
FreeBSD/PowerPC dla systemów embedded

Rafał Jaworowski
raj@amgua.org

Kraków, 27 Listopada 2004

Spis treści

- ◆ **Wstęp - definicja problemu**
- ◆ **Stan portu FreeBSD/PowerPC**
- ◆ **FreeBSD/PowerPC dla kart CompactPCI**
- ◆ **Środowisko**
- ◆ **Proces uruchomienia systemu (bootstrap)**
- ◆ **Budowa loader-a**
- ◆ **Kernel startup**
- ◆ **Niskopoziomowa architektura FreeBSD**

Wstęp – definicja problemu

- ❖ Cel: FreeBSD na kartach CompactPCI
- ❖ Co to jest system wbudowany (ang. *embedded*)?
 - ◆ Element większej całości
 - ◆ Na ogół wykonuje tylko jedną specjalizowaną aplikację (inaczej niż komputer ogólnego przeznaczenia)
 - ◆ Silna integracja, małe rozmiary, niski pobór energii
 - ◆ Ograniczone interfejsy
 - ◆ Nacisk na niezawodność, bezobsługowość, trwałość (odporność mechaniczną)
 - ◆ Szerokie zastosowania, m.in.: automotive, aircraft, telecom
- ❖ CompactPCI
 - ◆ Standard przemysłowy, część specyfikacji PCI
 - ◆ Sprzęt telekomunikacyjny, carrier grade, highly available
- ❖ Główne problemy portowania
 - ◆ Jak zbudować loader i jądro
 - ◆ Oprogramowanie układów specyficznych dla platformy (mostek host-PCI, kontrolery przerwań, in.)

Stan portu FreeBSD/PowerPC

- ❖ **Architektura**
 - ◆ RISC, superskalarny
 - ◆ Nowoczesny, wydajny, elegancki (liniowy model pamięci, uniwersalne rejestry)
 - ◆ Oparty na architekturze POWER, Apple-IBM-Motorola
- ❖ **Wsparcie dla maszyn Apple Macintosh**
 - ◆ Newworld+
- ❖ **Work-in-progress, braki m.in.:**
 - ◆ KLD, konsola, sterowniki
 - ◆ Lokalne patch-e dla GCC
 - ◆ Jeszcze nie Tier 1
- ❖ **Kernel/loader silnie uwikłane w OpenFirmware**

