





(2.)

## RENDSZERARCH.

- kimeneti fellet  $\rightarrow$  nyitott ballastor  
 $\rightarrow$  nemül persh-pull
- kevés külső csatlakozás

(1.) INDUKTÍV V. TRANSZFORMATORÁS:

- pulzus áramú áramú szinguláris helyettesítés  
 $\phi$  megváltozása
- egyenlítő / alvó / de ballastorok minimálisra állítása
- sebesség / áramsebesség ( $\approx 10 \text{ kHz}$ )
- figyelem kell az impedanciaillikcióra

(2.) ÚJABB MŰKÖDÉS:

- mind amilyen felvezető tulajdonságú épül
- jól integrálható, a kábelek alkatrészeihez illeszkedő  
 interfészekkel
- az esetben a fűtési interfésze is integrálható van  
 (RS485, PROFIBUS)  $\leftarrow$  megfelelőjeje jelölésnek

(3.1.)

KAPACITÍV CSATLAKOZÁS:

- Texas Instruments (ISO 72 xx)
- alapvetően nagyfeszültségű áramú szinguláris  
 lényegében megismerés egy PWM modulál.

$$\left[ \begin{array}{c} M \\ M \\ M \end{array} \right] \rightarrow \boxed{\text{SLA284}}$$

$$\left[ \begin{array}{c} M \\ M \\ M \end{array} \right]$$

(3.2.)

ERŐS MÁGNESES HATÁSOK ALAPJEL  
SÜGÉRTELT FELTÉTEL:

az új fűtési jelenség (1988 feljel.  $\rightarrow$  2007. október)

$\downarrow$   
 az alapvető alkatrészek: DVE Compensation  
 ISOloop

- DC áramú is lehetséges

(3.3.)

felhasználás  $\rightarrow$  PF áramú szinguláris áramú  
 csatlakozás

- RF protokoll: egyenlő ON/OFF jellegű

- Reaktorvezérlő:

- nagy választék

- működik az aktuális manipulátor mellett a legközelebbi választási

- zártkörű

## Rendszerarchitektúrák előadás

2010.március 11.

AXI

IBM CoreConnect

PowerPc alapú beágyazott rendszerek

- kifejezetten a PLB buszra csatlakozó, illetve OPB
- semmi egyéb tulajdósága nincs, ezek a buszok szabják meg

PowerPC busz rendszer

- PLB: 128 ill. 256 bit
- OPB: 64 bites
- mindkettő 32 bites címtartománnyal rendelkezik

MicroBlaze: lágy proc.

- régebben nem rendelkezett PLB interfésszel
- nem indokolta semmi, hogy kialakítsák rajta (5-6 évvel ezelőtt)
  - az az áramkör, amely a processzor és a PLB busz között lett volna, nagyon bonyolult lett volna
  - inkább egy a kisebb komplexitású buszra illesztették
  - azóta eltelt 6 év, mostmár az OPB busz nem játszik szerepet
  - több, bonyolultabb memóriaillesztőre lenne szükség
  - a régi rendszereket így újra kell fordítani
- egyszerűbb, egy szintű busz struktúrát lehet megvalósítani

CoreConnect busz arch.

- PLB: nagyobb képességű
  - viszonylag korlátozott
  - előírások: a buszra csatlakoztatható eszközök száma (kis késleltetés biztosítása)
  - kihasználtsága jó
- átvitel séma:
  - átlapolat átvitel
  - szétválasztott adat és cím
  - az egyes Masterek elindítanak olvasásokat (melyek késleltetéssel zajlanak le)
  - a késleltetés tehát nem biztos, hogy a periféria lassúsága miatt van
  - az írás csatornán viszont nem feltétlenül van forgalom, ezért az írást rögtön el lehet indítani (ha a slave is készen van, nem olvasnak éppen belőle, a slave-ek többsége nem támogatja azt, hogy szimultán írás/olvasás legyen)
  - tehát a PLB buszon hatékony kétirányú adatátvitel valósítható meg

PLB írási ciklus:

- handshakes kapcsolat, lezárás acknowledge segítségével
- az első ack a címezési, a második az adatátviteli fázisra vonatkozik
- van olyan átvitel is, ahol nincs szükséges ack-ra (a periféria nem igényel várakozást)

burst:

- először lezajlik a címezés
- aztán késleltetés
- majd megjelennek az adatok
- a burst végén jelezzük, hogy nem kell tovább burstolni

- tehát vagy fix hosszúságú burst, vagy egy jellel termináljuk (ekkor a periféria befejezi a működést és nem válaszol)

#### PLB felépítés

- szokásos, mesterek, slave-ek
- master oldal
  - mesterektől közvetlenül jutnak be az adatok, címek a buszstruktúrába (az ábrán nincs különösebben bonyolítva)
  - a buszhálózat dönti el a kérések elfogadását
- slave oldal
  - a visszaérkező adatokat egyetlen olvasási buszra tereli (vagy kapcsolatos áramkör)

#### PLB buszhozzáférés vezérlő

- a tervezésnél meg kell mondani, mennyi master lesz
- minél kevesebb master, annál gyorsabb lesz az arbitrációs logika
- slave-ek száma is korlátozott (a terhelés szinten tartása)

#### Prioritás

- csoportos körforgó prioritás
- 4 szint lehetséges, amelyet definiálhatunk az egységekhez
- lehet több master is egy pr. szinten, akkor az arbiter oldja meg, de általában nem szokott nagy lenni a mesterek száma (pl. proc, DMA, nagy sebességű adatforrások, kb ennyi)
- tehát nem kell, hogy bonyolult legyen az arbiter logika

#### PLB buszhíd

- speciális periféria: az eszközvezérlő regiszterekhez való hozzáférést lehet megoldani vele

#### OPB:

- jelentősen kisebb képességű
- nincsenek szétválasztott adatvonalai (egyetlen egy adatvonal)
- a lebonyolítható busztranszferek is kevesebben vannak
- de itt is megvalósítható átlapolatú átvitel, burst is (ez ma már elvárás)
- request és ack egy kombinációs hálózaton vannak visszakötve
- a késleltetést nem feltétlenül lehet alacsony szinten tartani

#### OPB busz:

- adatok egy közös buszon (nincs szétválasztott írás/olvasás)
- előfordulhat egy lokális master
- az arbiter egység végzi ezek hozzáférését

#### Busz olvasás:

- egyszerű átvitel
- külön címezési és adatfázis
- a címezési fázisnak nincs ack periódusa (csak az adatátvitelnek)

#### DCR busz

- ezzel nem foglalkozunk

## OCM

### PowerPC OCM buszok:

- ábra: külső memória illesztése a processzorhoz (szorosan csatolt memóriák)
- miért érdemes: pl. egy IT rutin, ha egy belső utasításmemóriába helyezzük el, akkor ott van és marad is a helyén
- de cache esetén: ha régen volt hozzáférés, lehet hogy már nem tartalmazza a szükséges adatot

-OCM: cím, adat engedélyező vezetékek, ennyi

### OCM busz kialakítása

- belső cache memória
- közel 200-250 Mhz órajelsebességű átvitel belül

### alkalmazás:

- gyors: megszakítás, indulási kód
- adatoknál: olyan adatmennyiség, amely nem fér el a regiszterekben (de gyakran szükség rá)

### proc és buszsebesség viszonya:

- a buszon túlságosan nagy sebességet nem lehet elérni

## LMB

- nagyjából ugyanaz, mint az OCM, de most a MicroBlaze-ről van szó
- szorosan csatolt memória itt is: közvetlen, jó elérhetőségű memória: szétválasztott adat és cím (nem kellene külön ciklusok)

## AMBA AXI:

- jó, hogyha vannak pont-pont kapcsolatok (ezekhez nem kell speciális buszstruktúra)
- csővonalas átvitel
- FSL
- jól kihasználható olyan egységek között, amelyeket a processzor mellé teszünk (pl. speciális feladatok elvégzése hardveresen: kódolás)
- könnyű adatmozgatás
- olyan, mint egy FIFO (32 bit + jelző flag (jelzés, vagy normál adat jelzése))
- egy processzorhoz 8 csatolható
- hasonlóan: a cache memóriához vezető cache link
- viszonylag bonyolult, de jól használható rendszert alkottak meg ilyen módon

- együttműködés az ARM-mal (AMBA), és valószínűleg a MicroBlaze-zel is a jövőben

FSL:

- egyszerű, lebutított csatorna típusú átvitelként tekinthetünk rá

előnyei:

- nem nagyon van hátránya, max annyi a követelmény, hogy mindkét helyre be kell építeni, nyilván :)

.  
.  
.  
.10 tény Izrael félvezető iparáról :)  
.  
.  
.

