BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Mikrorendszerek felépítésének általános modellje

Fehér Béla, Raikovich Tamás
BME MIT

Hardver lehetőségek

Régebben :

- 1-2 mikron CMOS technológia
- 50 – 100 mm2 felület
- 100000-300000 tranzisztor
- 20-50 MHz, 1-2 W

Ma:

- 0.1-0.03 mikron, 10-20 mm2 felület
- 5 -20 millió tranzisztor, 500-1000 MHz

FPGA fejlődés

1985: Első FPGA példányok 1000 kapu

1990: XC4000 sorozat 10000-50000 kapu

2000: Virtex sorozat 1000000 kapu

2005: Virtex II Pro legújabb 10000000 kapu

2010: Virtex 5, Virtex6

2015: Kintex, Virtex, Zynq UltraScale

A félvezető technológia egyik motorja

A probléma

A tervezési olló egyre tágul

- a technológia 5 év alatt 7-10 szeresre javul
- a rendelkezésre álló kapuszám több mint 50x
- a tervezésre fordítható idő folyamatosan csökken (évről-évre 20-25%), ez kb. 2x
- új verziók kiadása még gyorsabban

KÉRDÉS:

Tudjuk követni 100x hatékonyabb tervezéssel?

Mi a megoldás?

- **A (nyers) HW olcsó és elérhető**
- **Hogyan használjuk ki?**
- **Hogyan tervezhetők hatékonyan ilyen komplexitású berendezések?**
- **Hogyan ellenőrizhető a működés?**
- **Hogyan lehet különböző verziókat gyorsan, olcsón fejleszteni?**

Rendszerterv variációk

Adott tengernyi logikai kapu

Tervezzünk funkcionálisan, alulról építkezve?

- Milyen mélyről induljunk?
- Milyen korábbi eredmények használhatóak újra?
- Mi biztosítja az új rendszer helyes működését?

Tervezési lépések

Az algoritmus és architektúra elemzése

Kezdetben figyelembe veendő az összes alternatíva

Folyamatos finomítás a paraméterek területén, választás az egyesített tulajdonságok alapján

Lehetséges opciók (1)

Tervezés adatfolyam modell alapján

- Absztrakciós szint alacsony
- Alkalmazandó eszköz: bit szintű tervezést biztosító, konfigurálható áramkör eASIC/FPGA

Tervezés műveleti igények alapján

- Byte/szó alapú ALU-k, ezek hálózata és centralizált/decentralizált vezérlés
- Alkalmazandó eszköz: Byte granularitásu FPGA-k, sok műveletvégzős hálózatok

Lehetséges opciók (1)

Egyetlen nagyteljesítményű processzor

- VLIW Very Long Instruction Word proc.
- Tipikusan mikroprogramozott, Harvard felépítésű, sok (többfajta) műveletvégző
- Optimális jel és képfeldolgozásra
- Superscalar processzor: általános célú nagyteljesítményű feldolgozó

Legfőbb előny: Könnyű alkalmazás, hatékony fejlesztési algoritmus leképezés

Lehetséges opciók (3)

