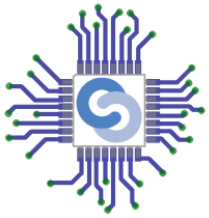


Calculatoare Numerice (2)

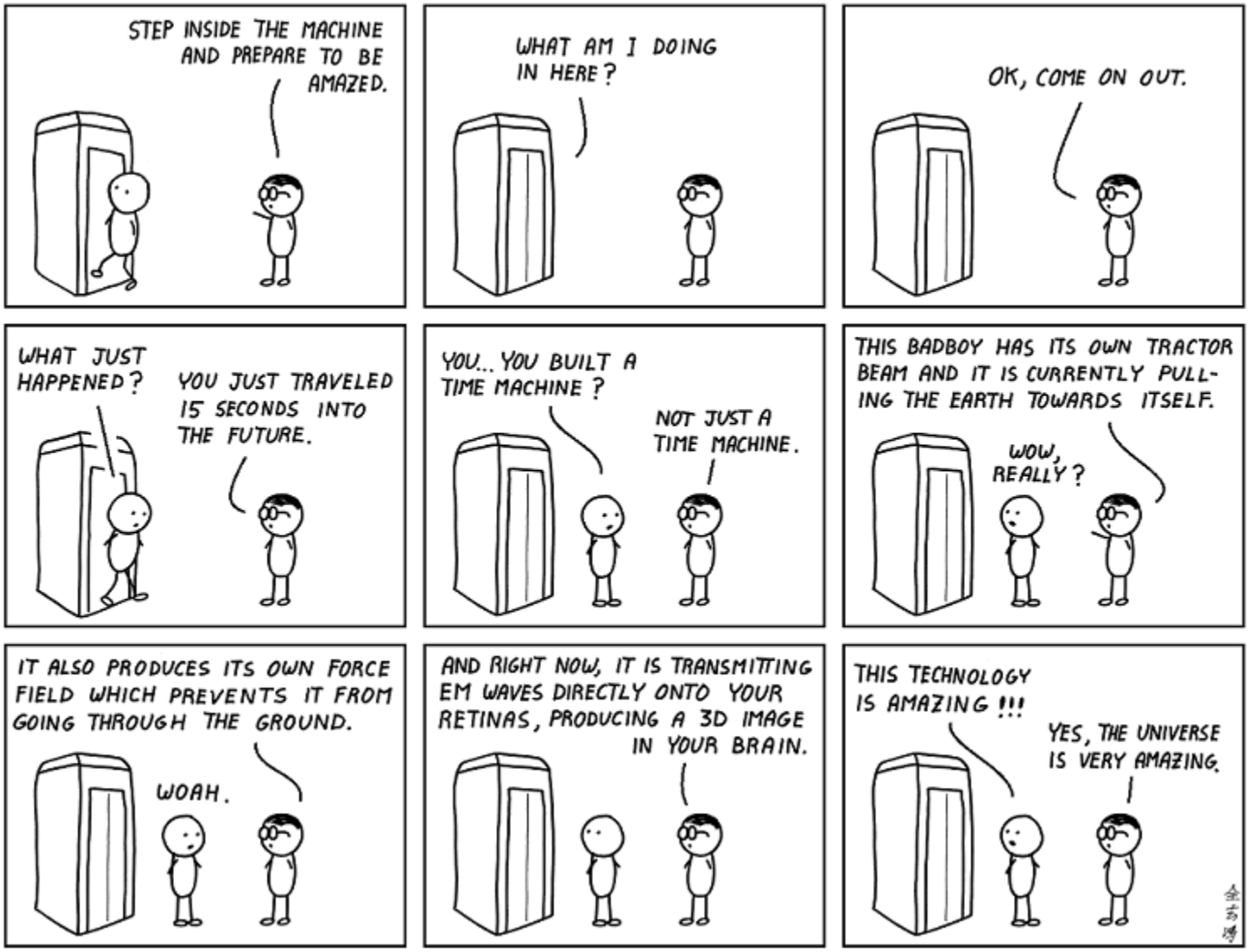
- **Cursul 8 –**

DMA și PCI(e)