FreeBSD/PowerPC dla kart CompactPCI

❖ Loader

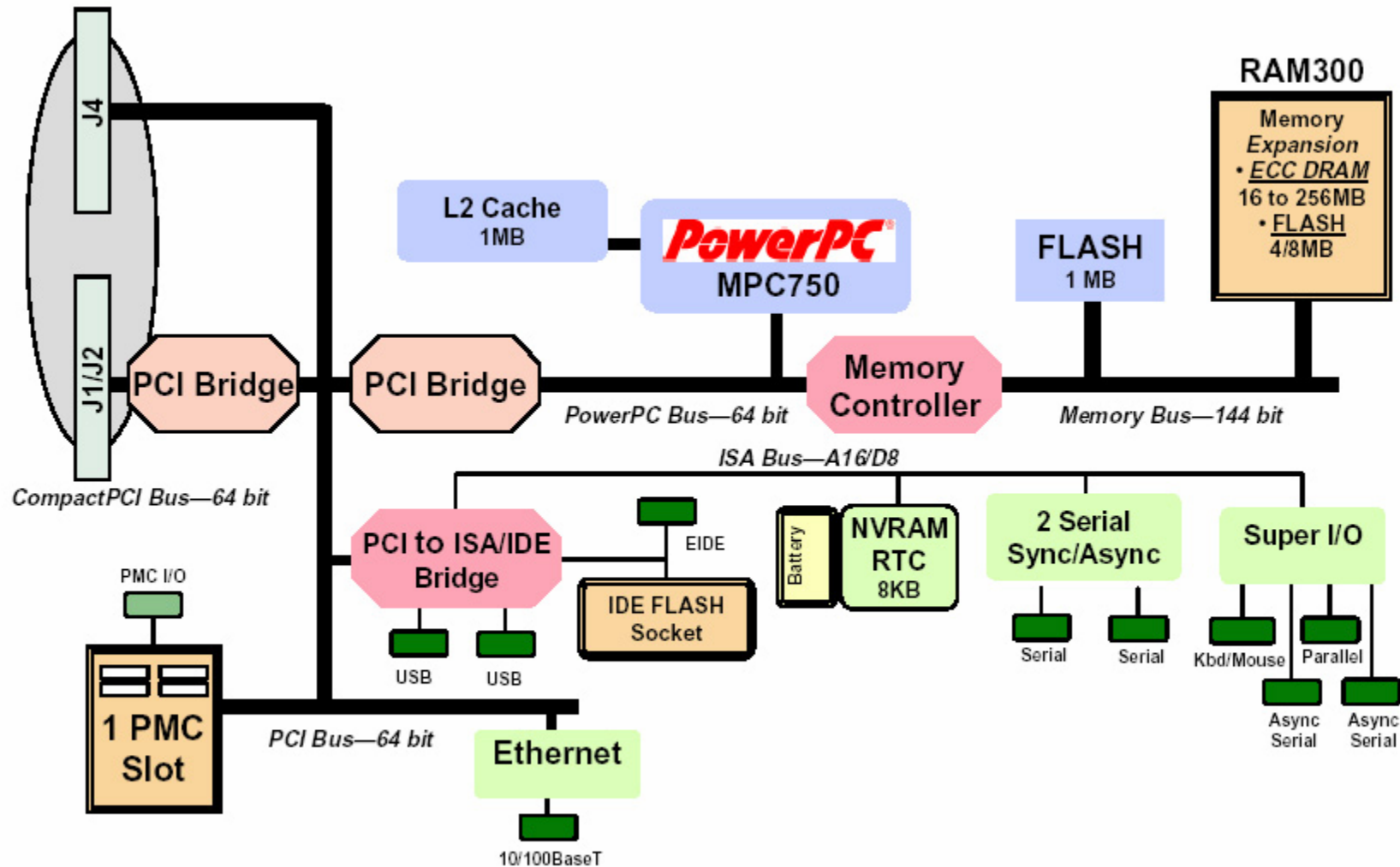
- ◆ Inny firmware (PPCBug vs. OpenFirmware)
- ◆ Odłączenie uzależnienia OF i podłączenie do PPCBug
- ◆ Gotowy, sprawny

❖ Kernel

- ◆ Driver dla mostka host-PCI (Raven) gotowy
- ◆ Driver dla wtórnego kontrolera przerw gotowy (na bazie istniejącego sterownika układów typu OpenPIC)
- ◆ Prace nad sterownikiem dla pierwotnego kontrolera przerw (para 8259)

❖ CPU MPC750 (a.k.a G3), PCI, ISA, USB

FreeBSD/PowerPC dla kart CompactPCI – c.d.



FreeBSD/PowerPC dla kart CompactPCI – c.d.



FreeBSD/PowerPC dla kart CompactPCI – bootlog

Copyright Motorola Inc. 1988 - 2000, All Rights Reserved

PPC1 Debugger/Diagnostics Release Version 4.9 - 07/10/00 RM01
COLD Start

Local Memory Found =10000000 (&268435456)
MPU Clock Speed =233Mhz
BUS Clock Speed =67Mhz

WARNING: Keyboard Not Connected

Reset Vector Location : ROM Bank A
Mezzanine Configuration: Single-MPU
Current 60X-Bus Master : MPU0
Idle MPU(s) : NONE
L2Cache: 1024KB, 117Mhz
System Memory: 256MB, ECC Enabled (ECC-Memory Detected)

PPC1-Bug>nbo

Network Booting from: DEC21140, Controller 0, Device 0
Device Name: /pci@80000000/pci1011,9@e,0:0,0
Loading: loader.bin

Client IP Address = 10.0.0.50
Server IP Address = 10.0.0.3
Gateway IP Address = 0.0.0.0
Subnet IP Address Mask = 255.255.255.0
Boot File Name = loader.bin
[...]

FreeBSD/PowerPC dla kart CompactPCI – bootlog c.d.

[...]

Network Boot File load in progress... To abort hit <BREAK>

Bytes Received =&153264, Bytes Loaded =&153264

Bytes/Second =&153264, Elapsed Time =1 Second(s)

Console: PPCBug console

FreeBSD/PPCBug firmware/PowerPC bootstrap loader, Revision 0.2

(raj@amgua.org, Fri Oct 8 22:51:49 CEST 2004)

Memory: 256MB

network: net0 attached to 00:01:af:01:48:b7

net_open: server addr: 10.0.0.3

net_open: server path: /tftpboot

Loading /boot/defaults/loader.conf

/boot/kernel/kernel data=0x167dd4+0x1d134 syms=[0x4+0x1ed20+0x4+0x27876]

Hit [Enter] to boot immediately, or any other key for command prompt.