Socket alapú kommunikáció

- hordozhatóságot biztosítani kéne: erre találták ki a zokni alapú kommunikációt :)

- standard interfész
- a digitális eszközök natív adatátviteli tulajdonságait valósítsuk meg
- legyen egy eszköz, amely ezt tetszőleges buszra fordítja
- tehát továbbra is a megszokott buszokat használjuk, de az a cél, hogy "ne buszspecifikus legyen a tervezés"

Socket alapú SoC rendszer

- pont-pont kapcsolat: az egység kommunikál az adattorral
- egyszerűsített problémát kell megoldanunk

OCP blokkvázlat

- lényegében ugyanazt a struktúrát használjuk, mint korábban
- különbség: azonosítjuk azokat az elemeket, amelyek speciálisan arra szolgálnak, hogy különböző pl. buszstruktúrákat kössenek össze
- egyik oldal: bus target, a másik oldalon nincs jelölve, hogy az milyen busz
- bus wrapper: busz becsomagoló hálózat, a követelményeket fogja össze

OCP jelek:

- 3 típusú jelcsoport van
  1. Adatfolyam
  2. Egyéb, még szükséges jelző flagek
  3. Beépített tesztelés
- fontos a VLSI technológiában (3)
- kívülről, a saját interfészeivel nem tudjuk ezeket az eszközöket tesztelni



- DFT: Design For Test - úgy tervezünk, hogy tesztelhető legyen
- DFM: Design For Manufacturing - úgy tervezünk, hogy gyártható is legyen
- egyszerű hozzáféréseken kívül bonyolultabb átvitelek (pl. burstös, vagy csomagátvitel)
- maga a protokoll már tranzakciókat különböztet meg, nem egyedi átviteleket
- egy művelet: több fázisból

#### OCP átvitel

- kérés, handshake, átvitel
- normál átvitel, részben átlapolt, teljesen átlapolt
- hiányos átvitel (csak egy jelzés) - ez is külön tranzakciónak számít

#### Profilok kialakítása

- különböző alkalmazásokhoz
- könnyebb a tesztkörnyezet kialakítása
- különböző típusú profilok:
  - alapvető: regiszterhozzáférés (ez ugye alapvető)
  - ha van memóriainterfész: blokkos átvitel (tkp. burstös)
  - X-Bus átvitel: csomag alapú átvitel
  - H-Bus: egyik buszról a másikra lehet fordítani az átvitelt
- ezeknek a bonyolult ábráknak nincs jelentősége :)
- .
- .
- .

#### VSIA Virtual Component Interface

- semmi lényeges különbség nincs
- ugyanaz a lényeg: P-P kapcsolat, kérés-válasz alapon szinkronizálva
- minimál, normál, advanced verzió
- egy réteg (ami eddig adapterréteg volt): kezdeményező, fogad
- nem különbözik az OCP-től

#### PVCI: egyszerűbb

- BVCI, AVCI: bonyolultabb, átlapolt
- lehetséges Out of Order jelzés is

#### PVCI protokoll

- szinkron és aszinkron verzió
- ha a valid jel érvényes, akkor az órajel felfutó élénél van mintavétel, ha van ack is
- tehát ez nem aszinkron, de mindegy

## BVCI

- burstös átvitel
- nem megszakítható (ha blokkot elkezdett átvinni, annak végig kell futni)
- a busz egy központi erőforrás: ha sokáig lefoglaljuk, akkor mást kiéheztethetünk
- tehát érdemes okosan megválasztani a csomagméretet
- szokásos átvitel: kezdeményezőtől kérések, erre a target válaszol, megfelelő sorrendben mennek vissza a cuccok

példa: 1. csomag megmondja hány csomag lesz az átvitelben

- ha ez befejeződik, akkor elindulhat egy másik átvitel
- felszabadult a csatorna
- a második átvitel elindulhat, ezután visszatérhet a vezérlés az első tranzakcióhoz
- tehát ha az első tranzakciót nem bontanánk fel részekre, akkor az erőforrások blokkolódhatnának

## AVCI tranzakciók:

- elindítottuk a két tranzakciót
- ha felszabadult a busz, akkor végrehajtjuk
- csak annyi a különbség, hogy itt nem mondják meg, mikor kezdődnek a következő read folyamatok
- a kimenő forgalmat optimalizálja

## Tervezési tippek

- topológiák:
  - két egység összekötése egyszerű
  - buszhoz csatlakozás esetén buszillesztő réteg szükséges

egy megoldás: egy buszillesztő, a másik oldalon pl. három periféria, akkor ezeket egyetlen buszinterfészre kötjük (pl. kisebb regiszterinterfészek összekapcsolása egy nagyobb mezőbe)

## Összegzés:

- áramkörön belül nem foglalkozunk a vezetékek számával
- nem feltétel a multiplexálás
- ha van lehetőség, akkor P-P kapcsolat megvalósítható/megvalósítandó
- de szükség lehet buszstruktúra kialakítására:
  - vagy készítünk egy egyszerűt, vagy választunk egyet
  - ARM AMBA: kereskedelmi forgalomban nem feltétlenül alkalmazható
  - egyébként általában szabadon használhatóak
- egyre erősödni fog a socket alapú interfészek jelentősége

## Rendszerarchitektúrák előadás

2010. március 25.

előző órán: processzorok általános jellemzői

- milyen típusú eszközök fordulnak elő jellemzően a beágyazott eszközökben
  - mindegyiknek vannak előnyei, hátrányai
  - az alkalmazáskor derülnek ki a hátrányok, de a csere nem ajánlott, mert a másikat is megvannak a problémái, amikor akkor derülnek ki
  - ha egy eszközt már ismerünk, annak nagyobb jelentősége van
  - ezért éri meg a cégeknek az egyetemi kereteken belül támogatni az oktatást
  - pl. egyetemünkön az NXP
- 
- megtalálhatóak az Intel beágyazott rendszerekbe tervezett processzorai
  - megtalálhatóak tipikusan beágyazott környezetre tervezett processzorok
  - általános célú vs. jelfeldolgozási processzor
    - kezdetben alapvető különbség: szorzás-összeadás művelet
    - ez már nem jelentős, mert amióta több tranzisztor elfér ugyanakkor helyen, egy sima processzor is rendelkezik ezekkel
    - a belső erőforrás sokszorozása a jellemző: memória blokkok sokszorozása, párhuzamosítása

- jelfeldolgozó processzor: rendelkezik az általános célú processzor képességeivel - operációs rendszer is telepíthető

Párhuzamosíthatóság igénye:

- VLIW: Very Long Instruction Word
- Superscalar
- párhuzamos szálak
- miért van ezekre szükség: sokszor a beágyazott rendszerekben is elvárjuk a párhuzamosíthatóságot

VLIW:

- emlékeztető: mikroprogramozott vezérlő
    - horizontális: ahány vezérlési pont van a hálózatban, annak mindnek van egy bite a memóriában
    - vertikális: megnézi mik azok a vezérlési pontok, amelyek nem egymással kapcsolatban változnak, ezeket összefogja és egy keskenyebb mezőbe, majd kódolással/dekódolással kezeli
    - vigyázni kell a konfliktusokra: inkább 2 utasítást használok, minthogy +1 bitet használjak a memóriában
- VLIW eredete: horizontális mikropr. vez.

példa: Signal 32 - egy mikroprogramozott jelfeldolgozó rendszer

- bitselect processzor (mai SSE utasítások)
  - lásd: 2 db 4 bites processzor összekötése, ebben az esetben 8 db összekötése (shift vonalak, alu, stb.)
  - 196 bites utasításrendszer
- 
- egy ciklusban több utasítást hajt végre
    - van egy viszonylag nagy méretű egységes regiszter tömb
    - ehhez van több feldolgozó egység, és ezek egyszerre több utasítást hajtanak végre (utasítás csomag végrehajtás)
    - ha véletlen a különböző utasításoknak nem azonos a végrehajtási ideje egy adott csomagban, akkor bevárják egymást
    - az utasításcsomag összeállítása fordítási időben történik
    - a hatékonyság a fordító függvénye: ha jó a compiler, akkor jól össze tudja szervezni a utasítások végrehajtását
    - ebből kifolyólag a végrehajtás egyszerű

