How to put FreeBSD power into small MIPS switch/router, EuroBSDcon 2012

Aleksandr Rybalko

October 15, 2012

EuroBSDcon 2012

Complete picture (almost)

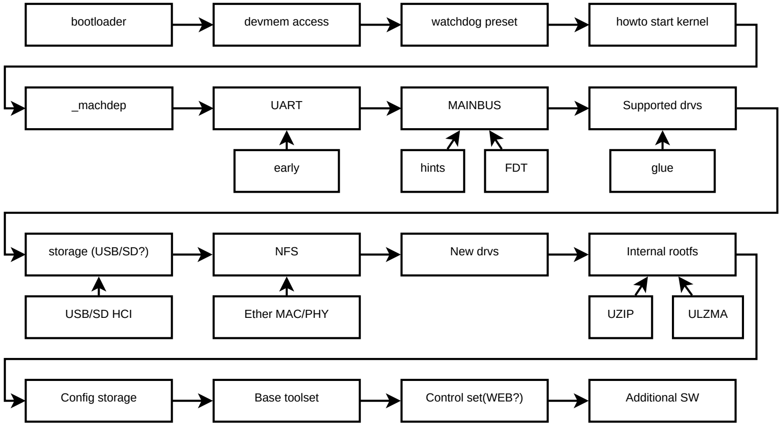

Figure: It's how I do that

Step 1 - Learn bootloader

Most common bootloaders:

REDBOOT	Red Hat Embedded Debug and Bootstrap firmware
Das U-Boot	Universal Bootloader
CFE	Common Firmware Environment, developed by Broadcom
...	

Step 2 - Find ways to access devices memory

What we need?

Memory write - can write, so able to test things.

Memory read - at least we need to know how THAT configured in bootloader.

Memory dump - always better to read many regs at a time.

2.1 No access to devices memory

What if we don't have any ways to access device memory? Try to:

- Make second loader (U-Boot)
- Make some loader application. (f.e. U-Boot have API for simple programs)
- Skip to «Ways to start kernel» :-)

Step 3 - Enable watchdog

What is **watchdog?**

It is a simple counter. We able to configure its countdown value. It is decrements on each CLK (CPU or main bus clock) and when 0 reached, watchdog reset SoC (or call CPU with interrupt).

So, enable watchdog and preset it to maximum value. Then you will not require to have direct access to device and/or remote controlled power switch.

Step 4 - Ways to start kernel

Find all possible ways to start kernel. It will helps for development.
in U-Boot:

go 0x00001000	- pass control to address 0x00001000.
bootm 0x00000000	- boot OS packed in U-Boot image loaded at 0x00000000.
ubldr	- use FreeBSD loader(8) “linked” with U-Boot. (sys/boot/uboot)

Step 4 - Ways to start kernel

Find standard way to start kernel. That will help to make users happy (we don't need to update bootloader).
Most U-Boot devices do bootm on address in Flash memory right after U-Boot + his Environment block:

0xbf000000 - 0xbf040000 - U-Boot

0xbf040000 - 0xbf050000 - U-Boot Environment

0xbf050000 - 0xbfxx0000 - OS kernel image packed with **mkimage** tool

Step 5 - Chip family startup code

We have code for platform start (ARM/MIPS/PPC/etc.), now we want our own point to start.
Copy one of already done _machdep.c files for that platform :-)
If you will do copy, developers can easily merge startup code later :-)

Step 6 - I can see! or just UART

If we don't have UART, we are blind.
We need just basic support of serial controller and that way much more easy than many other type of text communications.

But please remember, we still not have:

- DELAY(or it can't work)
- interrupts (controller not initialized yet or no driver for it)
- correct virtual bus address (we must define it later)

Step 7 - How can we find resources

acpi? nexus? fdtbus?

autodetect - good to have. PCI, USB and some other buses define places for device IDs, so we can match it.

hints - easy, but too simple. We need to define every property for device, as result big hints file.

