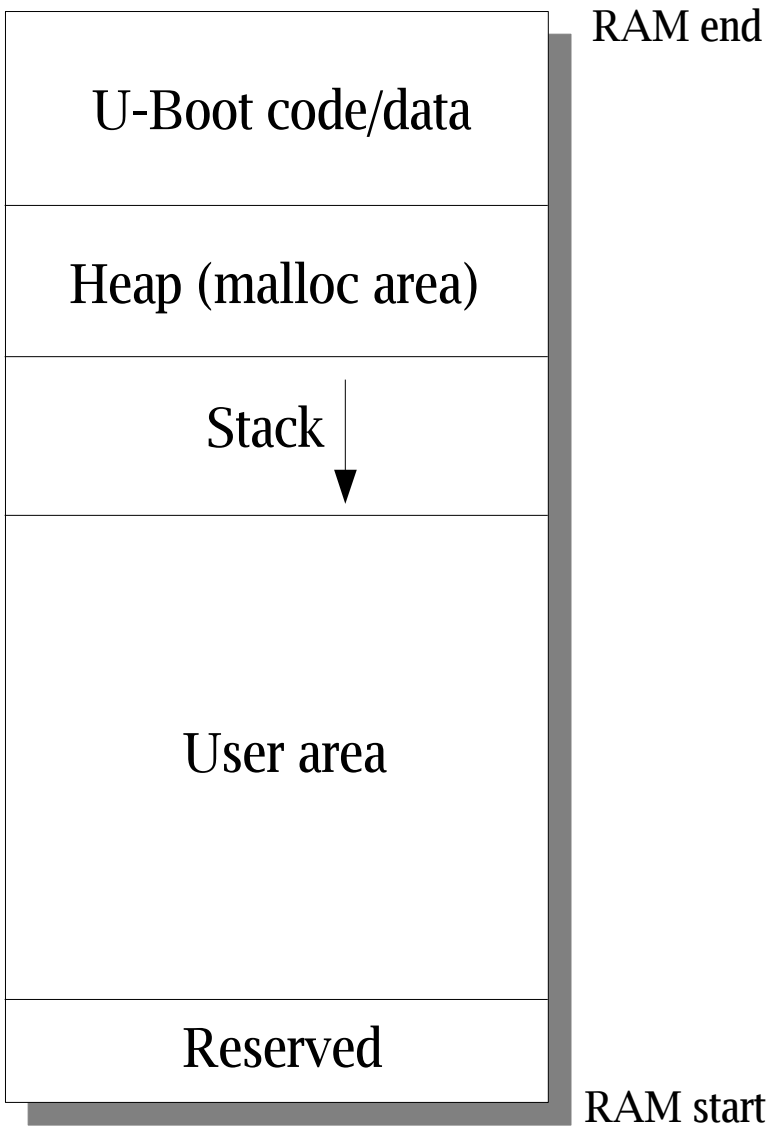
Introduction – TOC

- U-Boot basics
- Overview of FreeBSD booting process
- loader(8) + U-Boot
- New API for standalone applications
- Summary

Was ist *Das U-Boot*?

- Firmware
  - Stored in permanent memory (NOR/NAND FLASH)
- Initial level boot loader
- Configures H/W and starts operating system
- Overall operation
  - First executing from the permanent memory
  - Elementary CPU initialization, cache enable/disable, stack fixup etc.
  - Console
  - RAM initialization (controller, memory modules)
  - Relocation to RAM, continue executing from there
  - Further configuration of a system
  - Autoboot or interactive loop (user prompt)
  - No interrupts (polling)

After relocation to RAM



U-Boot highlights

- Written in C and assembly
- Accompanied by helper applications, host tools, documentation
- Self-contained source tree
- Closely tied to Linux
  - Follows its standards
  - Coding style
  - Interfaces (Flattened Device Tree)
  - License (GPLv2)
- Omnipresent in the PowerPC world, popular on other architecters too
- History: PPCBoot