Booting [/boot/kernel/kernel] in 8 seconds...

Booting [/boot/kernel/kernel]...

FreeBSD/PowerPC dla kart CompactPCI – bootlog c.d.

```
Booting [/boot/kernel/kernel]...
Kernel entry at 0x11d410 ...
Copyright (c) 1992-2004 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 6.0-CURRENT #82: Sun Oct 24 22:55:28 CEST 2004
Preloaded elf kernel "/boot/kernel/kernel" at 0x2cd000.
Timecounter "decrementer" frequency 16665000 Hz quality 0
cpu0: Motorola PowerPC 750 revision 3.1, 233.34 MHz
cpu0: HID0 809000a4<EMCP,DOZE,DPM,SGE,BTIC,BHT>
real memory  = 268435456 (256 MB)
avail memory = 256135168 (244 MB)
nexus0: <Nexus device>
openpic0: <OpenPIC Interrupt Controller> on nexus0
pcib0: <Motorola Raven Host-PCI bridge> on nexus0
pci0: <PCI bus> on pcib0
ravenpic0: <Motorola Raven PIC> mem 0x3c000000-0x3c03ffff at device 0.0 on pci0
openpic0: Version 1.3, supports 2 CPUs and 16 irqs
pci0: <bridge, PCI-ISA> at device 11.0 (no driver attached)
pci0: <mass storage, ATA> at device 11.1 (no driver attached)
pci0: <serial bus, USB> at device 11.2 (no driver attached)
pci0: <old, non-VGA display device> at device 11.3 (no driver attached)
de0: <Digital 21140A Fast Ethernet> port 0xfffff00-0xfffff7f mem 0x3bffff00-
0x3bffff7f irq 10 at device 14.0 on pci0
de0: [GIANT-LOCKED]
de0: 21140A [10-100Mb/s] pass 2.2
de0: Ethernet address: 00:01:af:01:48:b7
de0: if_start running deferred for Giant
[...]
```

FreeBSD/PowerPC dla kart CompactPCI – bootlog c.d.

```
[...]
pcib1: <PCI-PCI bridge> at device 20.0 on pci0
pcib1:  secondary bus      1
pcib1:  subordinate bus    1
pcib1:  I/O decode         0xf000-0xefff
pcib1:  memory decode     0x3bf00000-0x3befffff
pcib1:  prefetched decode  0xfff00000-0xfffff
pci1: <PCI bus> on pcib1
Timecounters tick every 10.000 msec
de0: enabling Full Duplex 100baseTX port
de0: link down: cable problem?
Root mount failed: 22.

Manual root filesystem specification:
  <fstype>:<device>  Mount <device> using filesystem <fstype>
                    eg. ufs:/dev/da0a
  ?                 List valid disk boot devices
  <empty line>      Abort manual input

mountroot>
```

Środowisko i zależności

- ❖ **Target: system wbudowany, brak monitora, dysku**
 - ◆ Konsola szeregową
 - ◆ Diskless bootstrap (TFTP, NFS)
- ❖ **Host: FreeBSD/i386 5.2.1-p9**
 - ◆ Kompilacja loader-a i jądra
 - ◆ Komunikacja z target-em (RS232, LAN)
 - Serwer TFTP, BOOTP, NFS
- ❖ **Source tree**
 - ◆ 6.0-CURRENT (październik 2004)
 - ◆ Lokalne patch-e dla GCC
- ❖ **Cross-kompilacja**
 - ◆ Narzędzia powstają podczas
`make TARGET_ARCH=powerpc buildworld`

Proces uruchomienia systemu (bootstrap)

- ❖ Trzy etapy boot-owania (nie wszystkie konieczne)
 - ◆ MBR (/boot/boot0)
 - ◆ Bootbloki (/boot/boot1, /boot/boot2)
 - boot2 m.in. „rozumie” filesystem, może bezpośrednio uruchomić kernel, na ogół uruchamia *loader*
 - ◆ Najważniejszy etap trzeci: loader(8)
 - Odczytywany i uruchamiany z filesystem-u (/boot/loader)
 - User friendly
 - Umożliwia kontrolę sposobu uruchomienia [wybranego] jądra, załadowanie/odładowanie modułów etc.
 - Pozwala ustawiać zmienne jądra kenv(2), ręcznie oraz przez device.hints(5)
 - Forth - skrypty startowe (menu z diabełkiem)
 - Najczęściej w postaci „czystego” binarium, ładowanego przez firmware (BIOS)
 - Rozpoznaje format ELF (taką postać ma jądro!)

