Porting OpenBSD on the MIPS64-based Octeon Platforms

Paul Irofti
pirofti@openbsd.org

BSDCan, Ottawa
May 2014

Outline

Who Am I?

Reverse Engineer (6 years in the AV industry)

- anti-virus engines
- emulators: static and dynamic analysis research

OpenBSD Hacker:

- power management, ACPI
- mips64: Loongson and Octeon
- `compat_linux(8)` maintainer
- porter

Research Assistant and PhD student:

- Faculty of Automatic Control and Computers at the Polytechnic University of Bucharest
- PhD on parallel signal processing algorithms using GPGPU (OpenCL, CUDA)

My connection to Octeon

- played with other mips64 ports in the past
- mostly worked on the Loongson architecture
- first contact while sitting next to jasper@ at t2k13
- mentioned it in the undeadly report after the hackathon
- article led to a kind donation from Diana Eichert

Dealing with U-Boot

After a bit of reading, the magic tftpboot'ing uboot commands are:

```
D-Link DSR-500 bootloader# dhcp
D-Link DSR-500 bootloader# tftpboot 0 bsd
D-Link DSR-500 bootloader# bootoctlinux ./bsd
```

Copyright Crash

Kernel crashed after copyright:

```
Copyright (c) 1982, 1986, 1989, 1991, 1993
  The Regents of the University of California.
  All rights reserved.
Copyright (c) 1995-2013 OpenBSD. All rights reserved.
  http://www.OpenBSD.org
```

Octeon Memory

In-depth investigations pointed to the memory setup routines.
This is how Octeon memory is organized:

PA Chunks	From	To
1st 256 MB DR0	0000 0000 0000 0000	0000 0000 0FFF FFFF
2nd 256 MB DR1	0000 0004 1000 0000	0000 0004 1FFF FFFF
Over 512MB DR2	0000 0000 2000 0000	0000 0003 FFFF FFFF

First Encounter: Too small

The DSR-500 has 128MB of memory.

- the smallest system memory so far
- `octeon_memory_init()` assumed at least 256MB
- BUG: start of the 2nd bank after 256MB
```
phys_avail[1] = OCTEON_DRAM_FIRST_256_END;
realmem_bytes -= OCTEON_DRAM_FIRST_256_END;
```
- FIX: cap to `realmem` if less than 256MB
```
phys_avail[1] = realmem_byte
```

Userland

After the fix I got a big reward:

```
OpenBSD 5.4-current (GENERIC) #32:
  Fri Aug 30 14:19:07 EEST 2013
[...]
scsibus0 at vscsi0: 256 targets
softraid0 at root
scsibus1 at softraid0: 256 targets
root device:
```

Dmesg

Inspecting the **very** short dmesg there were some obvious problems:

- `octcf at iobus0 not configured`
- `0:0:0:  mem address conflict 0xf8000000/0x8000000`
- `ukphy0 at cnmac0 phy 0`
- `/dev/ksyms:  Symbol table not valid.`

Where to?

Going forward my plans were to:

- enrich the platform by adding new drivers
- add storage support through internal cf and umass
- enable networking
- help jasper@ to improve the 2nd-stage bootloader
- make the port able to stand on its own

Dissapointment

The SDK license made me sad:

```
This Software, including technical data, may be subject
to U.S. export control laws, including the U.S. Export
Administration Act and its  associated regulations, and
may be subject to export or import  regulations in other
countries.
```

Warm-up driver

I decided to write a simple driver to get to know the platform.

Why the RNG?

I chose the random-numbers generator because it seemed:

- easy to initialize
- simple output
- clean integration with the OpenBSD random subsystem

RNG Setup

Initialization is done through a control register:

- read the control register
- set the output flag
- set the entropy flag
- write-back the control register

The above should start producing randomness in a few seconds.

Fetching

Random numbers are written in the entropy register.

- read 8-bytes from the register address
- feed it to `add_true_randomness()`
- schedule another read after 10ms

That's it!

Lessons Learned

Obstacles:

- endianess confusion when dealing with register addresses
- required read after write when setting the control register
- get 8-bytes, feed only 4 to the random subsystem

Itching

NFS-booting was annoying me everytime with this:

```
WARNING: file system time much less than clock time
WARNING: CHECK AND RESET THE DATE!
```

It had to stop!

Available clocks

Some boards have an RTC that can be used as a TOD clock:

- DS1337 clock model
- resolution of 1 second
- provide gettime and settime routines
- register them to be used as the system TOD clock

Two-Wire Serial Interface

Time is read through the TWS registers:

```
uint64_t v:1; /* Valid bit */
uint64_t slonly:1; /* Slave Only Mode */
uint64_t eia:1; /* Extended Internal Address */
uint64_t op:4; /* Opcode field */
uint64_t r:1; /* Read bit or result */
uint64_t sovr:1; /* Size Override */
uint64_t size:3; /* Size in bytes */
uint64_t scr:2; /* Scratch, unused */
uint64_t a:10; /* Address field */
uint64_t ia:5; /* Internal Address */
uint64_t eop_ia:3; /* Extra opcode */
uint64_t d:32; /* Data Field */
```

How TWS Access Works

Operating the TOD clock:

- set the address field to the RTC register
- afterwards use current internal address across calls
- set the operation type by setting/clearing the read flag
- read from or write to the data field