- fő probléma: viszonylag nagy méretű, sok portos regisztertömb, vagyis egy olyan komponens, amiből egyszerre 8-16 adat olvasható, ezen az egységek műveleteket végeznek, majd a művelet eredménye visszaíróható a regisztertömbbe
- a regisztertömb teszi drágává ezt a struktúrát
- kompromisszum: általában két részre osztják a regisztereket
  - regisztertömb A, regisztertömb B: mindegyik rendelkezik tipikusan azonos műveleti egységekkel
    - a vezérlés azonos, ott nem egyszerűsödött a struktúra
    - a kettő között úgynevezett cluster buszon lehet kommunikálni
- a hardverkölttség csökken
- a fordító feladata bonyolultabb
- összegzésében érdemes ez a megoldást megvalósítani

pl. TEXAS C6X család

- általános mag (VLIW): regiszterek (A,B), négy egységből álló adatfeldolgozó (feladatai: adatfeldolgozás, címszámítás, store/load)
  - nagy teljesítményűek, jól skálázhatóak
- pl. C6472: 6 darab párhuzamos magot tartalmaz (az előző egység hatszor megszorozva)
- ehhez tartoznak még lokális memóriák
  - közös memóriablokkok

tipikus példa: intelligens videó feldolgozás

- nem az első szintű feldolgozásról van szó
- mi számít komoly feladatnak: egy nagy felbontású (pl. 720x500) képnél valós idejű képtömörítés, kódolás
- ezekre a funkciókra önálló hardveres gyorsító egységet tesznek be
- manapság: térfigyelő kamerák - képesek arra, hogy a személyeket azonosítsák

másik példa VLIW struktúrára: Freescale (Motorola) Starcore

- mit tartalmaz egy ilyen mag:
  - nem szimmetrikus cluster: van kifejezetten adatfeldolgozó+adatregiszterek, van címaritmetika+címregiszterek, valamint tartozik hozzá egy vezérlő egység (sequencernek is hívják, mivel ő állítja össze az adatok sorrendjét)

- felépítése: M1 szintű memória: mérete 1.5 kB
  - ezen belül egy belső busz, ahova a StarCore és a memória a QBC-n keresztül kapcsolódik (QBC: ...)
  - továbbá: egy megszakításvezérlő
  - külső kapcsolatok
- az egész egység neve: SC140 mag

MSC8126 processzor: benne 4 db SC140 mag található, amelyek egy komplex buszrendszerrel vannak összekapcsolva

- tehát ez egy komplex SoC rendszer, amely buszrendszert és memóriákat tartalmaz
- a jelfeldolgozó processzorok közül ezek a VLIW technikájú processzorok feszegetik a határokat

További processzortípusok...

Superscalar:

- nem tipikusan beágyazott környezetben használatosak, de megjelennek itt is
- fő tulajdonság: egy időben több utasítás végrehajtását indítják
- minden végrehajtó egység több példányban áll rendelkezésre
- vagyis ezeknek a processzoroknak a tervezése első közelítésben nem igényel semmi rendkívülit
- pl. van egy feldolgozó egység (4 feladat): utasítás elővétel, dekódolás, végrehajtás, visszaírás
- még egy egységet mellé rakunk, ez így működik, és ok (többszörös pipeline)

- de mi van előtte: futási időben a vezérlő dönti el a kiadható utasítások párosítását
- ugyanaz a probléma, mint az előző esetben: az utasítások függősége (egyik utasítás eredményét használja-e a másik)

#### Vektorprocesszorok:

- valóban nem beágyazott környezetbe valók
- ugyanakkor pl. médiaalkalmazásokban (akár hang, akár kép), ahol az adatdinamika kicsi (pl. 8, 16 bit elegendő), ott több azonos feldolgozó egység könnyedén beépíthető

#### Szálszintű párhuzamosíthatóság:

- egy időben több feladat végrehajtása történik
- itt is van több műveletvégző, cél: ezeket minél hatékonyabban leterheljük
- durva szintű szálcapcsolás, finom szintű szálcapcsolás, egybefont (szimultán) szálcapcsolás

pl. négy szál, négy egység

- durva esetben először az A szál végrehajtását csináljuk, amíg van vele feladat
- ha az A szállal már nem tud tovább foglalkozni, akkor elkezdi a B-t, és így tovább
  - finomabb feldolgozás
  - megteheti azt, hogy megpróbálja az egyes szálakból kiválasztani az a feladatot, amiből 4 műveletet tud végezni egyben
- egybefont szálcapcs:
- minden egyes esetben úgy dolgozik, hogy összes műveletvégző elem használva legyen
- ameddig éppen feladat van, teljesen le tudja terhelni a végrehajtó egységet

#### Energiahatékonyság:

- nagy teljesítményű CPU: fogyasztás is megnő
- ha a kihasználtság indokolja, akkor ezt elfogadjuk
- változó kihasználtság mellett kellemetlen, hogy adott esetben a dinamikus fogyasztás csökken, de a statikus fogyasztás marad
- próbáljuk arányaiban javítani, valamint a statikus fogyasztást redukálni
  - ha a proc. részegységei nincsenek használva, akkor azokat kikapcsoljuk
  - megoldás: valamilyen intelligens energiagazdálkodás
  - ennek az igénye természetes, de a megvalósítás nem triviális
  - a működés adott szintű fenntartása elfogadható energiaigény mellett
- egyik megoldás: dinamikus frekvencia- és feszültségváltás
  - általában az alkatrészek CMOS technológiával működnek
  - szivárgási áram miatt: statikus fogyasztás (konstansnak mondható, de feszültségfüggő)
    - dinamikus fogyasztás: valamilyen konstans \* frekvencia \* feszültség négyzete
    - a különböző frekvencia és feszültségtartományok működés során változtathatóak (nem használhatjuk viszont mindig a legkisebb lehetséges feszültséget, mert minden feszültséghez tartozik egy felső működési határfrekvencia)

Megoldás: alapesetben normál tápfeszültség, normál frekvencia

- ha nincs feldolgozási igény: először frekvencia csökkentése, amikor az stabilizálódott, akkor feszültség csökkentése
- az úgynevezett magfeszültséget csökkentjük (az I/O feszültségeken természetesen nem változtathatunk)
- ekkor van a kis fogyasztású állapot: 60 %-nyi teljesítmény, 30 % fogyasztás mellett
- ha lecsökkentjük a feszültséget, akkor nem hibernálódik a rendszer (ez tehát nagyban különbözik a sleep állapottól)
- ha valamilyen esemény történik, ami igényli az eredeti állapot visszaállítását: először feszültség visszaállítása, majd az ehhez a feszültséghez már használható frekvencia visszaállítása

Érdekes elv:

- általában ha digitális rendszereket tervezünk, akkor worst case-re
- jelentős belső tartalékkal dolgozunk
- újdonság: ne veszítsük el az időt, és ha kell, javítás
- minden fázisban megnézzük, hogy az általunk kockázatosan beállított sebesség jól működik-e, és ha nem, akkor korrigálunk
- mintavétel a maximális végrehajtási idő előtt: ha a korai eredmény megegyezik a helyes eredménnyel (amit a maximális idő eltelte után mintavételezünk), akkor minden rendben van
- a rendszert megpróbáljuk egy határig felhúzni, és e határ körül mozgunk
- kihozzuk a maximumot a rendszerből azon az áron, hogy egyfolytában ellenőrizzük a rendszert
- az ARM az újabb generációs processzorokban használni fogja ezeket a technikákat
- mi a probléma: a memóriák fogyasztási igénye fog sokat nőni, "hiába" optimalizáljuk a processzor fogyasztását (habár vannak ígéretesnek tűnő megoldások)

következőkben:

- FPGA: hatékonyabbak, kisebb órajelfrekvencián működtethetőek
- párhuzamosíthatóság

## Rendszerarchitektúrák előadás

2010. március 29.

### Analog Devices

#### DSP processzorok

- ADS21xx
- Blackfin ADSP-BF5xx - fixpontos, általános célú
- SHARC - fix/lebegőpontos, nagy teljesítményű

### Blackfin processzor

- próbálja a jelfeldolgozási igényeket előnybe helyezni (Mac, párhuzamos adatelérés, stb.)
- jó memóriastruktúrával rendelkezik: részben ellentmond azzal, amit a DSP-ről tudunk (külön adat- és címmemória)
- struktúra: hasonlít egy normál processzorra
- dinamikus teljesítményszabályozás: az igényeknek megfelelően a működési feszültséget, órajelfrekvenciát csökkenti
- elsősorban DSP jellegű, de mindazokat a funkciókat, amiket a mikrokontrollerek tudnak, tudja

MCU: vezérlés, kommunikáció, real-time jellegű működés

- ezek ötvözve a jelfeldolgozási képességgel: ASIC jellegű

Elsődleges szempont: jelfeldolgozás

DSP:

- valós idejű kimenet/bemenet
- ha valamennyi flexibilitás megengedhető, akkor több mintát megvárhatunk
- fix pontos számábrázolás
- sok esetben az algoritmusok nem igényelnek lebegőpontos számítási pontosságot

Controller:

- stack, képes pointerekkel dolgozni (a jelfeldolgozó processzorokra nem jellemző)
- cache
- byte címzés (a jelfeldolgozó processzorok egyébként olyan szélességben címzik a memóriát, amilyen a memória természetes szószélessége)