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

Comic of the day

Embedded Systems Laboratory

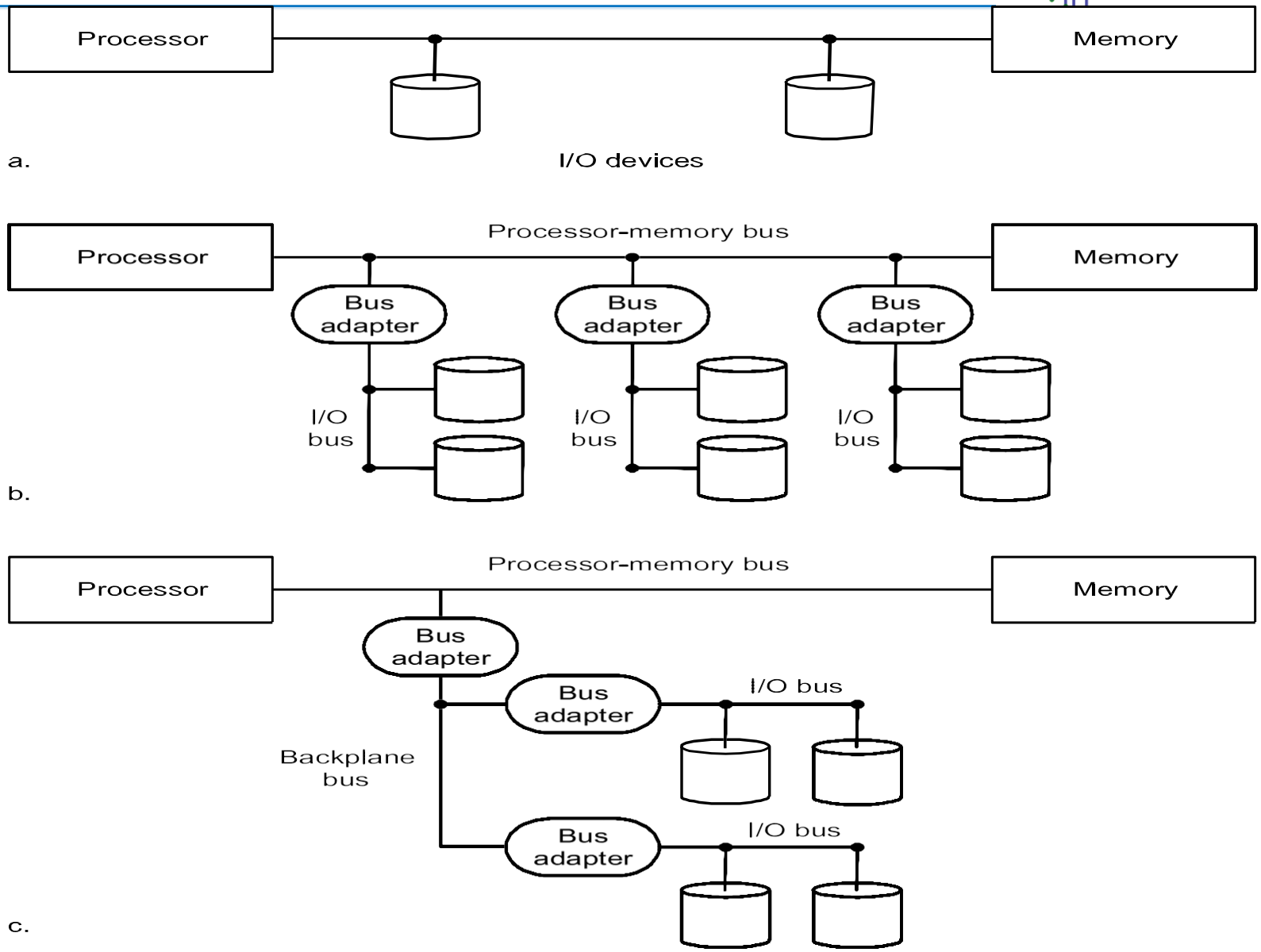


Din episodul anterior

1. **Processor-memory (local) bus:**
   - De obicei scurt și de viteză mare pentru a maximiza lățimea de bandă
2. **I/O Bus**
   - Lung, trebuie să suporte viteze de date diferite de la dispozitive diferite
   - Trebuie să facă față unei game largi de latențe, lățimi de bandă și specificații
3. **Backplane Bus**
   - Este numit așa pentru că era la început pe spatele șasiului
   - Permite interconectarea memoriei, procesorului și a dispozitivelor I/O
   - Necesită logică suplimentară pentru a se lega la alte magistrale

- Magistralele locale sunt de obicei design-specific în timp ce cele de I/O și backplane sunt portabile și de obicei respectă un standard industrial
- Un sistem poate să folosească un backplane sau o combinație de toate trei

Configurații de bus



Exemplu de calcul performanță

Embedded Systems Laboratory

Un bus sincron are o perioadă de ceas de 50ns și fiecare transmisie durează un ciclu de ceas. Alt bus asincron are nevoie de 40ns pentru fiecare handshake. Fiecare bus are 32 de biți de date. Care este lățimea de bandă a fiecărui bus pentru citiri de un cuvânt dintr-o memorie cu 200ns per read.

Răspuns:

Pașii pentru magistrala sincronă:

1. Trimite adresa la memorie: 50 ns
2. Citește memoria: 200 ns
3. Trimite datele la dispozitiv: 50 ns

} 300 ns

Lățimea maximă de bandă este 4 octeți la 300 ns $\Rightarrow \frac{4 \text{ bytes}}{300 \text{ ns}} = \frac{4 \text{ M B}}{0.3 \text{ sec}} = 13.3 \text{ MB/sec}$

Exemplu de calcul performanță

Secvența de pași pentru magistrala asincronă:

1. Memory read address atunci când apare ReadReq : 40 ns
2,3,4. Data ready & handshake: max(3x40 ns, 200 ns)= 200 ns
5,6,7. Read & Ack. : 3 × 40 ns = 120 ns

} 360 ns

Lățimea maximă de bandă este 4 octeți la 360 ns $\Rightarrow \frac{4 \text{ bytes}}{360 \text{ ns}} = \frac{4 \text{ MB}}{0.36 \text{ sec}} = 11.1 \text{ MB/sec}$

