# Porting OpenBSD to MIPS based devices

Rainer Giedat

<rainer@staatssicherheit.com>

1./2. Dec 2007, OpenCON Venice

# Motivation

- Do something new/challenging
- Learn how stuff works (OpenBSD/Hardware)
- Do something usefull
- Have fun!!

# The hardware

Linksys WRT54G

- Broadcom BCM47xx SoC
  - MIPS32 4Kc CPU (BCM3302)
  - 200 MHz
  - RAM: 32 MB (14MB)
  - 2 FastEthernet interfaces
  - BCM43xx 802.11 interface
  - ADMtek ADM6996L switch

# Software

- OpenWRT (Linux, GPL)
- CFE (Broadcoms Common Firmware Environment, BSD)
- Maybe OpenBSD soon ;)

# More MIPS32 hardware

- Alchemy systems
- Routerboard.com
- IBM z50
- Atheros based WLAN routers
- All kinds of embedded systems...

# Start

Small demo programm with serial console (barebone)

- Learn how to compile cross
- Learn little MIPS assembly
- learn how to load and boot binaries
- Find the serial console

# Cross Compiler

- Nice OpenBSD cross compile framework
- Hard to configure gcc
- Crazy bugs (ld)

# Toolchain target `mipsel` for OpenBSD

- Building BFD library to support mipsel on OpenBSD
- Configuring ld, gas and gcc for the new target Examples:
  - gcc (gcc/gcc/config.gcc)
  - ld (binutils/ld/configure.tgt)

# Use the cross compile framework

Makefile.cross

- Board and CPU

```
export TARGET_ARCH=mipsel
export TARGET=bcm47xx
make -f Makefile.cross cross-tools
```

# Learning MIPS assembly

- Documentation from MIPS Inc.
- A lot of tutorials from universities
  - All using the SPIM simulator
- Porting Linux to MIPS howto `http://linux.junsun.net/porting-howto/`
- Use gcc -S and/or objdump

# Play with it

- Get `http://linux.junsun.net/porting-howto/src/barebone.tar.gz`
- Read some easy code! Understand it!
  - start.S
  - barebone.lds
  - Makefile

# Try to compile and run it

- Does it crash/reboot?
- Right load adress?
- You do not see anything?
- Why not?

# Serial console

- See firmware info

```
CFE> show devices
uart0    NS16550 UART at 0x18000300
```

- See linux dmesg

```
ttyS00 at 0xb8000300 (irq = 3) is a 16550A
```

# MIPS32 memory layout (I)

| VA | Name | Address Range | Mode | Size |
|---|---|---|---|---|
| 0b111 | kseg3 | 0xFFFFFFFF→0xE0000000 | Kernel | $2^{29}$ bytes |
| 0b110 | ksseg | 0xDFFFFFFF→0xC0000000 | Super | $2^{29}$ bytes |
| 0b101 | kseg1 | 0xBFFFFFFF→0xA0000000 | Kernel | $2^{29}$ bytes |
| 0b100 | kseg0 | 0x9FFFFFFF→0x80000000 | Kernel | $2^{29}$ bytes |
| 0b0xx | useg | 0x7FFFFFFF→0x00000000 | User | $2^{31}$ bytes |

# MIPS32 memory layout (II)

- Modes: User, Kernel, Supervisor
- Determined by the StatusRegister
- Kernel Mode: kseg1 mapping (0xBFFFFFFF-0xA0000000) → (0x00000000-0x1FFFFFFF)

# Serial Console (II)

```
.data
        x:      .byte   0x41
.text
        lb      v0, x
        sb      v0, 0xb8000300
```

# Compile a Kernel (I)

- Just copy the shit from arch/mips64 to arch/mipsel
- keep conf/files.mipsel in sync
- do the same for arch/sgi to (i.e.) arch/bcm47xx
- Delete everything from GENERIC what you do not need
- keep conf/files.bcm47xx in sync

# Compile a Kernel (II)

- Compile with

  `MACHINE=bcm47xx MACHINE_ARCH=mipsel make`
- Will not work → Port assembly to MIPS32
  - Mainly cuting constant values
  - Changing load/store instructions to 32bit (not CP0)
- Don't forget to set the correct LINK_ADDRESS

# Load the new Kernel

- Most Firmwares can load ELF binaries by TFTP (CFE does)
- Print chars every step to see how far it goes
- Fix ABI in locore.S ;-)
- call `mips_init`