Supported systems

- Architectures
  - ARM, AVR32, BlackFin, ColdFire, MicroBlaze, MIPS, NIOS, PowerPC, SH
- SoCs
- Platforms (boards)
  - `$ ls include/configs/ | wc`
    `423 423 4363`
- Peripherals
  - FLASH, DRAM chips, I2C, NICs
- Can boot various operating systems
  - FreeBSD, Linux, LynxOS, NetBSD, QNX, RTEMS, VxWorks, WinCE
  - Provisions for others
  - Historically oriented around Linux

U-Boot user features highlights

- Booting the OS
  - FLASH (NAND, NOR)
  - Network
  - Disk (S/ATA, SCSI, USB)
  - Handle RAMDISK image
  - Filesystems (EXT2, CRAMFS, FAT, JFFS2)
- Diagnostics (POST framework)
- Memory inspection, modification
- PCI/USB devices enumeration
- Self image update in permanent memory
- “scripts” - sequence of commands
- Environment variables (settings)
- Typically serial console, but supports selected graphics controllers, LCD, keyboards

Bootup example

```
U-Boot 1.3.2-dirty (Mar 10 2008 - 09:52:39)

CPU:   8555, Version: 1.1, (0x80790011)
Core:  E500, Version: 2.0, (0x80200020)
Clock Configuration:
       CPU: 825 MHz, CCB: 330 MHz,
       DDR: 165 MHz, LBC:  82 MHz
L1:    D-cache 32 kB enabled
       I-cache 32 kB enabled
Board: CDS Version 0x11, PCI Slot 1
CPU Board Revision 1.1 (0x0101)
    PCI1: 32 bit, 33 MHz, sync
    PCI2: 32 bit, 66 MHz, sync
I2C:   ready
DRAM:  Initializing
    SDRAM: 64 MB
    DDR: 256 MB
FLASH: 16 MB
L2 cache 256KB: enabled
PCI-X will only work at 66 MHz
In:    serial
Out:   serial
Err:   serial
Net:   TSEC0, TSEC1
=>
=> ping ${serverip}
Speed: 100, full duplex
Using TSEC0 device
host 10.0.0.204 is alive
=>
```

FreeBSD booting process overview

- Initial platform-level bootstrap (before OS)
  - BIOS, EFI
  - OpenFirmware, OpenBoot (IEEE 1275-1994)
  - RedBoot
  - U-Boot
- First stage FreeBSD boot
  - MBR, bootblocks (i386)
  - boot0iic, boot0spi (ARM)
- loader(8)
- Kernel
  - Receives boot info via metadata

loader(8)

- Last stage native FreeBSD bootloader
- Prepares metadata for the kernel
  - Boot parameters (single/multi user, root mount etc.)
  - Debugging info
  - Dynamic modules (KLD) chain
  - Environment variables (kenv(9)), device.hints
- FORTH (scripting)
- Uniform UI accross architectures and platforms
- Allows to load/unload kernel modules (including the kernel itself)

loader(8) structure

- Modular
  - Machine/platform dependent
  - Common, *archsw* dispatching
  - Various optional features (filesystem support, network protocols)
- Based on libstand(3)
  - Standalone applications library (“mini libc”)
  - printf(), malloc()/free(), setenv(), ...
  - Network protocols (BOOTP/DHCP, TFTP, NFS)
- Relies on underlying firmware for elementary operations
  - put/get console character
  - read/write network packet
  - read/write storage block

