Embedding FreeBSD/powerpc

Notes on the journey to the embedded world

Rafał Jaworowski
raj@semihalf.com

BSDCan 2007, Ottawa
Introduction – overview

- FreeBSD/powerpc on the embedded PowerPC platform
  - Freescale Semiconductor's MPC85xx PowerQUICC III integrated communications processor
- Focus on the bring-up level development
  - The most critical areas required for the system to run in single and multi-user
    - Booting, low-level init
    - Virtual memory, MMU
- Other functional areas also mentioned but not thoroughly discussed
• Why PowerPC (POWER)
  - Widespread, deployed in countless embedded and desktop devices
  - Architecture actively maintained and supported (cross industry POWER specifications – power.org)
  - AMCC, Freescale, IBM, P.A. Semi
  - Clean design, RISC architecture
FreeBSD/powerpc before

- Limited to Apple's PowerMac platform (*newworld* machines), now going obsolete
- Still Tier 2
- Most important architectural limitations:
  - No SMP
  - No 64-bit support
  - Only works with underlying OpenFirmware
  - Only supports the traditional PowerPC (Apple/IBM/Motorola) definition
Hardware platform

- Freescale Configurable Development System
  - Arcadia motherboard
  - Modular, SoC on the daughter card, different variants can be used in one frame
- MPC85xx communications processor (PowerQUICC III)
  - Core CPU
  - SDRAM controller
  - I²C
  - DUART
  - Interrupt controller
  - Ethernet (TSEC)
  - PCI
  - DMA engine
  - Security engine
  - CPM (dedicated RISC comm processor)
e500 core

- Low-power embedded RISC processor
- Implements the Book E definition of the PowerPC architecture
- 32-bit (using lower words in the 64-bit general-purpose registers)
- Compatible with the user-instruction set architecture (UISA) of the traditional PowerPC definition (AIM)
- Auxiliary processing units (APUs) - signal processing, vector instructions
- 32-bit effective addresses and integer data types of 8, 16, and 32 bits
- Superscalar, can issue and complete two instructions per clock cycle
- L1 caches for instructions and data (32-Kbyte for each)
- Memory management units
Critical e500 features

- MMU different from the traditional PowerPC model (AIM)
  - no BATs and segments: TLB-only
  - no real-mode i.e. valid TLB entries always needed
- L1 MMUs
  - maintained entirely by the hardware (LRU algorithm)
- L2 MMUs
  - 16-entry, fully-associative unified (instruction and data) TLB, variable-sized pages
  - unified TLB for 4-Kbyte page size
    - 256-entry, 2-way set-associative (e500v1)
    - 512-entry, 4-way set-associative (e500v2)
  - maintained by the software
Critical e500 features cont'd

- Exceptions/interrupts model
- Registers
  - UISA-level compatible with traditional AIM
  - e500-specific instructions and registers (rfmci, rfci, MCSRRs, CSRRs etc.)
- No traditional PowerPC FPU (floating point instructions trigger an FP exception)
  - use SoftFloat emulation library with libc
  - handle kernel traps
Environment

- FreeBSD 6.1-p10 source tree
- Toolchain
  - Cross-building
  - gcc 3.4.6 from base
  - e500-specific asm switches
- Layout of arch/powerpc
  - aim/booke split
  - platform subdirectory
  - re-design directory layout to accommodate this and other core variants in the future
Booting

- **Underlying firmware: U-Boot**
  - extensions to the firmware: networking, new API

- **loader(8) - 2\textsuperscript{nd} stage bootloader**
  - running as an application on top of U-Boot
  - U-Boot support library can be re-used on other FreeBSD ports like ARM, MIPS
  - effective addressing issues at the loader-kernel boundary (ELF processing, producing metadata)

- **Locore kernel**
  - does not assume any hardware setup on the firmware other than very basic initialization of the memory controller and clocking
  - allows to boot the kernel from other bootloaders as it's not tied to any particular one (provided they construct and pass the metadata)
  - de-OF-ing: current FreeBSD/powerpc port is strongly entangled with OpenFirmware
Firmware/bootloader memory view

- **U-Boot code/data**
- **FreeBSD kernel image**
- **metadata**
- **loader(8)**

**RAM start**

- **RAM end**
  - Linked against 0xC000_0000
  - Loaded at 0x0100_0000

**Entry point**

- **__start()**
  - Discover its whereabouts
  - Re-init MMU settings
  - Firmware/bootloader no longer available

- **booke_init()**
- **mi_startup()**
Low-level VM design and implementation

- Basic characteristics
  - 32-bit effective address
    - 32-bit real address (e500v1), 4G space
    - 36-bit real address (e500v2), 64G space
  - Big- and true little-endian per page
  - No real-mode, translations always required
  - At reset core begins execution at fixed virtual address, MMU has a default translation