Budowa loader-a

- ❖ **Statyczne binarium skompilowane pod konkretny adres**
- ❖ **Podstawa – libstand(3)**
 - ◆ Podstawowe funkcje dla samodzielnych aplikacji (działających bez systemu operacyjnego)
 - ◆ `printf()`, `malloc()/free()`, `setenv()` itp.
 - ◆ Implementacja protokołów sieciowych BOOTP, BOOTPARAMS, NFS
- ❖ **Dwie części: zależna od architektury (BIOS-u, firmware-u) i niezależna**
 - ◆ Należy „zaopatrzyć” część niezależną (libstand et al.) w niskopoziomowe funkcje elementarne zależne od architektury (`putchar()`, `getchar()`, `devopen()`, `devclose()` itd.)
 - ◆ Na zewnątrz loader(8) prezentuje taki sam interfejs niezależnie od architektury
 - Komendy `lsmod`, `lsdev`, `boot`, `set` i in.
 - ◆ Entry point dla loader-a
 - Niewielka część w asemblerze („klej”)
 - Ustawia rejestry CPU w stan początkowy (m.in. wyłącza obsługę przerwań, wyłącza translacje MMU)

Budowa loader-a c.d.

❖ Modyfikacje do celów PPCBug

- ◆ Zapewnienie wysokopoziomowej części loader-a dostępu do firmware-u
- ◆ Funkcja wołania „systemowego” PPCBug
- ◆ Obsługa niskopoziomowa sieci (wysłanie, odebranie ramki, w trybie pollingu) – do użycia przez protokoły BOOTP, NFS itd.

❖ Referencje

- ◆ http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/boot.html
- ◆ http://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/boot.html
- ◆ **man**
 - loader(8)
 - boot(8)
- ◆ `/usr/src/sys/boot/`

Kernel startup

❖ Jądro jest *prawie* zwyczajnym binarium w formacie ELF

- ◆ `$file /tftpboot/boot/kernel/kernel`
`/tftpboot/boot/kernel/kernel: ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (FreeBSD), dynamically linked (uses shared libs), not stripped`
- ◆ `$ readelf -l /tftpboot/boot/kernel/kernel`
Elf file type is EXEC (Executable file)
Entry point 0x11d410
There are 4 program headers, starting at offset 52
Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
PHDR	0x000034	0x00100034	0x00100034	0x00080	0x00080	R E	0x4
INTERP	0x0000f4	0x001000f4	0x001000f4	0x0000d	0x0000d	R	0x1
[Requesting program interpreter: <code>/red/herring</code>]							
LOAD	0x000000	0x00100000	0x00100000	0x167e50	0x184f80	RWE	0x10000
DYNAMIC	0x167784	0x00267784	0x00267784	0x00068	0x00068	RW	0x4
- ◆ `$ readelf -d /tftpboot/boot/kernel/kernel`
Dynamic segment at offset 0x167784 contains 8 entries:

Tag	Type	Name/Value
0x00000001	(NEEDED)	Shared library: <code>[hack.So]</code>
0x00000004	(HASH)	0x100104
0x00000005	(STRTAB)	0x112098
0x00000006	(SYMTAB)	0x105438
0x0000000a	(STRSZ)	45944 (bytes)
0x0000000b	(SYMENT)	16 (bytes)
0x00000015	(DEBUG)	0x0
0x00000000	(NULL)	0x0

Kernel startup c.d.

- ❖ Po wejściu do jądra, loader znika z horyzontu

- ◆ Pamięć używana do innych celów, zamazywana

- ❖ Entry point jądra

- ◆ /sys/powerpc/powerpc/locore.S (lub /sys/i386/i386/locore.s)

- ◆ Ustawia CPU w początkowy stan, potem

```
[...]
        bl      powerpc_init
        bl      mi_startup
[...]
```

- ◆ lub (i386)

```
[...]
        call   init386
[...]
        call   mi_startup
[...]
```

- ◆ Zależny od architektury init

- M.in. inicjuje CPU, kernel stack, struktury jądra, VM (pmap), konsolę itd.