Creșterea lățimii de bandă

- Lățimea de bandă este de obicei determinată de protocol și de caracteristicile de temporizare a semnalelor
- Alți factori:
  - Lățimea magistralei de date:
    - Transferă cuvinte multiple, necesită mai puțini cicli de bus
    - Creșterea numărului de linii de date (scump!)
  - Multiplexarea liniilor de adrese și date:
    - Folosirea liniilor separate de date și adrese accelerează tranzacțiile
      - Simplifică logica de control pe bus
      - Mărește numărul de linii de date
  - Transfer pe blocuri:
    - Transferă cuvinte multiple de la adrese consecutive în rafală
    - Mărește timpul de răspuns deoarece tranzacțiile vor fi mai lungi
    - Mărește complexitatea logicii de control a magistralei

Bus Master

- Single master
  - Bus master (ex. procesorul) controlează toate accesele pe bus
  - Bus slave (ex. device sau memorie) doar răspunde la cereri
- Multiple master
  - Mai multe dispozitive pot iniția o tranzacție
  - Bus control logic și protocolul adoptat rezolvă conflictele

Arbitrare

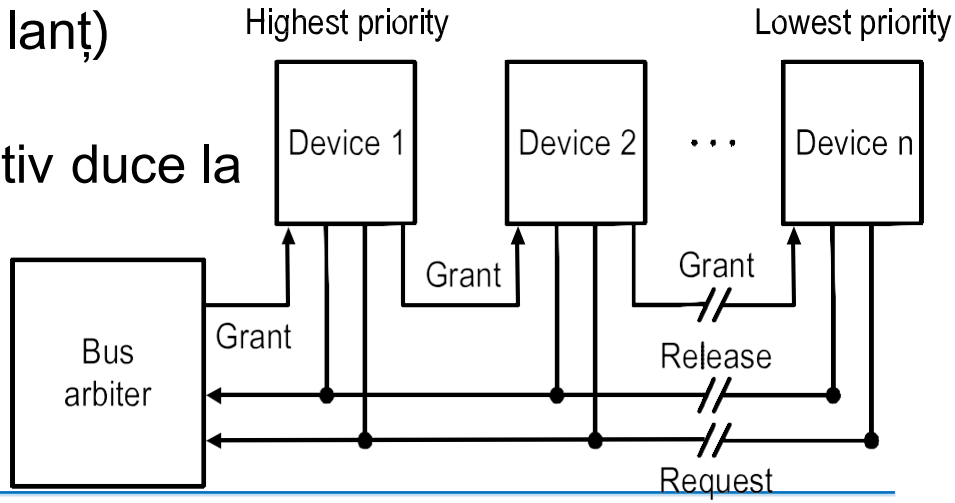
- Arbitrarea pe bus coordonează utilizarea magistralei folosind mecanisme de request, grant, release
- Arbitrarea încearcă de obicei să echilibreze doi factori atunci când alege dispozitivul care preia magistrala:
  - Dispozitivele cu prioritate mai mare trebuie servite primele
  - Menține corectitudinea; nici un dispozitiv nu va fi reprimat total
- Timpul de arbitrare este un overhead
  - Vrem să îl minimizăm

Tehnici de arbitrare

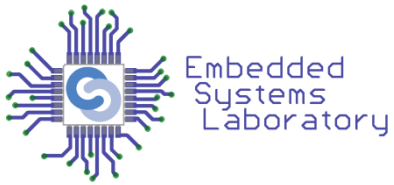
- *Distributed arbitration by self-selection:* (ex. NuBus folosit la Apple Macintosh)
  - Folosește linii multiple de request pentru dispozitive
  - Dispozitivele care fac request determină cui i se va da acces
  - Dispozitivele asociază un cod cu cererea, indicând prioritatea
  - Dispozitivele cu prioritate mică cedează busul dacă observă o cerere de prioritate mai mare
- *Distributed arbitration by collision detection:* (ex. Ethernet)
  - Dispozitivele cer independent acces la bus și preiau accesul
  - Cereri simultate produc o coliziune
  - Un algoritm de selecție între entitățile care au generat coliziunea
    - Ethernet: back off and try again later

Tehnici de arbitrare (Cont.)

- Daisy chain arbitration: (e.g. VME bus)
  - Linie specială de Grant Access, partajată de toate dispozitivele în ordinea priorității
  - Dispozitivele de prioritate mai mare arbitrează cererile dispozitivelor cu prioritate mai mică
  - Simplu de implementat
  - Duce la starvation pentru dispoztitivele de prioritate mică
  - Limitează viteza transmisiilor (semnalele de grant ajung cel mai târziu la ultimul device din lanț)
  - Permite propagarea defectelor
    - (o defecțiune a unui dispozitiv duce la defectarea întregului lanț)



Rolul sistemului de operare



- Sistemul de operare are rol de interfață între I/O hardware și programe
- Caracteristici importante ale sistemelor cu I/O:
  - Sistemul I/O este partajat între mai multe programe
  - Sistemele I/O folosesc întreruperi pentru a comunica starea lor procesorului
    - Întreruperile trebuie să fie tratate de SO pentru că transferă execuția în modul supervizor
  - Controlul low-level al unui dispozitiv I/O este complex:
    - Evenimente concurente
    - Cerințele pentru controlul corect al dispozitivului pot să fie foarte complexe

Responsablitățile SO

- Protecție pentru resursele I/O partajate
  - Garantează că un proces nu poate să acceseze I/O-ul altor procese
- Abstractizare pentru accesarea dispozitivelor
  - Rutine care supervizează operarea low-level a dispozitivului (drivere).
  - Interrupt handling pentru I/O
  - Acces echitabil la resursele I/O
    - Toate programele trebuie să aibă acces egal la resursele I/O
  - Planifică accesele pentru a mări productivitatea sistemului