Blackfin Core:

- maga a core közvetlenül a cache memóriához fér hozzá (adat és utasítás cache)
- a külvilághoz a DMA vezérlőkön keresztül
- felső sor: a mikrovezérlőkhöz tartozó perifériák
- JTAG emulator
- adatfeldolgozó egység, címaritmetikai egység, sequencer: ezek a kontrollerek a jelfeldolgozási folyamatokat támogatják (loop)

Aritmetikai egység:

- 2 MAC egység (16 bitesek)
  - 2 db 16 bites szám szorzása, 32 bites eredmény
  - 8 bit kiterjesztés: 256 bites szűrő megvalósítása biztonsággal
- 2 db szorzó
- új tulajdonság (a DSP-khez képest): load/store architektúra: műveletet csak regiszterben lévő adaton képes végezni
- regiszterkészlet: 16 vagy 32 bitesként használható
  - a műveletek többsége páronként használja a regisztereket

Barrel Shifter:

- tetszőleges számú eltolást lehet egy órajelciklus alatt elvégezni

- vannak továbbá olyan műveletek, amelyek hardverrel egyszerűek, de a processzornak nehézkes

- bitszintű XOR (CRC-hez kell)
- véletlen generátor

Cím kiszámító egység:

- általános pointer regiszterek: adatstruktúrákhoz könnyen lehet indexelten hozzáférni
- cirkuláris buffer, bit-reverse címzés, stb.
- stack pointer

Címzés:

- reverse carry add: egy összeadásnál megfordítjuk a carry terjedését (ez segít megvalósítani a bit-reverse címzést)

Sequencer

- az utasítás elővétel és végrehajtás a lehető leghatékonyabb legyen
- a program memóriaigénye minél kisebb legyen
- az utasítások tipikusan 16 bitesek (ritkábban 32 bitesek (közvetlen ugrás), vagy vannak 64 bitesek is (bonyolult funkciók))

Program Flow

- linear
- loop: magában a hurokban nem kell foglalkozni a az adminisztrációval (ez előnyös DSP-nél)
  - a sequencer hardveresen elvégzi a looppal kapcsolatos adminisztrációt (pl. ciklusváltozó dekrementációja)

Sequencer Registers:

Instruction Pipeline:

- utasítás elérés, adatelérés, végrehajtás, visszaírás

Event Handling:

- ha történik valamilyen hiba/esemény, akkor a pipeline-t ki kell üríteni pl. interrupt
- interrupt források: nincs jelentősége

Instruction Packing

- a proc. mindig egy 64 bites szót olvas be, és abban megkeresi az utasítás határát

A compiler több stratégiával dolgozhat a ciklusok végrehajtását illetően

Ciklus előkészítés

Ciklus mag

Ciklus lezárás

Memory Hierarchy

L1 cache: lehet cache, ROM is

- utasítás, vagy adat

L2 memória: vagy van, vagy nincs

L3 memória: inkább külső mem.

- ez a struktúra az általános processzorokra hasonlít

Configurable Memory

- a belső memóriát normál memóriaként, vagy cache-ként használjuk (alkalmazásfüggő)
- L1

Cache and Memory Management

- LRU: a legrégebben használt adatot dobja el, ha valamit nem talál a cacheben

DMA:

- párhuzamosan tud működni a core-ral, ha egyszer fel lett programozva
- esetleg közben interruptokat kér



#### Power Management Options

- változtatható órajelsebesség, ennek függvényében a működési feszültséget beállíthatjuk

ábra: ha csak a sebességet csökkentenénk, akkor nem csökkenne jelentősen a fogyasztás, ezért kell és van lehetőség a feszültség csökkentésére

#### Hardware Debug Support

- a belső erőforrások folyamatos megfigyelhetősége
- az alkalmazás leállíthatósága

#### Optimizing for Execution Speed:

- az ábrán látszik, hogy mi az ami hatékonyabbá teszi: optimalizálás, cache, manuális javítás

#### Concept and Tools

##### C előnyei:

- gyorsabb fejlesztés
- könnyű ellenőrzés
- azért nem árt ismerni, hogy milyen hardverre írunk

#### Optimalizált és nem optimalizált kód:

- optimalizált, jelentősen rövidebb (a rövid kód mellett nem lettek feltüntetve az egyéb folyamatok)

#### VDSP++ Statistical Profiler

- megmutatja, hogy az idő nagy részét hol töltjük: ez jó kiindulópont az optimalizáláshoz

#### Tuning...

- nem használunk lebegőpontos műveleteket
- a gyártó fejlesztett egy gyors lebegőpontos könyvtárat (ennyit a max támogatás erre)
- inkább integer, vagy törtszámábrázolás az előnyös

#### Effects of Vectorisation...

- példa: egy művelet 3 utasításból áll
- ha egy kicsit átszervezzük: egy időben két adatot olvasunk be, elvégezzük rajtuk a műveletet
- ugyanúgy 3 utasítás, csak két adaton végeztük el

## Rendszerarchitektúrák előadás

2010. április 1.

Vendégelőadó: Lőre Géza

- az ARM-nál írta a diplomamunkáját

ARM Based Systems

- tavalyi évben 4 milliárd ARM alapú chipet adtak el világszerte

ARM Ltd.

1990-ben alapult

- 12 emberrel indult egy pajtában :)

ARM: kezdetben Acorn Risc Machine volt a jelentése (manapság Advanced)

- chipeket nem gyártanak le, csak a digitális terveket szolgáltatja (licenzelik a technológiát)

- amikor a dia készült, akkor több, mint 550 partnere volt az ARM-nak

Termékek:

Processzorok:

- Secure Core: biztonságos rendszerekben (egy elszeparált csoport fejleszti)

Rendszerszintű modulok:

- AMBA buszspecifikációk, és az e köré épülő buszmegoldások

- System Controllers: DMA vezérlők és hasonló

Multimédia részleg:

- videó kodekek, engine-ek

Fizikai IP

- ASIC implementációhoz szükséges

Szoftvermodulok, debug termékek

"Soha nem vagy messzebb, mint 2 méterre egy ARM-tól"

Architektúra verziók

ARMv4: első igazán elterjedt

- mai napig a bevétel 50%-a ebből van

ARMv5

- kiegészítések

ARMv6

- ARM11 modellek, sok mobilban ez van

ARMv7

- vezető processzorcsalád

- Cortex család (itt már nem vízszintes az építkezés)

Cortex M: viszonylag kis teljesítményű (mikrokontrollerek)

Cortex A: nagyobb teljesítményű

ARM Architektúra

- 32 bites Load/Store Architektúra

- Thumb utasításkészlet

- az ARM legtöbb IP-je erősen konfigurálható (mivel gyakorlatilag verilog kódot adnak el)

- pl. Jazelle: hardver Java bájtkód végrehajtást tesz lehetővé

- NEON: kisebb vektorprocesszor

- TrustZone: biztonsági kiegészítés: olcsó formája a virtualizációnak (a biztonságos és nem biztonságos világ között egy jól specifikált interfészen lehet csak kommunikálni)

ARM utasításkészlet:

- nagy teljesítményű utasításkészlet
  - minden utasítás 32 bites, feltételesen végrehajtható az aktuális flagektól függően

THUMB:

- újrakódolt részhalmaza az ARM utasításkészletnek
- 16 bit: megnő a kódsűrűség

THUMB-2:

- az ma javasolt, a THUMB kiegészítése
- 16 és 32 bites utasítások, a processzor teljes funkcionalitása rendelkezésre áll

interworking:

- az egyiket megírjuk az egyik utasításkészlettel, a másikat a másikkal
- a legtöbb processzorban csak a branch és jump hajtható végre feltételesen
- barrel shifter

Processzor módok

- a klasszikusokban 7 mód van

User: a legtöbb alkalmazás fut (nem kivételes: az utasításkészlet egy kis részét nem lehet végrehajtani)

FIQ

IRQ

Supervisor: operációs kernel kód egy része futhat ebben a módban

Abort: pl. egy olyan memóriaterület megcímzése, ami nem létezik (hibakezelés lehetséges)

Undef: pl. lebegőpontos utasítások kezelése abban az esetben, ha egy másik módban erre nem volt lehetőség

System: lényegében olyan mint a User mód

Register Set

- 16 db általános célú: nem teljesen igaz
- 13 db általános
- R13: stack pointer
- R14: link reg. (szubrutin visszatérési cím)
- R15: PC
- CPSR: Current Program Status Register

- ha jön egy magas prioritású megszakítás: R8-R14ig ha valaki hivatkozik ezekre a regiszterekre, akkor ezeket a regiszterek nem az eredeti helyükre képzik le fizikailag

Exception Handling

ARM state: ARM utasításkészletbe váltás

Microarchitecture

ARM7TDMI

- van egy saját, egyedi memóriainterfésze

Cortex M3 Processor

- DAP: Debug Access Port (JTAG hozzáférés)
- Memory Protection Unit: kritikus kódot/adatot lehet megvédeni attól, hogy user taskok hozzányúljanak
  - ez egy egyszerűsített MMU
- Data watchpoint

Cortex M3 adatút:

- Harvard architektúra

Pipeline

- fetch: egy feltételes utasítás minden lehetséges kimenetelét fel tudja hozni

Cortex-A9

- konfigurálható: max. 4 processzormag
- FPU
- a 4 processzor egy koherens clusterben tud együtt működni
  - a processzorok cache-eiben konzisztens adat található
- 2 db 64 bites AXI interfész

Pipeline

Out-of-Order: ha egy utasításhoz nem áll készen valami, akkor egy következő utasítás végrehajtható, majd visszairás következik annak érdekében, hogy konzisztens legyen

AMBA

AXI 3

- relatív egyszerű pont-pont kapcsolat

Channels

- minden csatornán van egy ID jel

handshake: átvitel csak akkor van, ha mindkét jel 1 értékű (valid, ready)

AXI tranzakciók

- főlán :)

## Rendszerarchitektúrák előadás

2010. április 8.