Kernel startup c.d.

❖ Entry point jądra c.d.

◆ mi_startup() – Machine Independent

- Nie wraca!
- Uruchamia w pętli wszystkie podsystemy z infrastruktury SYSINIT
 - `SYSINIT(announce, SI_SUB_COPYRIGHT, SI_ORDER_FIRST, print_caddr_t, copyright)`
- `/usr/src/sys/sys/kernel.h` zawiera identyfikatory podsystemów `SI_SUB_XXX`
- Jako ostatni podsystem uruchamiane są moduły typu `SI_SUB_RUN_SCHEDULER`. Odpowiednio tworzą wątek/proces `[schedcpu]` oraz uruchamiają `[swapper]`

Kernel startup c.d.

◆ Istotne podsystemy z punktu widzenia urządzeń

```
[...]  
SI_SUB_KLD           = 0x2000000,    /* KLD and module setup */  
[...]  
SI_SUB_INTR         = 0x2800000,    /* interrupt threads */  
SI_SUB_SOFTINTR     = 0x2800001,    /* start soft interrupt thread  
[...]  
SI_SUB_DRIVERS      = 0x3100000,    /* Let Drivers initialize */  
SI_SUB_CONFIGURE    = 0x3800000,    /* Configure devices */  
[...]
```

◆ Ostatni krok – uruchomienie `init(8)`, pierwszego procesu użytkownika

- Uruchamia RC
- Kończy się przejściem do trybu multiuser

◆ Referencje

- http://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/boot-kernel.html
- `/usr/src/sys/powerpc/`