# mips_init in machdep.c

- Initialize Console
- Configure MMU
- Configure Cache
- Configure physical memory
- Initialize interrupt handlers

# CFE interface (I)

```
A0: Firmware handle
A1: NULL
A2: Firmware entry point
A3: Seal (0x43464531)
```

→ everyting we need in `mips_init`

# CFE interface (II)

- Read/Write system console (wrapper)
- Manage caches (Invalidate/Flush D/I Caches)
- Get physical memory blocks
- Polling interface to network devices

# Get physical memory

`cfe_enummem`

- New block of memory on each iteration
- Store it in `phys_memseg` array
- Tell UVM about them with `uvm_page_physload`

# Initialize the MMU

- Set pagesize(normally 4k)

  ```
  uvmexp.pagesize = PAGE_SIZE;
  uvm_setpagesize();
  ```

- Load physical memory to UVM
- Set the size of the TLB (guess it, or better read it from the config register )
- Flush the TLB
- Set TLB PID (ASID) to 1 for proc0

# Initialize interrupt handler

- Only 0x80 bytes space for them
- Just copy them to the correct addresses
    - TLB MISS EXC
    - CACHE ERR EXC
    - Generic EXC

# System BUS

SBBUS Silicon Backplane BUS

- Different cores
- Main Core0 always present at 0x18000000
- 1k register space for each core
- All have the `coreid` at the same place

# BUS Probing

- Map first 1k at 0x18000000
- Get the numer of cores from CoreCommon config
- Map the next 1k for each of them
- Read their `coreid`

# Cores on WRT54G

- Core 1 id: 0x806 ethernet core
- Core 2 id: 0x816 mips3302 core
- Core 3 id: 0x817 usb 1.1 host core
- Core 4 id: 0x80f memc sdram core
- Core 5 id: 0x812 802.11 core
- Core 6 id: 0x81c roboswitch core

# TODO

- Write DMA Code
- Enable interrupts on it
- Write driver for Ethernet
- Write driver for 802.11

# Compiling Userspace

- Repeat stuff you did for the kernel
    - Copy machine dependend stuff from mips64 to mipsel
    - Change it until it compiles ;)
- `make -f Makefile.cross cross-distrib`
- Build a ramdisk

# Config for Ramdisk

- Kernel config GENERIC

```
option   MINIROOTSIZE=3000
option   RAMDISK_HOOKS
config   bsd     root on rd0a swap on rd0b
pseudo-device   rd      1
```

# Preparing for Ramdisk

- Steal a SRCDIR/ramdisk/bcm47xx somewhere
- Delete entries from `list` you do not need
- Build it =)

# Building the Ramdisk

- Use `make -f Makefile.cross cross-env`
- Then just `make`
- Do not forget

```
make unconfig
```

  - Unmounting the pseudo filesystem `svnd0`

# Funky Bug

- Kernel crashed at random points
- Always the same point, same problem
- Changed sometimes after recompilation
- Seemed to be a bug in memory management

# Hunting the Bug

- Checked the code in pmap.c
- Got ddb running
- Learned how TLB works
- Wrote code to print page tables
- Read UVM code and documentation
- Discussed it with people (Mickey, ...)

# Finding the Bug

- It was elfrdsetroot
- It prints size of the ramdisk and size of the image on different bases (hex and decimal)
- It did not complain, that there is not enough space
- Searched the bug for weeks, because I can not read and elfrdsetroot can not write ;)

# What works so far?

- Kernel is booting
- Console and ddb works
- Interrupts work
- Syscall work
- Init is starting...

# What does not work?

...then it crashes!

- mmap, mprotect, atexit,
- __do_init, __init, __start
- Then it reads the old StackPointer
- The stack contains crap
- Calling __perf_init crashes, because of not aligned StackPointer (0x7ffd7ea1)
- In the last syscall, the stack was OK

# Plans to find the Bug

Port to Qemu/MIPS

- Emulates MIPS 4k CPU
- Interrupt controller i8259 PIC
- Timer i8254 PIT
- Then use the Qemu debugger
- Should not be too hard...
- Who wants to help? ;)

# Thanks

- Discussions, Answering Questions:
  - Mickey, Uwe Stühler, Martin Reindl, Alexander Bluhm
- Hardware Donations:
  - Hans Höxer, Wim Vandeputte, Klaus Landefeld
- Kicking my ass over and over again:
  - Too many People