### Szenzorhálózatok

- miért foglalkozunk velük: egy egészen újfajta alkalmazási területet jelentenek a beágyazott világban
- környezeti intelligencia: próbáljunk meg a környezettel interaktív kapcsolatba lépni, onnan információkat gyűjteni
  - lokális méretű (egészen közeli)
  - nagyobb körben (pl. városi közlekedési hálózat is kapcsolódhat ide, vagy egy épület teljes fűtésrendszere)
- Fizikai környezet: azt szeretnénk, ha a fizikai környezet és a beágyazott rendszerek egymással lényegében teljesen együttműködve egyfajta komplex megoldásként tudnának működni
  - beágyazott rendszer feladatai:
    - mérés, adatgyűjtés (információgyűjtés)
    - beavatkozás - vezérlési funkciók
- általában vezeték nélküliek (szenzorhálózatok), de nem mindig
- általában jellemző, hogy nem csak az adatgyűjtés, hanem sok esetben a feldolgozás is a hálózaton belül történik
  - nem minden esetben teljesen egyértelmű: előfordul, hogy a hálózatot csak adatgyűjtésre használjuk, de ha nem csak arra, akkor úgynevezett elosztott/disztributív számítási modellről van szó
- a hálózatok felépítése magában hordozza azt, hogy az egyes node-ok milyen képességekkel rendelkeznek
  - sokszor minimális képességek, példák: számítási, kommunikációs, működési képességek

### Alkalmazási példák

- környezeti megfigyelések:
    - zajszintmérés
  - fizikai paraméterek mérése
    - hőmérséklet
    - páratartalom
    - nyomás
    - fényerő, intenzitás
  - szeizmológiai mérés: katasztrófa utáni utó rengések mérése
  - intelligens ház
    - energiagazdálkodás (hőmérséklet, fűtésszabályozás, szellőzés, gépészeti berendezések)
  - gépek (vagy szerkezetek) működésének ellenőrzése
    - sokszor lehet mérések alapján a gépek állapotára utalni
- hidak: ultrahangszenzorok érzékelik a hidakban a repedéseket
- Precíziós mezőgazdaság
    - igény szerinti öntözés
    - üvegház esetén teljes klímaszabályozás
  - Egészségügy, beteggondozás
    - vérnyomásprobléma, cukorbetegség
    - hordozható monitorozó berendezés biztosítja, hogy az ő riasztó adatait ellenőrizni tudják
  - Logisztika
    - intelligens azonosítók, "tag"-ek
    - a termék teljes útja követhető
  - Kereskedelem
    - rádiós árcédula, vagy polccímke

#### Használati módok:

1. Esemény érzékelés
  - esemény osztályozása is adott esetben
  - pl. betörés, adott fizikai folyamatban megnőtt valamilyen fizikai paraméter
2. Periodikus mérések
  - elemzés, továbbítás
  - egyenletes, vagy összegezve
3. Függvény érzékelés, közelítés
  - sok érzékelőt alkalmazunk, és a jelekből hely- vagy időfüggő adatbázist állítunk elő
  - pl. gyorsforgalmi mérés, a zajárnyékolás észlelése
4. Nyomkövetés
  - vagy mozgó szenzor, vagy kamerás rendszer (több kamera működik együtt, valamelyik azonosít egy objektumot, a többi követi a megadott paraméterek alapján, ha az első látóköréből kikerül)

#### Szenzorhálózatok jellemzői

- szolgáltatás színvonala
  - azt várjuk, hogy kis fogyasztású komponensekből álljon, ugyanakkor elvárjuk, hogy valamilyen jelentéssel bíró információ jöjjön vissza
  - válaszokat akarunk, nem számokat
- szolgáltatás minősége
  - mi az amit elvárhatunk:
    - pl. milyen sávszélességgel fog adatokat szállítani (hiába van több eszköz, a sávszélesség nem lesz nagyobb)
    - késleltetés: ha nem tudjuk bizonyos idő alatt kiolvasni az információkat, akkor nem ér semmit
    - biztosítani kell, hogy valamilyen korlátos idő alatt egy minimális adatmennyiség megjelenjen
- hibatűrés
  - elektronikai rendszer: felléphetnek alkatrészhibák
  - közvetlen környezeti hatások
  - energiaforrás problémák
- élettartam
  - egy hálózat meddig mondható használhatónak?
  - redundanciával csökkentjük a kellemetlen hatásokat
  - mi lesz, ha 10%, vagy mondjuk 50%-a meghibásodik: nem tudjuk hol a határ
  - használható, amíg egységes a hálózat
    - vannak node-jaink egy adott minta szerint, ez egy teljes hálózatot jelent (mindegyik eszköznek van egy hatósugara)
    - ha valamilyen eszköz meghibásodik, akkor a mellette lévőekkel megszűnik a kapcsolat és a hálózat két részhálózatra szakad
- skálázhatóság
  - sok elemet telepítettünk, igény szerinti aktivitás (csökkenthető)
  - mérési sűrűséget időben, térben csökkenthetjük
- programozhatóság
- karbantarthatóság

#### Technikai jellemzők

- Vezeték nélküli üzemmód
- energiatakarékos működés

#### 1. Többlépéses (multi-hop) kommunikáció

- fontos az adat és csomagolás aránya: hiszen minden kommunikációnál fel kell építeni a kapcsolatot, küldeni/fogadni a csomagot, majd lebontani a kapcsolatot
- nagy a redundancia
- energiahatékonyság elemzés

## 2. Energiaellátás

- saját energiaforrás, vagy van lehetőség energiagyűjtésre

## 3. Automatikus felkonfiguráció

- kényelmes
- új egyed a hálózatban
- lokalizáció: fontos téma, nincs igazán jó megoldás rá
  - mi kell hozzá: jeladó, szomszédoknál vevő (pl. rádiós jelzésből kideríteni, hogy hol van, ez nem egyértelmű)

## 4. Együttműködés

- önmagában nem sokra képes, csak ha másokkal is tud kommunikálni

## 5. Kommunikáció

- nem a szomszéd azonosítója a lényeg, hanem az, hogy az adatok eljussanak

Működés alapvetően lokális:

- néhányszor 10 m
- kilométeres tartományt is át lehet vele vinni

tápegység:

- elem v. akkumulátor + stabilizátor
- jól megtervezett power management
- mikrovezérlő
- nem túl nagy memória
- általános célú I/O
- AD konverter
- szenzorok

- következő alkalommal megnézzük, hogy az előbb említett egységek mit tartalmaznak, milyen energiaellátási lehetőségek vannak

## Rendszerarchitektúrák előadás

2010. április 12. hétfő

Vezeték nélküli szenzorhálózatok

előző órán: általános felépítést felrajzoltuk

- power supply
  - communication (radio)
  - processing unit
  - szenzorok
- ezek a főbb modulok, mindegyik mellett van egy szoftveres/firmware-es rész
- power management
  - hálózati protokoll
  - teljes alkalmazás + működtető operációs rendszer
  - szűrés, elő feldolgozás

Processing unit: elsősorban CPU-ban gondolkodunk

- CPU-k széles skálája képzelhető el
- pl. 4 bites CPU (EM6603): néhány uA áram működéskor, maximum 32768 Hz órajelfrekvencia
- AVR, PIC, 8051
  - legkényelmesebb: AVR
  - PIC: energiahatékonyság nem annyira jó, de olcsó
- ARM, PowerPC
- mindegyiknél belső memóriák (adat, program)
- programmemória mérete lehet kritikus, adatmemóriából ritkábban van szükség nagyobbra
- perifériák: soros kommunikációk jellemzőek