- New *pmap* module developed
  - FreeBSD pmap(9) interface – inherited from Mach VM design
  - Two-level forward page table approach
  - Not the traditional PowerPC inverted page table
Forward page table

Page table directory

Page tables

PTE
PTE

Real pages
L2 unified MMU resources

**TLB0**
- 4K pages
- Set-associative
  - 256/512 entries
- Used for dynamic translations

**TLB1**
- Variable sized pages
- Fully associative
  - 16 entries
- Used for permanent translations of kernel critical mappings

---

Embedding FreeBSD/powerpc

BSDCan 2007
Translation primitives summary

Effective addresses

```
“Long term” translations – page table
- State maintained by \textit{pmap} + higher VM
- Covers the whole address space of a process
```

TLB

```
Dynamic translations – TLB
- Lowest level mem management
- Updated by the TLB-miss handler
- Unrelated entries
- Translations cache
```

Real addresses
TLB in action

Virtual address (41-bit)

| AS (1-bit) | PID (8-bit) | Effective address (32-bit) |

L1 MMUs \(\leftrightarrow\) L2 MMU

- Operations on the L2 MMU via assist registers (MAS)
- Shared pages PID = 0

Real address (32/36-bit)
Simplified virtual memory view

Kernel

0xFFFFFFFF

Kernel text, msgbuf, kstack0 etc.

Kernel page table structures

Permanent translations covered by TLB1

Process

0xC000_0000

0x0000_0000

CCSR

KVA
More challenging low-level VM aspects

- Nested TLB-misses
  - Where they come from
  - How to manage
    - Very carefully – a mishandled nested TLB-miss is endless [hang]
    - TLB-miss handler is very tiny, hand-crafted, with only a small and well defined cases that can generate a nested miss, can accommodate recursion

- TLB-miss handling overview
  - Search for the faulting address in the page table (pdir -> ptable)
  - If VA -> PA translation found, put in the TLB, job done
  - If not found, create a fake translation: this will promptly cause a corresponding ISI/DSI exception on faulting address (and let higher VM manage this)
Nested TLB-miss

KVA

Process

1. D/I miss

2. D miss

3. D miss
Exceptions, interrupts

• Vectors are not required to be at prefixed location (contrary to the AIM design)
  – Critical Input
  – Machine Check
  – Data Storage (DSI)
  – Instruction Storage (ISI)
  – External input
    • OpenPIC interrupt controller
  – Alignment
  – Program
  – System Call
  – Decrementer
  – Fixed Interval Timer
  – Data TLB Error
  – Instruction TLB Error
  – Debug
Peripheral devices

- On-chip peripherals
- No auto-enumerating
  - Hints mechanism used in early development
- All drivers according to Newbus paradigms
  - OCP bus, local bus, PCI/host bridge, TSEC (Ethernet), QUICC etc.
- bus_space, bus_dma frameworks
  - Buses endianness
  - bus_space macros -> function ptr ops
Newbus hierarchy

root

nexus

RMAN I/O MEM

ocp

lbc
tsec

i2c

uart

pci

... opic

RMAN IRQ
Current state of MPC85xx support

- Single/multiuser, networking
  - Booting with rootfs mounted from ATA disk, USB, NFS
- Key integrated peripherals supported
  - host/PCI bridge
  - TSEC (integrated Ethernet)
  - UARTs
  - QUICC engine (SCC etc.)
- Debugging: kdb/ddb, gdb
- Runs happily on MPC8555, 8541, 8548
Work in progress

- Merging to the FreeBSD source tree
- Started work on Perforce integration, then main CVS
- Targetting 7.0-RELEASE
- CPU variation within the same architecture
  - Unlike any other FreeBSD arch
  - Single buildworld and release within PowerPC but different buildkernels (at least for now)
  - FPU problems
  - arch header files
Methodology notes

- Working with the target machine, i.e. not simulator etc.
- JTAG debugger
- Reviewing code of all other FreeBSD architectures
- Reviewing code of other operating systems for reference
- Come up with an optimal route, following FreeBSD kernel APIs and established conventions
Concluding remarks

- References
  - //depot/projects/e500/
  - sys/powerpc/e500
  - sys/powerpc/mpc85xx

- Acknowledgements
  - Ajay Hampapur, Pedro Marques, Marcel Moolenaar @Juniper
  - Marian Balakowicz, Piotr Kruszyński @Semihalf
Embedding FreeBSD/powerpc

*Notes on the journey to the embedded world*

Rafał Jaworowski
raj@semihalf.com


BSDCan 2007, Ottawa