Comunicația cu dispozitivele I/O

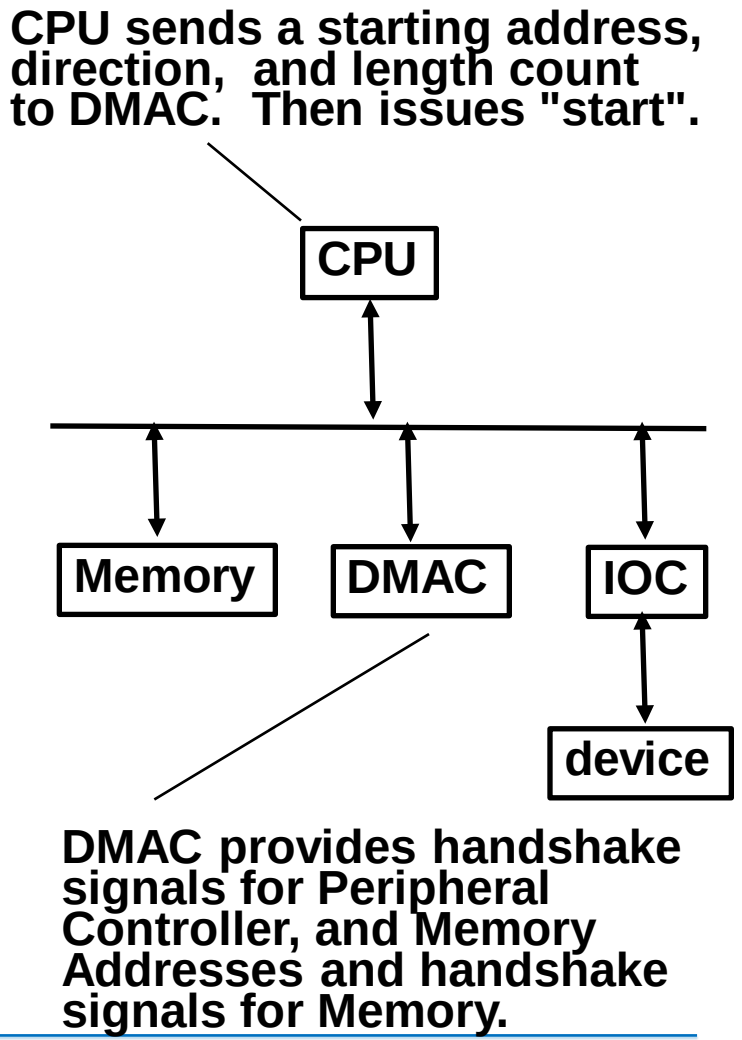
- SO trebuie să știe când:
  - Dispozitivul I/O a terminat o operație
  - Dispozitivul I/O a întâlnit o eroare
- Poate fi realizat în două moduri:
  - Polling:
    - Dispozitivul I/O pune informația de stare într-un registru
    - SO verifică periodic registrul de stare
  - I/O Interrupt:
    - Eveniment extern, asincron execuției principale, dar care NU interferează cu rularea codului principal
    - Ori de câte ori un dispozitiv I/O device cere atenție de la procesor, îl întrerupe
    - Unele procesoare tratează întreruperile ca pe niște excepții speciale

Slide: in part David Patterson

Direct Memory Access

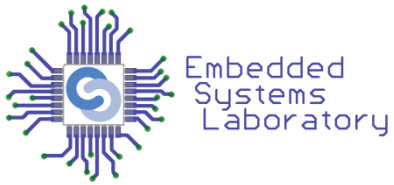
Embedded Systems Laboratory

- Direct Memory Access (DMA):
  - Extern CPU-ului
  - Folosește ciclii nefolosiți de pe bus (cycle stealing)
  - Funcționează ca master pe bus
  - Transferă blocuri de date spre și de la memorie fără intervenția CPU
  - Eficient pentru transmisia de mari cantități de date, ex. de pe disc
  - Existența cache permite procesorului să lase lățime de bandă destulă pentru DMA