Radio

- ISM sávú átvitel, valamilyen egyszerű modulációval: ASK, FSK, OOK - néhányszor 10 kbit/s
- ISM sáv felső határán (2.5 Ghz): Wifi, Bluetooth (1Mbit/s), ZigBee (néhányszor 100 kbit/s)

Használati mód

Rx-Tx energiaviszonyok

- adás: energiaigényes
  - hatótávolság-függő
- vétel: gyakoribb mint az adás, úgyszintén energiaigényes
- érdemes olyan protokollt használni, ahol a vételi állapotnak jelentősen kisebb a kitöltési tényezője
- rövid távolságnál gyakran az Rx teljesítményigény közel ugyanakkora, mint a Tx teljesítményigény

Teljesítményforrás:

Energia

- alkáli elemek (néhány ezer mAh, de csak egyszer használatos)
- akkumulátorok (majdnem akkora kapacitás, mint az alkáli elemeknél, általában inkább a fele; néhány száz újratöltés): jellemzően Li típusúak (előny: fél töltöttségből tölthetőek, hátrány: környezetszennyező)
  - esetleg szuperkapacitás (kis fogyasztás)
- legkritikusabb tervezési paraméter
  - nincs minőségi áttörés
  - néha lehet misztikus új energiaforrásokról hallani :)
- energiamenedzsment módszerrel próbáljuk ezt kiterjeszteni, amennyire csak lehet
- extra energia gyűjtés: napenergia (még a leghasználhatóbb)

WINS:

- mintha egy mitmót lenne, csak van még nagyon robusztus doboza



- ARM-ok intel által felturbózott verziója van benne
- adatfeldolgozás is történhet az eszközben
- operációs rendszer fut rajta
- kicsit bonyolultabb az átlagos mote-oknál

#### WINS NG 2.0

- SH4 processzor, DSP-vel
- linux 2.4 kernel
- nem a legjobb választás energia hatékony eszköz tervezésére
- dupla rádiós interfész
- képfeldolgozó
- tehát: komoly beágyazott rendszer, nem egy egyszerű szenzor node

#### Berkeley Motes

- másik véglet: minél kisebb legyen
- hőmérséklet, fény, páratartalom mérése
- kis energiafogyasztás
- az interfész sem kínál komoly képességeket
- ezt is kényelmesen lehet használni: TinyOS szoftverkörnyezet

TinyOS: minimálisra le van csupaszítva

- os-nek van hívva, de nem sok szolgáltatása van
- programhelyet viszont nem foglal sokat
- pl. rádiós interfész: 1.5 KB
- szenzorokhoz tartozó driverek: néhány 10 bájtnyi méret
- általános célú szolgáltatás: fél-negyed KB
- adattartomány sem túl nagy
- a felhasználónak meg van az a lehetősége, hogy tudja használni anélkül, hogy komolyan foglalkoznia kéne egyes kérdésekkel (pl. rádió kezelése)

#### UCLA iBadge

- dual processzoros egység
  - a) AVR: rendszervezérlő
    - lényegében az előbb említett funkciókat ellátja (rádió, csomagok összeállítása)
  - b) jelfeldolgozó proc.: komolyabb feladatokra
    - pl. hangelemzés
- szoftver: Sylph Middleware
- rétegezett struktúra
  - a felhasználónak nem kell az alacsony szintű, hardverspecifikus dolgokkal foglalkoznia

#### UCLA Medusa

- itt is szétválasztott funkciók
- alap funkciók: ATmega
- egyébként: ARM processzor

#### BWRC's PicoNode TripWire Sensor Node

- energiagyűjtéshez egy alkalmazási példa
- látványos, de valódi energiagyűjtésre nem alkalmas
- belső forrás: vegyi energia eltárolva

#### quick-and-dirty

- rövid ideig bírja elemmel

#### Sensor Node Energy Roadmap

- csökken a rendszer által felhasznált energia
- ideális: 0 energia, de 50-10 mW alá nem sikerült még csökkenteni
- kérdés: mi fogyasztja az energiát?
- hova fókuszáljunk?
- különböző mérőszámok vannak

#### Processing:

- Atmega és ARM példaként: processzorok különböző fogyasztása
- nem az alacsony fogyasztás a lényeg, hanem az energiahatékonyság
- tehát: egy adott feladatra mennyi energiát fogyasztunk el
- hiába fogyaszt kevesebbet az alacsonyabb teljesítmény, ha 5-ször annyi ideig fut
- pl. IBM 405: valódi processzor cache-sel meg mindennel, mégsem fogyaszt pl. többet a primitívebb processzoroknál

#### Radio

- mennyire hatékony a kommunikáció
  - valódi környezetben mások is vannak az éterben
  - minden üzenetet vagy 5-ször el kell küldeni
- sok esetben a mote-oknál csak a rádiós egységet nézzük vizsgálatkor
  - ahhoz, hogy együttműködjenek, ahhoz hálózati protokoll kell (ez pedig a processzoron fut)
- problémás: nem aktív állapot

#### Computation, Communication

- egy egyszerű feladatnál a rádió viszi el az energiát
- ha egy komolyabb feladat van (videófeldolgozás), még nem csökken az rádió energiaigénye, még így is legalább 1/3 aránnyal részt vesz az energiafogyasztásban

#### Sensing

- nem az érzékelés az energiaigényes, hanem az a kérdés, hogy mit csinálunk a jellel
- képi/videó jelek feldolgozása: nagy energiaigény

#### Actuation

- előfordul, hogy a sensor node-oknak még valamit csinálni kell (ez a legrosszabb fogyasztás szempontjából)

#### Some Observations

- hogyan tudunk megtakarítani energiát?
- a rádiós kommunikációt is ennek a stratégiának rendeljük alá

#### Processor Energy Management

- a teljes szoftverünket úgy kell elkészíteni
- teljesítményfigyelő viselkedés:
  - a legalsó rétegtől kezdve ezt a szemléletet kell alkalmazni (egészen az alkalmazás szintig)
- processzoroknál tudjuk csökkenteni a működési frekvenciát, feszültséget (esetleg dinamikusan)
- rádiós energiaigény optimális kihasználása:
  - üzenetváltások vizsgálata
  - nem fogunk új rádiós blokkot tervezni
  - igyekszünk a megfelelőt kiválasztani, maximum az antennacsatlakozó helyéről gondolkodunk

#### Példák

- nagyobb távolságnál az elektronika fogyasztása elhanyagolható
- adó/vevő elektronika

kommunikációs protokoll:

- van egy adott bitsebesség
- ha hosszú ideig adunk, akkor lineárisan növekvő fogyasztást jelent
- két görbét összevetve optimumot adunk
  - kis teljesítmény: inkább az elektronika fogyasztása dominál

Power breakdowns and trends

- rádiós vétel analóg elektronikával
- lehetőleg digitálisan csináljuk, így az AD-DA átalakító kérdése marad
- hogyan lehet sávkorlátozott nagyfrekvenciás jeleket feldolgozni kisebb sebességű AD-DA átalakítókkal

Alvási/aktív idő megosztása

- ...
- egyik lehetőség: gyorsan átküldjük az üzenetet és alszunk
- vagy: nagyobb távolságról - kiszámoljuk hogy kéne hej
- eszközök feléledése is időbe telik
- figyelni kell arra, hogy sokszor annyit veszünk a feléledésen, mint ha ébren hagynánk a rendszert
- másik fontos szempont: kikapcsolhatjuk a rádiót, de akkor "halottak vagyunk"
  - néha bele kell hallgatni az éterbe
  - a főmodul dönti el, hogy csinálni kell-e valamit az üzenettel (az egész rendszert fel kell ébreszteni)
  - inkább a rádiós részbe építsünk be valamilyen vezérlést: döntse el, hogy nekünk szól-e a csomag
  - ha nem, akkor a főrendszer felélesztése nélkül dolgozzon

Rádiós alrendszer

- kettős felépítés
  - elsődleges kommunikációs forrás (ad és vesz)
  - megfigyelő vevő: alacsony frekvenciás, alacsony sebességű
    - csak az ébresztési funkciót valósítja meg
    - tehát ez lehet egy gagyi minőségű rádió, jelzéseket küld ("he, veled szeretne valaki kommunikálni")

Putting it all together

- teljesítményhatékonyság
- de nem az a legfontosabb szempont, hogy mindig az energiával takarékoskodjunk, hanem hogy a funkciók rendesen működjenek

Nyomkövető rendszer:

- egyes node-ok: érzékelésre képesek
- intelligens típusú adattovábbítás
- ez nem teljesen megvalósítható: nem irányított antennákról van szó
- az egységek általában nincsenek azonosítóval ellátva
- szép elképzelés: de nem így megy a valóságos vezeték nélküli rendszerekben