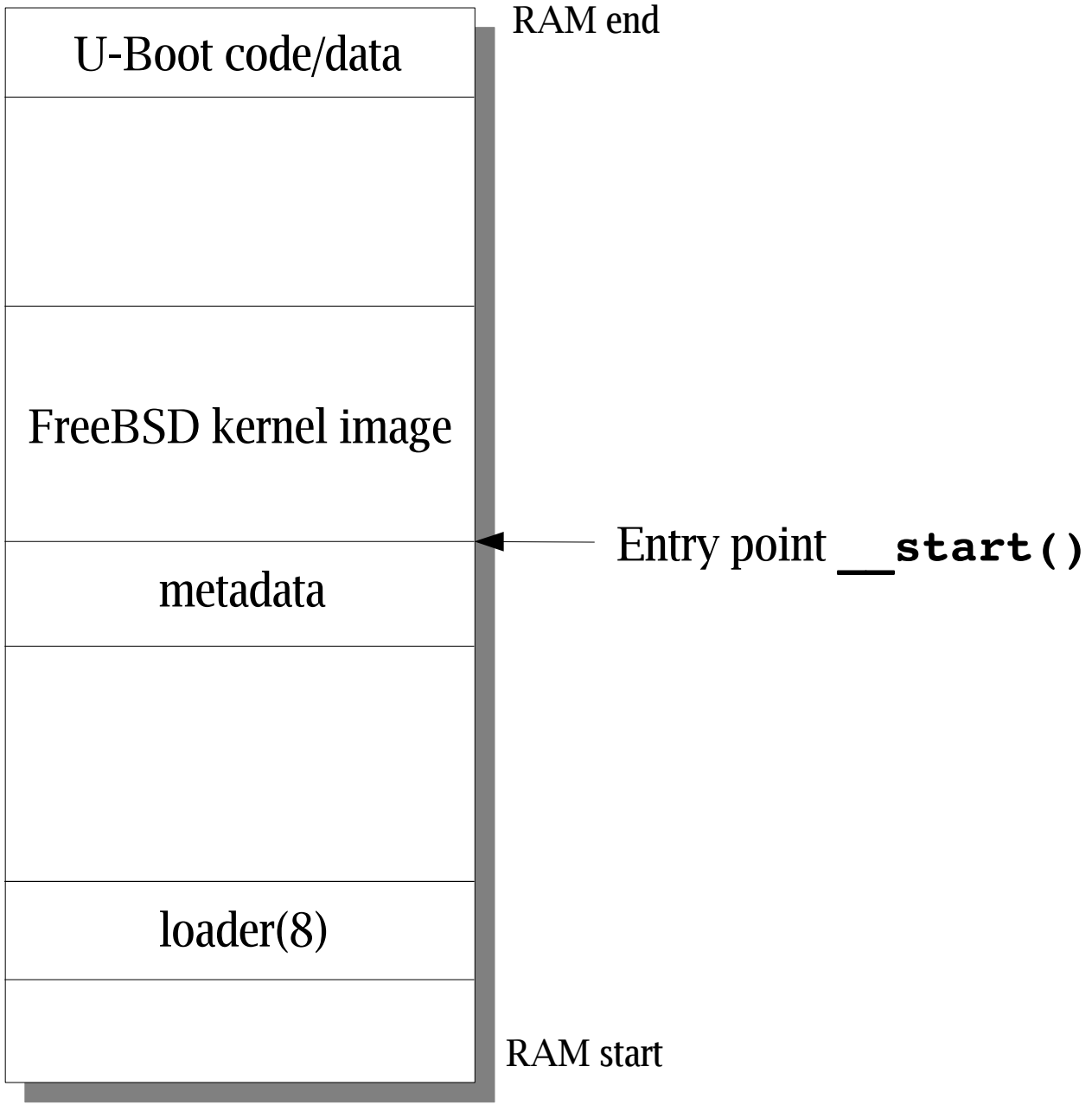
How U-Boot starts an OS

- Canonical way – *bootm* command
  - Operates on pre-packaged images – *mkimage* helper tool required
  - Different image types: kernel, ramdisk, script, filesystem, standalone application, multi-file
  - Images can be compressed
  - `do_bootm()`
    - processes the image header
    - loads contents to destination location
    - passes control (if applicable)
- Alternatives
  - *bootelf*
  - *go <address>*