* Figure is courtesy of Dave Patterson

Funcționarea DMA

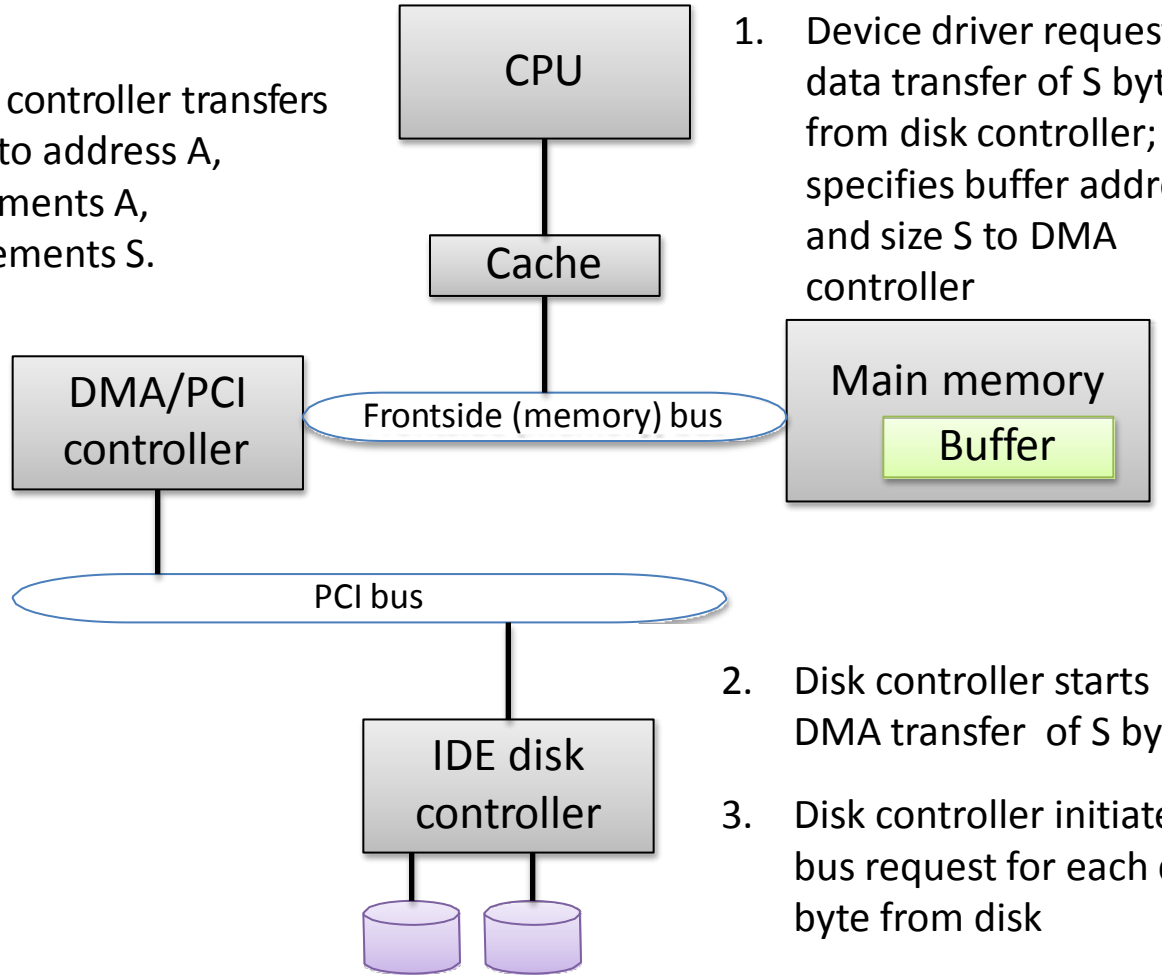


- CPU inițializează device ID, adresele din memorie și numărul de octeți de transferat
- DMA controller (DMAC) inițiază accesul și devine bus master
- Pentru transferul mutiplu de date, DMAC incrementează adresele
- DMAC întrerupe CPU la terminarea transferului

Direct Memory Access

- Evită *I/O programat* pentru majoritatea datelor
  - De ex. rețea rapidă sau interfețe de disc
- Necesită un *controller DMA*
  - De obicei implementat în procesor, în zilele noastre
- Ocolește CPU și transferă datele direct de la I/O la memorie
  - Nu blochează CPU-ul
  - Salvează lățimea de bandă
  - Doar o singură întrerupere pe transfer

Transfer DMA old-style



1. Device driver requests data transfer of S bytes from disk controller; specifies buffer address A and size S to DMA controller

2. Disk controller starts DMA transfer of S bytes

3. Disk controller initiates bus request for each data byte from disk

4. DMA controller transfers byte to address A, increments A, decrements S.

5. When S == 0, DMA controller interrupts CPU to indicate transfer complete

Avantajul de bază al DMA

- **Decuplează** transferul de date de procesarea de date
  - CPU nu trebuie să copieze datele de la/la dispozitiv
  - Nu poluează cache-ul CPU
  - Pot fi procesate când decide CPU (sau SO)
  - Performanță mărită: CPU și device I/O merg în paralel
- Dezavantaje posibile:
  - Overhead mare pentru transferuri foarte mici de date
  - De obicei nu e o problemă (până și UART-urile fac DMA!)

DMA și memoria Cache

- DMA înseamnă ca memoria devine **inconsistentă** cu cache-ul CPU
- Opțiuni:

1. CPU poate să marcheze bufferele DMA ca non-cacheable

⇒ large hit – probabil vrea să proceseze datele oricum

2. Cache poate să facă “snoop” la tranzacțiile DMA (dar nu scalează foarte bine la sistemele multiprocesor)
3. SO poate să golească/invalideze explicit regiuni din cache

⇒ cache management este o parte importantă a driverelor de dispozitive!