Reads(1)

1st step:

- set RTC register address
- set the read bit
- set the valid bit
- set op to use the current address if a read was done before
- write the TWS register

Reads(2)

2nd step:

- read-back the TWS register
- keep reading until the valid bit is cleared
- if cleared, the operation was completed
- fetch clock data from the 1st byte in the data field

Writes(1)

1st step:

- set RTC register address
- clear the read bit
- set the valid bit
- fill the data field
- write the TWS register

Writes(2)

2nd step:

- read-back the TWS register
- keep reading until the valid bit is cleared
- do an extra read-back after the operation was completed

RTC Format

The clock information is BCD-coded as follows:

```
tt->year = ((tod[5] & 0x80) ? 100 : 0) + FROMBCD(tod[6]);
tt->mon = FROMBCD(tod[5] & 0x1f);
tt->day = FROMBCD(tod[4] & 0x3f);
tt->dow = (tod[3] & 0x7);
tt->hour = FROMBCD(tod[2] & 0x3f);
tt->min = FROMBCD(tod[1] & 0x7f);
tt->sec = FROMBCD(tod[0] & 0x7f);
```

TOD Routines

The `settime` and `gettime` routines:

- pack/unpack the time data into/from BCD form
- read/write each packet through the TWS registers

Issues

The TWSI is very fragile and needs a lot of integrity checks.

Besides, some models have an RTC clock:

- D-Link DSR-500
- Portwell CAM-0100.

Others don't:

- Ubiquiti Networks EdgeRouter Lite.

Njetwork

Even though I used tftpboot and NFS-root on boot...

```
# ifconfig cnmac0
cnmac0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST>
        lladdr 00:de:ad:20:75:00
        priority: 0
        media: Ethernet autoselect (none)
        status: no carrier
        inet 192.168.1.9 netmask 0xffffff00 broadcast
# ping k.ro
ping: unknown host: k.ro
```

BCM53XX

Missing Broadcom PHY driver for the chip 53XX-family

- resulted in `cnmac0 at ukphy`
- looked at OpenWrt for the proper registers
- wrote a minimal PHY driver with dumb-mode only switch support

Status Routine

The key was the status PHY routine which reads the:

- link state
- duplex mode
- port's speed

via corresponding registers from the status page.

Reads(1)

1st step:

- set the page number if the current one differs
- set the register address
- read-back to check for operation completion

Reads(2)

Once the page-register tuple is in place:

- read 2 bytes from the first data register
- if needed go on with the 2nd, 3rd and 4th data registers

Results

With that in place the network seems better:

```
# ifconfig cnmac0
cnmac0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST>
        lladdr 00:de:ad:20:75:00
        priority: 0
        groups: egress
        media: Ethernet autoselect (1000baseT master)
        status: active
        inet 192.168.1.9 netmask 0xffffff00
# ping k.ro
PING k.ro (194.102.255.23): 56 data bytes
64 bytes from 194.102.255.23: icmp_seq=0 time=60.132 ms
64 bytes from 194.102.255.23: icmp_seq=1 time=63.555 ms
```

Switch Support

In the future I plan to add switch support.

Existing kernel switch frameworks:

- ZRouter solution from Aleksandr Rybalko
- IIJ solution from Kazuya Goda

Waiting for the IIJ framework to be made public before deciding.

Existing Work

- Cavium SDK
- IIJ driver for the CN30XX boards
- WIP driver I wrote that fries USB sticks

My Progress So Far

- clock setup
- host-mode setup
- core setup
- dma setup
- part of the interrupt routine

Interrupt Routine

Almost done:

- host channel interrupts
- issues: assumes single USB port
- device disconnect interrupts
- issues: doesn't callback

Show-Stoppers

- can't use the SDK: **U.S. export control**
- SDK and IIJ register poking logic is completely different
- can't reuse IIJ's code as it doesn't work on my D-Link
- USB is hard
- USB w/o documentation is harder
- writing a driver requires time, which is the worst

Flash Memory

OpenBSD doesn't support the DSR-500 flash memory:

```
octcf at iobus0 base 0x1d000800 irq 0 not configured
```

FreeBSD does through CFI:

```
cfi0: <AMD/Fujitsu - 32MB> on ciu0
cfi0:
cfi0: [256x128KB]
cfid0 on cfi0
```

Plans

Discussed with David Gwynne (dlg@):

- write a small ATA driver
- plug it into the rest of the system via atascsi
- no multiple concurrent commands
- no port multipliers

Implementation

Writing a driver was slower than I expected:

- it's my first disk driver
- atascsi has too many abstractions
- the reference drivers `ahci(4)` and `sili(4)` are complex
- cf doesn't even do dma, it does bus space reads/writes
- `wdc(4)/pciide(4)` is messy

Work In Progress

I just started working on the driver. Unsure about:

- mimicking the FreeBSD abstraction: *cfid* → *cfi* → *atascsi*
- keeping the driver MD or making it MI
- supporting the entire CFI specification
- including the Intel mess
- StrataFlash?

Mostly Harmless

Conclusions

- a lot of work was put into `OpenBSD/Octeon`
- lack of documentation is slowing progress
- SDK copyright capped the pace even further
- open problems: USB, switch framework, CFI
- work continues to make this port complete

So Long, and Thanks for All the Fish

Questions?