Marrying FreeBSD with U-Boot

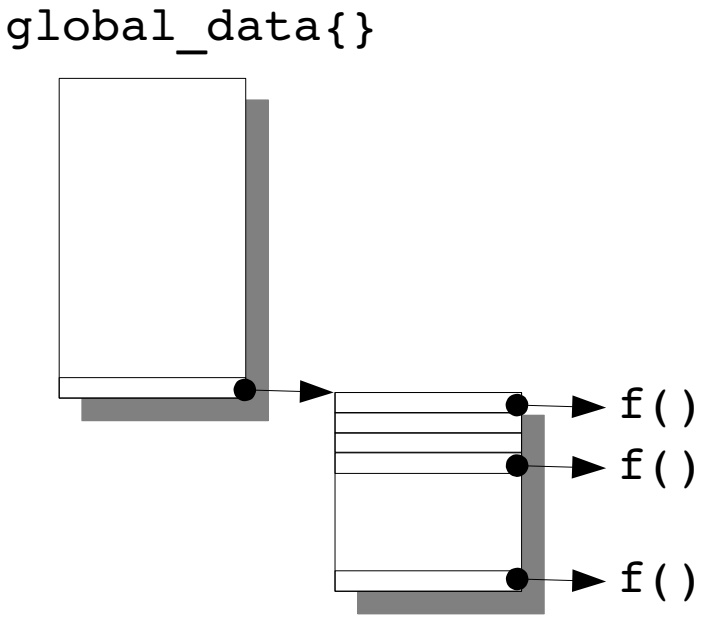
- Run FreeBSD kernel directly
  - ELF
  - Pure binary
- Native U-Boot approach
  - Teach *bootm* + *mkimage* system about FreeBSD
  - Cons
    - Functional changes, future extensions etc. require burning U-Boot image in permanent memory
    - *mkimage* helper tool as another dependency within OS build system
    - FreeBSD booting features not available (modules, metadata etc.)
- Native FreeBSD approach
  - Interface loader(8) with underlying firmware
  - loader(8) as U-Boot standalone application (ELF, pure binary)

U-Boot memory layout



Standalone applications in U-Boot

- Jump table
- Selected U-Boot functions are exported
  - Depend on platform configuration
- Accessed via global_data structure



PowerPC *global_data* dependencies

```
typedef struct  global_data {
        bd_t            *bd;
        unsigned long   flags;
        unsigned long   baudrate;
        unsigned long   cpu_clk;        /* CPU clock in Hz!             */
        unsigned long   bus_clk;
#if defined(CONFIG_CPM2)
        /* There are many clocks on the MPC8260 - see page 9-5 */
        unsigned long   vco_out;
        unsigned long   cpm_clk;
        unsigned long   scc_clk;
        unsigned long   brg_clk;
#endif
#if defined(CONFIG_MPC7448HPC2)
        unsigned long   mem_clk;
#endif
#if defined(CONFIG_MPC83XX)
        /* There are other clocks in the MPC83XX */
        u32 csb_clk;
#if defined(CONFIG_MPC834X) || defined(CONFIG_MPC831X) || defined(CONFIG_MPC837X
        u32 tsec1_clk;
        u32 tsec2_clk;
        u32 usbdr_clk;
#endif
#if defined (CONFIG_MPC834X)
        u32 usbmph_clk;
#endif /* CONFIG_MPC834X */
...
...
#ifdef CONFIG_BOARD_TYPES
        unsigned long   board_type;
#endif
#ifdef CONFIG_MODEM_SUPPORT
        unsigned long do_mdm_init;
        unsigned long be_quiet;
#endif
#if defined(CONFIG_LWMON) || defined(CONFIG_LWMON5)
        unsigned long kbd_status;
#endif
        void            **jt;           /* jump table */
} gd_t;
```