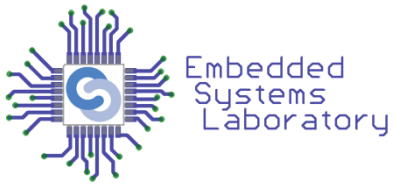
Probleme DMA: Cache

- Două copii ale datelor
  - Procesorul poate să nu știe că liniile din cache și memorie sunt diferite
  - Write-back cache poate să suprascrie datele I/O sau face DMA să citească datele care nu trebuie
- Soluții
  - Rutează activitățile I/O prin cache
    - Nu este eficient pentru că datele din I/O nu posedă localitate temporală
  - SO invalidează selectiv blocurile din cache înainte de I/O read sau forțează write-back înainte de I/O write
    - Se cheamă cache flushing și necesită suport hardware

Probleme DMA: VM

- Paginile au adrese fizice și virtuale foarte diferite
  - Paginile fizice se re-mapează la alte pagini virtuale în timpul operațiilor DMA
  - Multi-page DMA nu poate asigura existența unuor adrese fizice consecutive
- Soluții
  - Permite DMA bazat pe adresare virtuală
    - Adaugă logică de translație controllerului DMA
    - Paginile alocate de SO pentru DMA previn re-maparea până când DMA termină transferul
  - DMA partajat
    - Sparge transferul DMA în mai multe operații, fiecare transfer e o singură pagină
    - SO înlănțuie paginile

DMA și Memoria Virtuală



- Adresele DMA sunt **fizice**
  - Apar pe magistrala externă
- Codul utilizatorilor și al SO lucrează cu adrese **virtuale** (în majoritatea timpului)
- SO (și device drivers) trebuie să translateze manual virtual ↔ fizic atunci când programează controllerele DMA
  - Acest lucru necesită mai mult de o tablelă hardware de pagini!
  - DMA al unei singure regiuni de adrese virtuale s-ar putea să **nu aibă un corespondent contiguu** în spațiul de adrese fizice
  - **Scatter-gather** DMA controllers: DMA de la/la o listă de regiuni
- Sisteme foarte recente: implementează IOMMU
  - Funcționează ca MMU, dar pentru DMA scrie din dispozitive
  - Tot trebuie programat de SO pentru a fi în concordanță cu starea MMU

Unde sunt toate aceste registre?

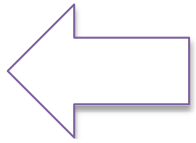
- De unde știe SO câte dispozitive I/O sunt conectate?
- Unde sunt mapate registrele dispozitivelor în spațiul fizic?
  - Și căror vectori de întrerupere le corespund?
- Soluție: include-le în designul *magistralei de I/O*
  - Exemplu: PCI

PCI este…

Embedded Systems Laboratory

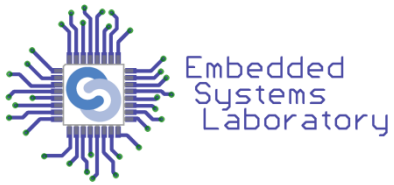
Peripheral **C**omponent **I**nterconnect

- Standard electric pentru conectarea dispozitivelor
  - Ca și PCMCIA, PCI-X, PCI-Express, etc.
- Un standard pentru conectorii fizici
- Un set de ”protocoale de bus” pentru comunicația dintre dispozitive
- O interfață vizibilă software-ului pentru operații I/O hardware



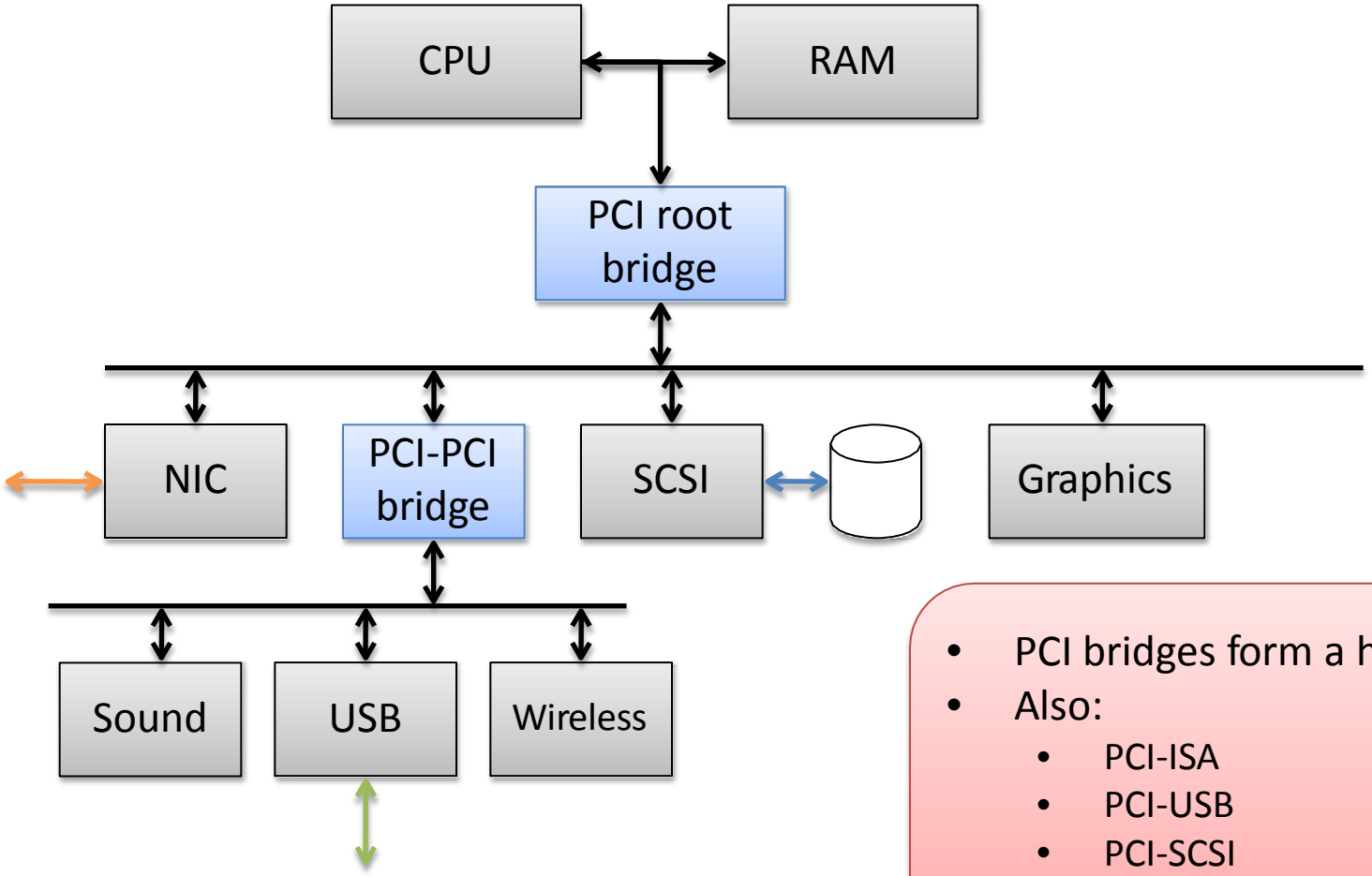
PCIe a urmat PCI, dar extinde aceeași interfață software