FDT - complex, but powerful. Will be best choice, when H/W vendors begin to manufacture devices with FDT capable bootloader. (and FDT blob passed to kernel will honor standards :-))

Step 8 - Do we have something done?

- Search source tree to find, maybe we already have such driver.
- Maybe we can just add some modification.
- Maybe we need split driver into driver itself and bus glue.
- Maybe we can just add bus glue for existing driver.

Step 9 - Find easiest pad to launch rocket

While we try to start system here, we need some easy reachable way to read files. It can be:

- USB - require USB HCI (uhci/ohci/ehci/xhci/dwcotg - cover 95% devices)
- SD - require SDHCI (less frequent, but may be there is same for most SoCs of that vendor which already supported)

Step 10 - Make access to updated data easy

Network file systems help us to build programs on one machine, but run on our newly supported box.
Best candidate is NFS, but network drivers required.
So we need to done:

- MAC support (NIC driver)
- PHY support (easy)
- embedded/onboard switch support (ask #bsdmips or #zrouter, hope someone will help)

Step 11 - Time for new drivers

New drivers implementation
Hints: Make drivers system/arch independent, SoC vendors are very lazy and device cores may migrate to SoCs with different arch or even different vendors.

Step 12 - Select Root FS type

Imagine, you have drive with 128 sectors 64k each, can you put something usual here?
Yes, but with tricks :)

- cd9660 with gz, GEOM_UZIP.
- cd9660 with lzma, GEOM_ULZMA.
- nandfs like (for NAND flashes).
- something like nandfs but for NOR flash ???
- Filesystem compression ???
- ...

How can we pack that all into image?

- mdconfig+dd+tar - mount memory disk (require root privileges).
- makefs - good, but files stay owned by user who build image.
- makefs+mtree - fine, but we need install to temporary tree, scan, fix perm, then build fs.
- makefs(tuned to NetBSD one)+install - best, we can install into image file direct.

Footshooting or updating rootfs on working system

- two rootfs - require more flash
- mfs rootfs - require more ram
- geom alias (/dev/map/upgrade) + geom intersection

Open question: How to partially preload shared library ???
Otherwise we waste memory, and then we can't load full update image to RAM.

Step 13 - Where to store configs?

Config storage:

- etc.tar.gz - mount tmpfs over RO /etc with unionfs and just pack it with tar into 1-2 flash blocks partition.
- FS mounted in Read-Write mode - not always possible.
- network-autoconfig - device can get everything configured from network (DHCP, PPP, SNMP, etc.), but anyway we should get some identification info somewhere.

Step 14 - Are we enough slim?

Select base toolset:

- standard base - everybody like it, but whole set even with LZMA/xz can't fit to box with 8M (or 4M) of flash.
- reduced base (only few required tools) - acceptable and works now.
- bsdbox - crunchgen packed toolset, like "/rescue/*". Works. Thank you Adrian!
- /bin/sh "applets" - ??? We can add some more builtins which frequently used in shell scripts. Then we have smaller replacment of original tools and they will be faster, since no exec required.

Reduce userland size

- limit library symbols (objcopy excluding unused).
- cut off unused(for embedded) code in tools (f.e. hide all JAIL symbols).
- alternative userland (musl libc, busybox, etc).

Step 15 - How we will control it?

Control set

1. shell+editors - just shell access and simple editor.
2. WebUI - complex subproject.
3. SNMP - too :-)
4. Big control framework with frontends 2 and 3 (maybe 1).

Step 16 - We need more!

If we need additional software, we can:
Import that software into project (License problems, hard to maintenance, etc).
Ports cross-building (Not done, but many good developers working on that).

Links

- http://www.FreeBSD.org
- http://wiki.FreeBSD.org
- http://lists.FreeBSD.org
- http://www.ZRouter.org
- http://www.ZRouter.org/hg/
- http://lists.ZRouter.org

Contacts

Thanks.

Aleksandr Rybalko

<ray@ddteam.net>

@FreeBSD.org

@ZRouter.org

@dlink.ua

TOC