Jump table dependencies

```
void jumptable_init (void)
{
        int i;

        gd->jt = (void **) malloc (XF_MAX * sizeof (void *));
        for (i = 0; i < XF_MAX; i++)
                gd->jt[i] = (void *) dummy;

        gd->jt[XF_get_version] = (void *) get_version;
        gd->jt[XF_malloc] = (void *) malloc;
        gd->jt[XF_free] = (void *) free;
        gd->jt[XF_getenv] = (void *) getenv;
        gd->jt[XF_setenv] = (void *) setenv;
        gd->jt[XF_get_timer] = (void *) get_timer;
        gd->jt[XF_simple_strtoul] = (void *) simple_strtoul;
        gd->jt[XF_udelay] = (void *) udelay;
        gd->jt[XF_simple_strtol] = (void *) simple_strtol;
        gd->jt[XF_strcmp] = (void *) strcmp;
#if defined(CONFIG_I386) || defined(CONFIG_PPC)
        gd->jt[XF_install_hdlr] = (void *) irq_install_handler;
        gd->jt[XF_free_hdlr] = (void *) irq_free_handler;
#endif  /* I386 || PPC */
#if defined(CONFIG_CMD_I2C)
        gd->jt[XF_i2c_write] = (void *) i2c_write;
        gd->jt[XF_i2c_read] = (void *) i2c_read;
#endif
}
```

Introducing U-Boot API

- Single entry point (*syscall*) to API – mimics UNIX syscall
  - C-callable function in the U-Boot text
  - Might evolve into machine exception trap handler
- Consumer application is responsible for producing appropriate context (call number and arguments)
- Upon syscall entry, the dispatcher calls other (existing) U-Boot functional areas like networking or storage operations
- Consumer application will recognize the API is available by searching a certain (assumed by convention) range of address space for the API signature
- The U-Boot integral part of the API is meant to be thin and non-intrusive, leaving as much processing as possible on the consumer application side (doesn't keep states, but relies on hints from the application)
- Platform/architecture independent

API calls

- Console related
  - Get, put character
- Devices
  - Enumerate all, open, close, read, write, send, receive
  - Two classes of devices: network, storage
- U-Boot environment variables
  - Enumerate all, get, set
- System
  - Reset, platform info
- Time
  - Delay, current

API structure overview

- Core API, integral part of U-Boot, mandatory
  - Implements the single entry point
  - Prepares calltable
  - Places API signature in memory for applications to find
- Glue
  - Entry point at the consumer side, allows to make syscall, mandatory part
  - Helper conveniency wrappers for consumer application, so it does not have to use the syscall directly, but in a more friendly manner (a la libc calls), optional part
- Consumer application
  - Calls API directly, or via the glue mid-layer wrapper routines (ready-to-use)

api_public.h, glue.h

```
/*
 * ub_ library calls are part of the application, not U-Boot code!  They are
 * front-end wrappers that are used by the consumer application: they prepare
 * arguments for particular syscall and jump to the low level syscall()
 */

/* console */
int     ub_getc(void);
int     ub_tstc(void);
void    ub_putc(char c);
void    ub_puts(const char *s);

/* system */
void                    ub_reset(void);
struct sys_info *       ub_get_sys_info(void);

/* time */
void            ub_udelay(unsigned long);
unsigned long   ub_get_timer(unsigned long);

/* env vars */
char *          ub_env_get(const char *name);
void            ub_env_set(const char *name, char *value);
const char *    ub_env_enum(const char *last);

/* devices */
int                     ub_dev_enum(void);
int                     ub_dev_open(int handle);
int                     ub_dev_close(int handle);
int                     ub_dev_read(int handle, void *buf, lbasize_t len, lbastart_t start);
int                     ub_dev_send(int handle, void *buf, int len);
int                     ub_dev_recv(int handle, void *buf, int len);
struct device_info *    ub_dev_get(int);
```

```
enum {
        API_RSVD = 0,
        API_GETC,
        API_PUTC,
        API_TSTC,
        API_PUTS,
        API_RESET,
        API_GET_SYS_INFO,
        API_UDELAY,
        API_GET_TIMER,
        API_DEV_ENUM,
        API_DEV_OPEN,
        API_DEV_CLOSE,
        API_DEV_READ,
        API_DEV_WRITE,
        API_ENV_ENUM,
        API_ENV_GET,
        API_ENV_SET,
        API_MAXCALL
};
```