PCI încearcă să rezolve probleme multiple:



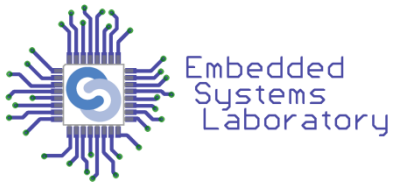
- Device discovery
  - Detecția tuturor dispozitivelor din sistem
- Alocarea de adrese
  - La ce adrese apar diferitele registre ale dispozitivelor conectate?
- Interrupt routing
  - Ce semnale de întrerupere de la fiecare dispozitiv se mapează și în ce vector de excepție?
- DMA inteligent
  - Dispozitive ce au “Bus mastering” nu mai au nevoie de controller DMA

Conexiuni fizice: PCI este un arbore



- PCI bridges form a hierarchy
- Also:
  - PCI-ISA
  - PCI-USB
  - PCI-SCSI
  - Etc.

Spațiul de adresă PCI este plat



- Fiecare device PCI cere un set de adrese
  - Spațiu fizic de adrese (32-biți sau 64-biți)
  - Spațiu adresă I/O (de obicei 16-biți)
- Bridges up the tree remap addresses to the device
- Rezultat:
  - Fiecare dispozitiv apare ca un segment contiguu de adrese
    - În spațiul de memorie
    - În spațiul I/O (doar pe x86)

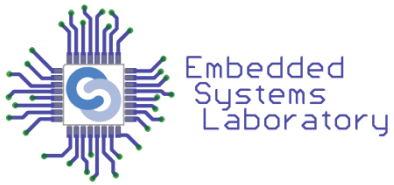
Dispozitivele PCI sunt self-describing

- Fiecare dispozitiv are un header pentru configurație
  - Accesat prin bridge-ul părinte, inițial
- Câteva câmpuri:

Bits	Description
16	Manufacturer ID (idenfies Intel, 3Com, NVidia, etc.
16	Model ID (specific to manufacturer)
24	Class code (what kind of device is this?)
8	Version identifier

- Plusuri:
  - Alocă automat spațiul de adresă
  - Intreruperi
  - Informații de natură electrică
  - Etc.

Localizarea tuturor dispozitivelor



- Găsește bridge-ul PCI “root”
  - PCI bridge este în vârful arborelui
  - PC-urile mari pot avea mai mult de unul
- Citește configurațiile pentru a găsi toate dispozitivele atașate
  - Adaugă la lista de dispozitive și funcții
  - Înregistează cerințele pentru spațiul de adresă
  - If a bridge, recurse!
- Rezultatul:
  - Lista completă a tuturor dispozitivelor din sistem cu toate cerințele de spațiu de adresă aferente

Alocarea adreselor

- Găsește adresele pentru fiecare dispozitiv și bridge
  Cerințele includ:
  - Fiecare dispozitiv are dată dimensiunea spațiului de adresă necesar
  - Toate dispozitivele de “sub” un bridge au adrese care sunt incluse în spațiul de adresă al bridge-ului
  - Fiecare bridge are un segment de memorie care include toate segmentele de memorie ale tuturor “copiilor”.
  - Fiecare segment este limitat de adrese putere a lui 2
- Apoi programează:
  - Fiecare bridge PCI cu informații legate de translatarea adreselor
  - Fiecare dispozitiv cu registre “base-address/range” (BAR)

Întreruperi PCI

- Patru linii de întrerupere
  - INTA, INTB, INTC, INTD…
  - Bridge-urile permit cablarea arbitrară a liniilor de IRQ a dispozitivelor la cele patru linii
  - Translatate de root bridge în intreruperi de sistem
- PCI Express introduce MSI
  - Message-signalled interrupts
  - Întreruperea codificată ca un PCI write într-un spațiu anume de adrese
  - Translatate de root bridge în intreruperi de sistem
  - Întreruperile pot fi rutate individual către un anumit core/APIC