Niskopoziomowa architektura FreeBSD

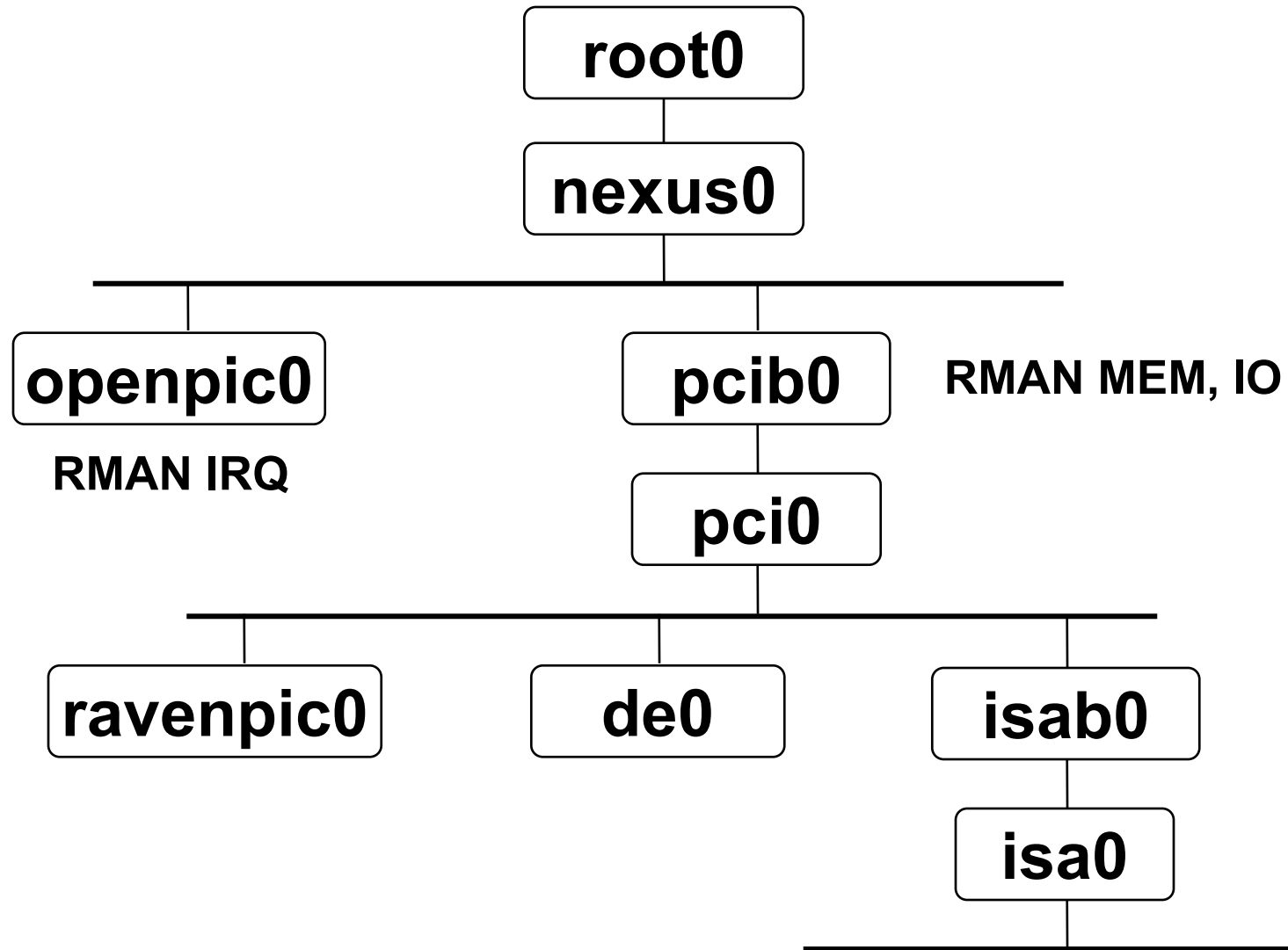
❖ NEWBUS

- ◆ Infrastruktura podłączania urządzeń zewnętrznych do systemu
- ◆ Zorientowana obiektowo
- ◆ Hierarchia (magistrale i urządzenia końcowe)
- ◆ Opisuje *potencjalne* wsparcie jądra dla urządzeń (statycznie wkompilowane lub załadowane z modułu)
- ◆ UWAGA: w rzeczywistości urządzenia wspierane przez jądro mogą fizycznie nie istnieć (i odwrotnie: fizycznie podłączone urządzenie jest bezużyteczne bez wsparcia jądra w postaci sterownika)
- ◆ Każde urządzenie zarejestrowane w hierarchii zostanie podczas autokonfiguracji spróbkowane i – jeśli się zgłosi – zainicjowane

❖ Struktura urządzeń

- ◆ root0 – wirtualne urządzenie pierwotne, tworzone statycznie
- ◆ nexus0 – wirtualna, abstrakcyjna magistrala, dodawana jawnie jako potomek root0
- ◆ Pozostałe urządzenia dodawane są do hierarchii automatycznie w fazie probe/attach, po zadeklarowaniu magistrali macierzystej danego urządzenia w jego sterowniku

Niskopoziomowa architektura FreeBSD



Niskopoziomowa architektura FreeBSD c.d.

❖ Autokonfiguracja

- ◆ Dwie fazy: probe, attach
- ◆ Moduł SI_SUB_CONFIGURE
- ◆ Dla każdego istniejącego w hierarchii driver-a urządzenia wołane są metody probe i attach
 - probe - próbuje urządzenie (sprawdza czy fizycznie istnieje - odczyt z rejestrów, vendor ID itp.)
 - attach - inicjuje (tylko jeśli urządzenie się zgłosi w fazie probe), alokuje zasoby dla urządzenia (IO, MEM, IRQ, DMA) od magistrali macierzystej np. pci0

❖ Moduł sterownika

- ◆ Deklaruje *klasę* (typ) urządzenia
- ◆ Na podstawie tej deklaracji podczas probe/attach tworzona jest *instancja* tego driver-a i przyłączana do hierarchii NEWBUS

Niskopoziomowa architektura FreeBSD c.d.

❖ Moduł sterownika - przykład

```
static device_method_t tulip_pci_methods[] = {
    /* Device interface */
    DEVMETHOD(device_probe,      tulip_pci_probe),
    DEVMETHOD(device_attach,    tulip_pci_attach),
    DEVMETHOD(device_shutdown,  tulip_shutdown),
    { 0, 0 }
};

static driver_t tulip_pci_driver = {
    "de",
    tulip_pci_methods,
    sizeof(tulip_softc_t),
};

static devclass_t tulip_devclass;
DRIVER_MODULE(de, pci, tulip_pci_driver, tulip_devclass, 0,
0);
```

Dziękuję za uwagę.

Rafał Jaworowski

Kraków, 27 Listopada 2004