loader(8) + U-Boot API

- Introducing generic U-Boot support library
  - sys/boot/uboot
  - Equivalent to other firmwares support in the tree
- Architecture-specific connection
  - PowerPC implementation
  - sys/boot/powerpc/uboot
  - Manages context save/restore around the syscall
- Other architectures easily supported
  - Only the thin connection required
- API integrated with main line U-Boot since release 1.3.2
  - Optional (CONFIG_API)

In action!

```
...
=> tftp 100000 mpc8572ds/ubldr
Speed: 100, full duplex
Using eTSEC1 device
TFTP from server 10.0.0.204; our IP address is 10.0.2.7
Filename 'mpc8572ds/ubldr'.
Load address: 0x100000
Loading: #############
done
Bytes transferred = 172012 (29fec hex)
=> bootelf
Loading .text @ 0x00010080 (135456 bytes)
Loading .rodata @ 0x000311a0 (4080 bytes)
Loading .rodata.str1.4 @ 0x00032190 (13579 bytes)
Loading set_Xcommand_set @ 0x0003569c (72 bytes)
Loading .rodata.cst4 @ 0x000356e4 (40 bytes)
Loading .data @ 0x00036000 (5344 bytes)
Loading .sdata @ 0x000374e0 (92 bytes)
Clearing .sbss @ 0x00037540 (120 bytes)
Clearing .bss @ 0x000375b8 (6920 bytes)
## Starting application at 0x00010080 ...
Consoles: U-Boot console
Compatible API signature found @ff90010

FreeBSD/powerpc U-Boot loader, Revision 0.6
(raj@builder.semihalf.com, Mon May  5 12:35:40 UTC 2008)
Memory: 1024MB
FLASH:  128MB
/boot/kernel/kernel data=0x24c774+0x5ab78
Hit [Enter] to boot immediately, or any other key for command prompt.
Booting [/boot/kernel/kernel]...
Kernel entry at 0x1000100 ...
GDB: no debug ports present
KDB: debugger backends: ddb
KDB: current backend: ddb
L1 D-cache enabled
L1 I-cache enabled
Copyright (c) 1992-2008 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.
FreeBSD is a registered trademark of The FreeBSD Foundation.
...
```

Summary

- Non-traditional U-Boot approach
  - Not handled by the `do_bootm()` routine
  - *mkimage* not used
  - FreeBSD native loader(8)!
- Advantages of the new API
  - Simpler, more robust and flexible approach
  - Only changes to the API itself require burning U-Boot code
  - loader(8) not entangled with U-Boot internals
  - Independent of platform configuration options, architecture
  - A generic API mechanism, not really specific to FreeBSD...
  - *jump table* approach could eventually be dropped in favour of the API

Concluding remarks

- References
  - sys/boot/uboot
  - sys/boot/powerpc/uboot
  - U-Boot manual
    - http://www.denx.de/wiki/DULG/Manual
  - Main line U-Boot
    - http://git.denx.de/?p=u-boot.git;a=summary
    - api/README, api_examples
  - U-Boot-FreeBSD custodian repository
    - http://git.denx.de/?p=u-boot/u-boot-freebsd.git;a=summary
- Acknowledgements
  - Wolfgang Denk and U-Boot community
  - Marcel Moolenaar

Interfacing FreeBSD with U-Boot

Rafał Jaworowski
raj@semihalf.com

http://www.semihalf.com/pub/2008_uboot_freebsd.pdf

BSDCan 2008, Ottawa