DMA peste PCI

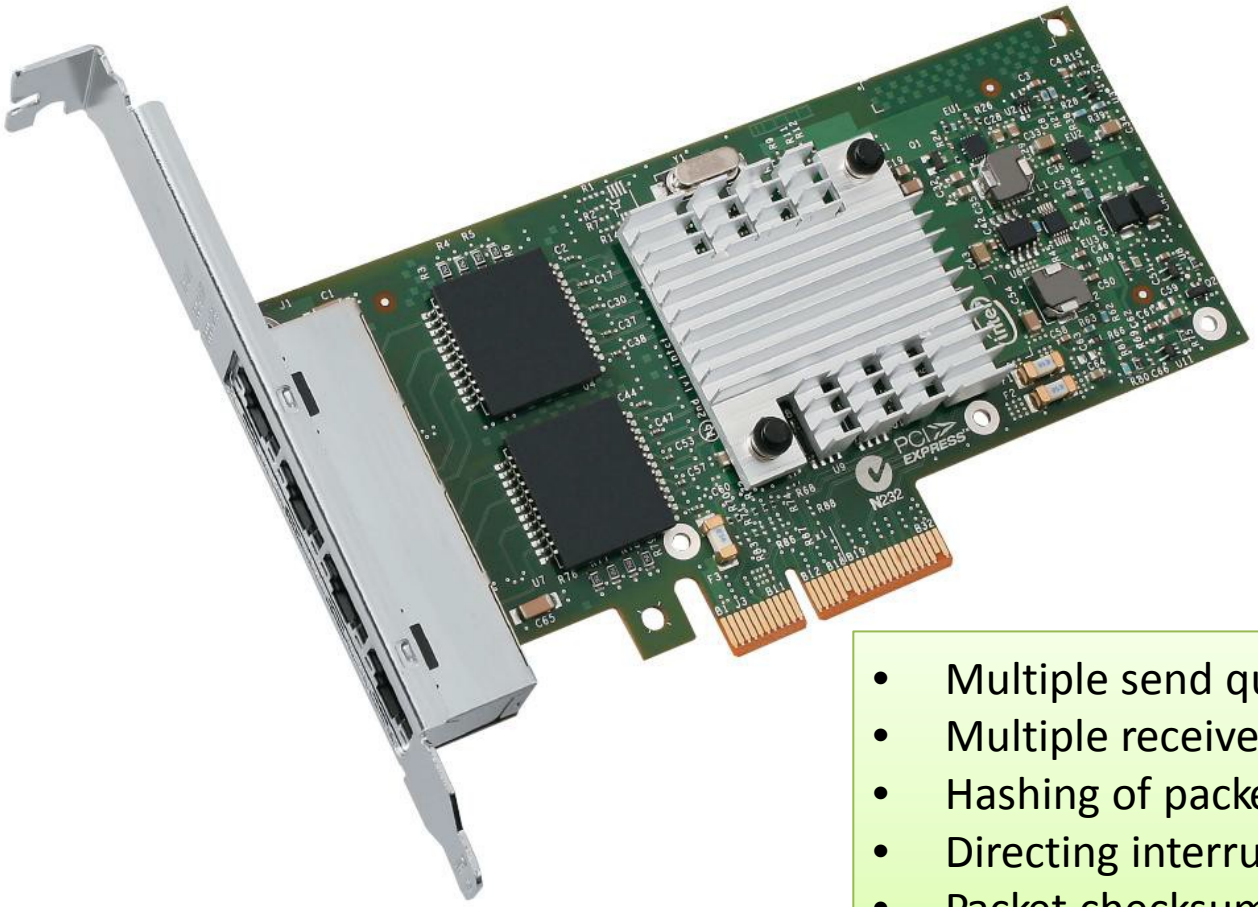
Embedded Systems Laboratory

- PCI permite **Bus Mastering**
  - Device-ul poate emite tranzacții de scriere/citire oriunde în memorie
  - Chiar (în anumite cazuri) spre alte dispozitive PCI
- Controller-ul DMA extern nu mai e relevant
  - Controller integrat în dispozitivul însuși
  - Principiul se aplică: device-ul face DMA pentru date de la/la memorie
  - Mult mai flexibil / dispozitive inteligente

Dispozitive inteligente

- Bus mastering, plus foarte mult spațiu de adresă acum
- Dispozitivele pot acum să acceseze autonom orice:
  - Locație din memoria principală
  - Alte dispozitive
- Permite protocoale complexe pentru interacțiunea CPU↔Device
  - Încearcă să țină și CPU și device-ul ocupat în perioadele de activitate mare
  - In RAM buffering
  - “Descriptor rings” – mecanism de schimbare de cereri și răspunsuri

Exemplu: Intel e1000 PCI- Express Ethernet card



- Multiple send queues
- Multiple receive queues
- Hashing of packet headers to queues
- Directing interrupts to different cores
- Packet checksumming in hardware
- etc.

Rezumat

- Dispozitivele și CPU comunică via:
  - Registre I/O mapate în memorie
  - Întreruperi și vectori de întrerupere
  - Direct Memory Access (DMA)

- Magistralele I/O (precum PCI):
  - Permit dispozitivelor să partajeze/aloce adrese fizice
  - Alocă întreruperi
  - Permit bus mastering direct memory access

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252