Tools

- egyszerűbb szimulátorokkal elvégezni a vizsgálatot, mint valós rendszereken

## Rendszerarchitektúrák előadás

2010. április 15. csütörtök

### FPGA technológiák ismertetése

Gajski-Kuhn

- digitális rendszerek tervezésének különböző aspektusai
    - a specifikáció a viselkedési tartományban fogalmazódik meg
  - a diagram inkább félvezetőkre vonatkozik (alsó oszlop)
- a mi tartományunk: a felső tartomány (a másik az technológia kérdés: EET)

Viselkedési leírás szintjei

- RTL szinten való gondolkodást szeretjük: habár tudunk logikai áramkörökkel tervezni, de azt szeretjük, ha ezekkel a problémákkal nekünk nem kell foglalkozni
- a logikai szintet néha felülről súroljuk
- RTL szinten tudunk jobban foglalkozni a fontos problémákkal

Rendszerspecifikáció

- a tervezési folyamat szigorúan iteratív, ciklikus
- nem jellemző, hogy a tervből rögtön a végleges megvalósítás jön létre
- a valódi kialakítás újabb kérdéseket vet fel, újabb ellenőrzés szükséges

Hierarchikus tervezési módszerek

- felülről-lefelé haladás
  - rendszerszintű problémákkal foglalkozunk először (funkciók)
  - ilyenkor nem látjuk előre, hogy rendelkezésünkre állnak-e a megfelelő eszközök, komponensek
- alulról-felfelé
  - a komponensekből indulunk ki
  - úgy alakítjuk, hogy az adott specifikáció által meghatározott eszköz legyen a végeredmény

Tervezési szintek

- a hierarchikus szintek szétválaszthatók
- gyakran egyedi tervezést igényelnek
  - pl. ahol gyors jelterjedés szükséges, ott másféle tranzisztorokat használunk, vagy egyéb trükköket alkalmazunk
- ilyen szinten már nem foglalkozunk ezekkel a problémákkal

FPGA:

- az eszköz által biztosított hardverrel dolgozunk
- felülről-lefelé: ezt szeretjük, de kénytelenek vagyunk az eszköz adatlapját is olvasgatni
- blokkok kidolgozásánál az elsődleges paraméterekre koncentrálunk, a többit a rendszer kidolgozza (de nekünk ellenőrizni kell később)
- pl. összeadó: nem bitenkénti összeadást adunk meg, mert a rendszer nem fogja tudni, hogy mit kell csinálni
  - magasabb szinten adjuk meg, a többit a rendszerre bízunk

Technológiai áttekintés

- egy korszerű mikrovezérlő szinte mindent tud (kivéve például power management)
- digitális eszközök típusai:
  - ASIC: jelentősége csökken
    - előállítási költsége nagy
  - FPGA: nagy előnye van
    - a megtervezés egyszerű, a struktúra egyszerű (ugyanaz a mintázat ismétlődik)
    - a tömeggyártás miatt elérhetőbb árú
  - programozható logikák (PLD)

- egyszerű PLD: cél az, hogy egyszerűen lehessen flexibilis logikai tervezést megvalósítani (logikai függvények egyszerű megadása)
  - architektúra programozás is felmerül később (GAL)
  - később született meg a PLD
  - egy probléma: a középső busz (korlátozott méretű)
  - a dián egy makrocella látható
- CPLD: központi huzalozási részleg viszi el az egész felületet (a logikai eszközök éppenhogycsak elférnek)
  - költség: Si réteg, tokozás teszi ki
  - hagyományos CPLD: 250 celláig mennek el
  - de: egyszerű tervezés, jó késleltetési adatok
  - általában: kicsi a statikus áram (CMOS-okra jellemző)
  - belső órajel terjesztés: órajel leosztása, majd mindkét él használata (a kapacitások okozta hatás csökkentése)
    - bemenetek: automatikus standby (ha nincs változás, marad ebben az állapotban)

- MPGA: maszk programozható áramkörök
  - a félvezető eszközök fejlődése lehetővé teszi, hogy több funkció beépíthető legyen, mint amennyit az felhasználó használ
  - sikeresség: prototípus tervezése nagyon könnyű
- FPGA: XC2064
  - a maga idejében komoly lehetőség volt
  - 50 Mhz toggle rate (ez csalóka, ez annak a sebessége, hogy egy flip-flopot visszacsatoltak egy inverterrel, és az kapcsolgatott)
  - valójában inkább 10 Mhz
- "fabless company idea": a cég csak megtervezi az áramkört, és keres olyan céget, aki a megfelelő eszközökkel rendelkezik a gyártáshoz
- manapság saját gyár: intel, ibm, amd...

#### FPGA technológiák

- nyers alkatrész bizonyos tulajdonságokkal, mi konfigurálhatjuk
- egyszer programozható is van: ez sokszor előnyös
  - ha a terv stabil
  - nem tartalmazza a járulékos áramköröket, amelyek például az átkonfigurálásra vannak (gyorsabb lesz a sebesség)
  - de persze fejlesztésre nem alkalmas
- többször újrakonfigurálható
- felhasználó általi programozhatóság
  - induláskor automatikusan
  - dinamikus átkonfigurálhatóság

#### OTP FPGA

- egyszer programozható
- átégethető biztosítékok: eredetileg szigetelő réteg, megfelelő feszültséggel vezetővé égethető
- jó alkatrész sűrűséget lehet elérni
- mikroszkópos kép: a roncsolás jól látható

#### Flash FPGA

- sajnos azért, hogy a lebegőelektródás kapcsolót megvalósítsuk, kisebb lesz a sűrűség

#### SRAM FPGA

- a konfigurációs bitek mindegyike egy statikus 6 tranzisztoros cellában van
- hátrány: nagy energiájú részecske eltalálja az egyik tranzisztort, az egész átbillenhet (ez veszélyes is lehet)
- de: erős tranzisztorokat tesznek oda :)

Értékelés technológia alapján (másfél éves slide-ok)

- az SRAM FPGA-k mindig az élvonalban vannak
  - nem bonyolult, de szabályos
  - nem kell hozzá új tesztchipet tervezni
- nagy lábszámokkal dolgozunk
- önmagában az eszköz sebessége nem túl nagy, de a használati mód miatt ezen a sebességen több párhuzamos egység tud működni
- nagy a nyugalmi áram
- bekapcsolási áramlökés

#### QuickLogic CSSP

- komolyabb gyártóknak tervez szükséges funkcionalitással rendelkező szabványos eszközt
- ezzel gyors tervezési időt tud elérni

#### FPGA felépítése

- a felhasználó sok logikai kaput lát
- a külvilághoz I/O blokkokon keresztül kapcsolódnak
- a programozható kapcsolatok döntenek el az áramkör minőségét, sebességét
- a késleltetés nagy része a huzalozás miatt van
- a LUT-ok méretét megnövelték: inkább vesszen el funkcionalitás, mint felesleges huzalozás legyen

- a konfiguráció memória egyetlen nagy shift-regiszter, amelybe bitenként betöltjük a konfigurációt
- ahhoz, hogy elfogadható sebességű legyen a konfiguráció, párhuzamos lehetőségek is kellenek

#### Általános tulajdonságok

- találjunk egy univerzális elemet
- NAND, MUX: tetszőleges logikai függvényt megvalósíthatunk
- LUT4: lényegében egy memória (ha a logikai fgv. igazságtáblázatát betöltjük, akkor úgy fog működni)
- 1T: tranzisztorokkal dolgozunk

szimmetrikus: ...

csatorna típusú: ASIC áramkörökből származik

hierarchikus PLD: - egy olyan CPLD, amelyben nem középen van a nagy huzalozás, hanem a logikát lehetőleg lokálisnak gondoljuk

#### Logikai cella példák

##### 1T finomságú konfigurálás

- nem volt versenyképes megoldás
- egy tranzisztorhoz kapcsolódik 20-30 kiégethető szál

##### NAND2, Latch

- ez is kihalt

##### MUX

- ma is használják
- a MUX-t transzfer kapukkal hozza létre (soros FET-ek)

##### SRAM alapú LUT

- a logikát memória valósítja meg
- pl. 2 bemenetű LUT (a,b): és kapu megvalósítható
- az áramkör generálása egyszerű
- szintézer: van egy tetszőleges bonyolultságú áramkör, mindegy mit csinál, ha négy bemenete van, akkor az összes funkció belefér a memóriába (a bemenetszám mindent meghatároz)

- de pl. mux-os kialakításnál: bonyolultabb

Logikai cella problémák

- VLSI: ...
- FPGA: ekvivalens kapuszám
  - mesterséges mérőszám: nem a legpontosabb

Xilinx: Spartan

- a legkisebb elem is 100.000 kapu bonyolultságú
- a kisebbek elég elfogadható áron vannak
- nem összehasonlítható egy mikrovezérlővel, de más is az alkalmazás

Logikai cella felépítése

Xilinx XC3S250E SliceM logikai cella

- a belső pontokban a megfelelő konfigurációs beállítások elemzése fontos
- az is fontos, hogy a szomszédos egységek milyen huzalozási kapcsolatban lehetnek

Actel logikai cella

- MUX alapú

I/O:

- alapfunkciója csak a be/kimenet
- egyéb funkciókat is szeretnénk
  - pl: regiszteres, duplaregiszteres

Huzalozás

- Xilinx rendszer: ngc, ncd
- a neocad felvásárlásából erednek ezek a fájlnevek

Huzalozás fontossága:

- rengeteg huzalozás van

Megoldás:

- inkább nagyobb cellákat valósítanak meg, a vezetékszegmensek csökkentése a cél

Összefoglalás

- 3 feltétel: komplexitás, sebesség, fogyasztás (ezek egyszerre nem teljesíthetőek)

SRAM FPGA Architektúrák

- összehasonlítás: a másfél éve újnak számító legnagyobbak (Stratix3, SC/M)
- következő dia: kicsivel olcsóbbak

Példa: Xilinx Spartan3E

- a technológia fejlődésével a középső mag egyre kisebb lett, és komplexebb

CLB struktúra

Gyors átvitelképzés

Szorzás logikával

- a szorzó komplexitása négyzetesen változik (az összeadóé lineáris)
- a belső szorzót érdemes használni

#### Belső multiplexerek

- a belső négy bites LUTokat 5 változós függvényekké foghatjuk össze
- növekszik a késleltetés idő, de nem annyira, mintha önálló CLB szinteken valósítanánk meg

#### LUT felépítése

- nem annyira érdekelt minket a felépítése
- a szintézer képzi le
- de akkor érdekesek az egyedi megoldások, amikor hasonló funkciót valósítunk meg nagyon sokszor

#### LUT RAM

#### LUT shift-regiszter

- egyfajta késleltetőnek tekinthető
- órajel engedélyezése van
- hosszúságát állíthatjuk
- FIFO-t, számlálót könnyű vele csinálni

#### LUT SRI

- véletlenszám generátor

#### LUT SRL

- nagyobb modulusú számláló
- egy ilyen számláló akár 300 Mhz-cel tud menni

#### LUT FF

- egyszerű flip-flop megfelelő vezérléssel
- kezdőérték megadható

#### Órajel erőforrások

- mindig egyetlen órajellel működő szinkronhálózatot használunk (ha lehetséges)

#### Órajel kezelés:

- bejövő órajelre szinkronizálás (a bevezetés által okozta késleltetés kiküszöbölése)
- belső szintézerrel további órajelet lehet előállítani
- fázistoló: finom állítás (szinte tetszőleges pozícióba állítható órajel)

#### Egyéb erőforrások:

- DSP
- nagysebesség soros interfészek
- beépített CPU



## Rendszerarchitektúrák előadás

2010. április 19. hétfő

### Globális órajel hálózatok

- igyekszünk minimalizálni a szükséges órajelek számát
- órajel: multiplexeren keresztül át tudjuk kapcsolni egyikről másikra
  - órajelátkapcsoló: van egy gyors és egy lassú órajelbemenete (+ egy választó bemenet)
    - bármikor történik átkapcsolás, mindig alacsony szinten kapcsol át
    - ha kell, akkor természetesen várakozik egy kicsit, hogy át tudjon kapcsolni
  - tranziens nélküli átkapcsolás
  - szükséges: nehogy valamilyen állapotátárolónál probléma legyen
- órajelforrás: tetszőleges belső jel is lehet

### Kevés órajelnél

- statikus időzítés analízis: jelterjedési idő számítása forrásoktól terhelésekig
- ### Órajel kezelő modul
- pl. késleltetéssel megoldja, hogy valamennyi késleltetés történt a jelterjedésben
    - ez periodikus jeleknél megoldható

### Órajel nélküli adatátvitel esetén

- órajel regenerálás
- ha jelben van elég élváltás, akkor képesek vagyunk az órajelet visszaállítani

- PPC processzor: önmaga tud működni 2-300 Mhz-cel, de ha valamit hozzáteszünk, akkor már nem
    - vagy lelassítjuk a processzort és illesztjük a buszrendszerhez, vagy külön órajelről használjuk őket és mindig szinkronizálunk
    - a processzor órajeléből származtatunk egy lassabb órajelet, ez jó megoldás, ebben az esetben szinkronizáció is biztosított
- ### Egyéb erőforrások:

...

### Speciális blokkok

- ezen egységek beültetésével megszűnt az egységes felépítés, a rendszer kialakítása bonyolultabb lett

### Blokk RAM

- adott méret: 2-4 KB
- a szélessége a felhasználótól függ
  - vagy egy nagyon mély 1 bites memória: valamilyen soros adatot szeretnénk beléptetni
  - szélesebb memória

### READ\_FIRST

- egy adott helyre írás történik, akkor előtte a korábbi adatot kiolvassa és csak utána fogja az adatot írni

### WRITE\_FIRST

- mintha transzparens
- a beadott adat egyúttal megjelenik a kimeneten is
- FIFO megvalósítása

- a blokk RAM-ok használata elvileg gondtalan
- de a valóságban ütközéses szituációk állhatnak elő
- az áramkör nem oldhatja meg az összes problémánkat

### BlokkRAM használat

- tulajdonságok:
  - bekapcsoláskor tudunk neki kezdőértéket adni
- a memóriához hozzátették a vezérlő logikát

## Szorzó

- szorzás optimalizálása mindig is fontos kérdés volt
- amikor egy jelfeldolgozó algoritmust leképzünk FPGA-ra akkor nagyon sok lehetőségünk van
  - pl. összeadók csoportosítása
  - kevesebb szorzó használata: kevesebb hardver, több órajel (multiplexer: választunk, hogy melyik bemeneteket szorozzuk össze)
  - vagy pl. lineár fázisú szűrő: szimmetrikusak az együtthetők, ez további optimalizálásra ad lehetőséget
  - a FIR szűrőt teljesen párhuzamosan valósítjuk meg
- ezt a sok trükközést egy hatékonyabb szorzóval kiválthatjuk
- egy komolyabb eszközben több száz szorzó van
- nagy fókuszú szűrő is működhet 100-200 Mhz-en

## I/O tulajdonságok

- egyetlen rossz tulajdonsága van az FPGA-nak: nagyon sok lába van
- sok tápfeszültség és föld láb van alapból
- nem ritkák a 12-14-16 rétegű nyomtatott áramkörök

## I/O blokk alapfunkciója

- DDR2
- két regiszter, DDR muxon keresztül ki tudjuk kapcsolni őket a kimenetre

## Select IO konfiguráció

- szinte az összes lehetséges áramköri szabványhoz kénytelenek alkalmazkodni
- megvannak a hagyományos CMOS, TTL szintű jelek
- hamar lemondtak a 3V feletti jelszintekről
- nagy sebességű TTL logikák: referenciafeszültséget használnak
- CMOS: tápfeszültség arányaihoz képest adja meg a szintet
- sokáig problémát okozott a PCI kompatibilis meghajtás beállítása: nem lehetett nagy túllövés a feszültségben

## IO meghajtás típusok

### Integrált forrásimpedancia beállítás

- ahhoz, hogy egy memóriavezérlő egy nagy sebességű memóriával megfelelően tudjon együttműködni, szükség van a tápvonal megfelelő impedanciabeállítására
- korábban: rengeteg ellenállás (a felületre mindig fel kellett jönni)
- most: a belső IO blokkok tartalmazznak hangolható ellenállást

## Mikroprocesszoros rendszer tervezése FPGA-val

- kezdetben: az FPGA megpróbálta a rést betölteni a CPU és egyéb logikai eszközök között
- később: elérték azt a funkcionalitást, hogy azokat a funkciókat, amelyeket a beágyazott rendszerek megvalósítottak, az FPGA-k is megvalósították
- majd: "bármire szükségünk van, FPGA-val valósítsuk meg" :)
  - de: az áruk nagyon drága
- rendkívül támogatott manapság mikroprocesszoros rendszerek tervezése
- nem a Xilinx találta ki, csak követő volt

## Design Process

### Development and Verification

- jó támogatottság jellemző
- tudunk tetszőleges szinten szimulációt végezni
- régebben: szabad lábakra vezettek ki belső jeleket

- chip-scope: ahhoz, hogy használni tudjuk, erőforrások kellene (RAM-okat is foglal)

- analóg bemenetek
- busztopológiák
- processzorok tulajdonságai
- szenzorhálózatok
- FPGA