Multiprocesszor hálózat

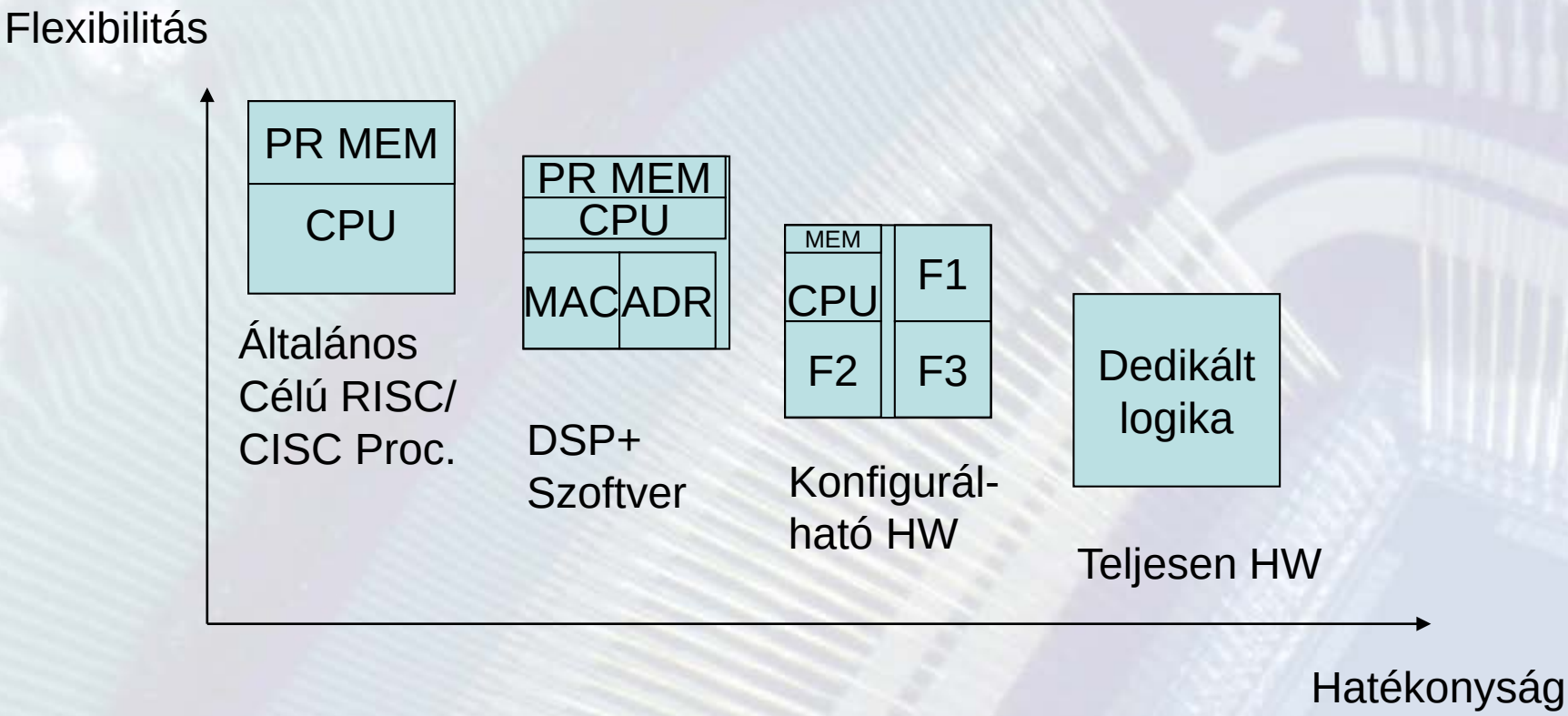
- Nincs igazán győztes modell
- Egyszintű mátrix elrendezés, azonos elemek
- Hierarchikus hálózat master és slave egységek

Rendszer felépítés első közelítésben egyszerű, a gondok rejtve vannak

- Kommunikáció, szinkronizáció,
- Terhelosztás statikus, dinamikus

Flexibilitás - hatékonyság

Ellentmondó fogalmak



HW megvalósítás

Felépítjük az áramkört

- Nincs utasítás elővétel
- Nincs regiszter töltés
- Közvetlen adatfolyam
- Esetleg optimalizált paraméterek

Nevezhetnénk ZISC-nek is, (Zero Instruction Set Computer), de az már foglalt

Megvalósítás

Az FPGA-k kínálják ezt a lehetőséget

- Alacsonyszintű erőforrás hozzáférés
- Nagy komplexitás

Tipikus módszer napjainkban

- Algoritmus leírása HDL nyelven
- Szintézis, elhelyezés, letöltés

Általában külső memória szükséges

- Input, output

Rendszerszintű igények

Feldolgozási feladatokra

- Közvetlen adatfolyam leképezés optimális
- Kevés erőforrás, hatékony működés
- Aritmetikai – logikai – kommunikációs funkciók

Vezérlési és management funkciók

- Kisebb állapotgépek OK
- Néhány x10 állapotra strukturális vezérlők

Rendszerszintű feladatok

Közvetlen igény belső vezérlők megvalósítására

Példa:

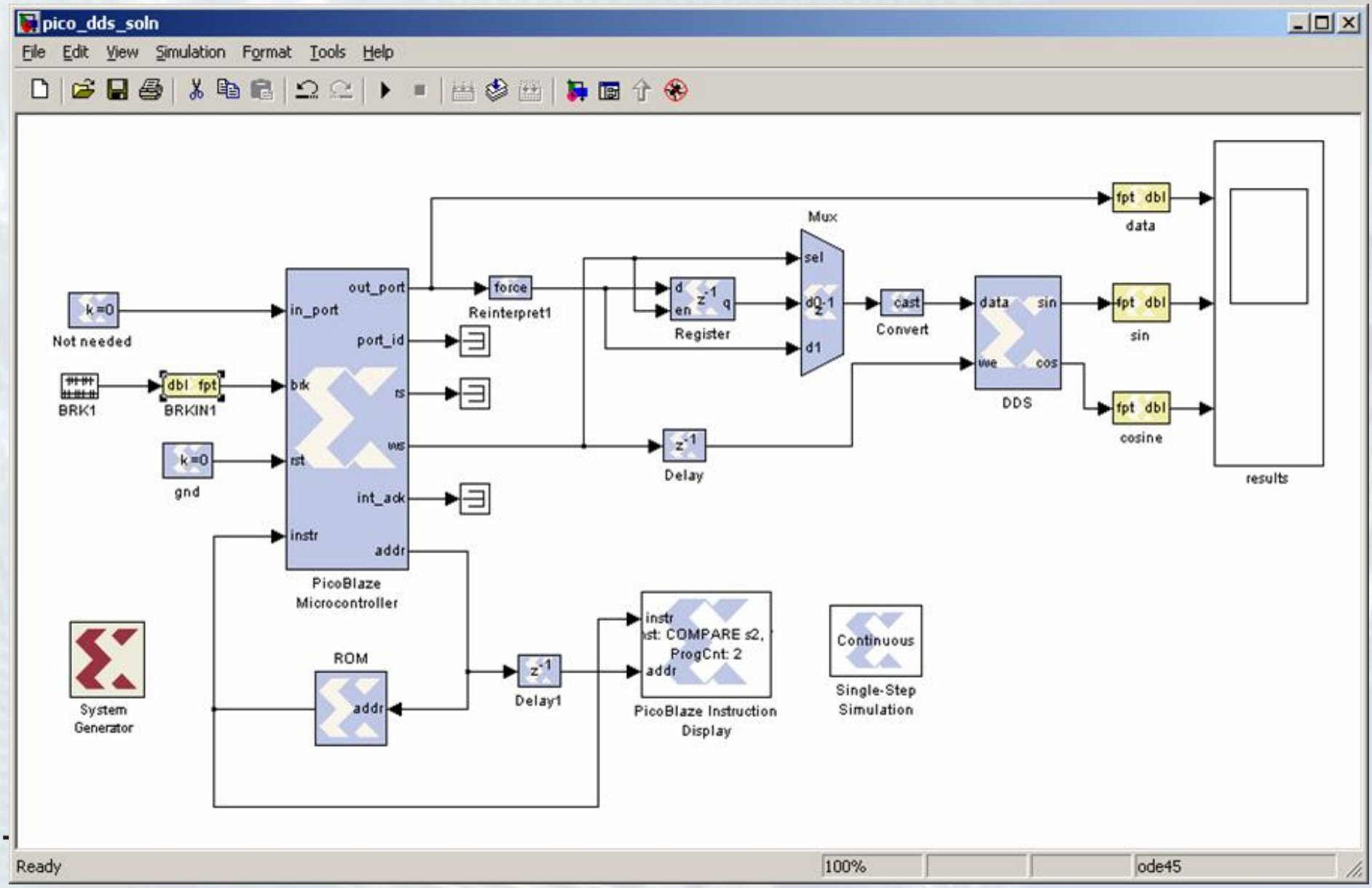
- Önálló labor feladat 1994
- HIFI Audio tesztminta generátor 20Hz-20kHz
  - Torzítás (Lineáris, intermodulációs, stb.)
  - Jelgenerálás egyszerű DDS + ΔΣ konverter
  - XC4005 + külső A/D + LCD kijelző + gombok

Tanulság

Tervezési folyamat

- DDS Jelgenerátor OK
- A/D + Analóg kiegészítők OK
- Kezelői felület ??????
  - Menürendszer összetettsége
    - Főmenű + almenük + paraméterek tárolása
  - Kijelzés komplexitása
    - Feliratok + poziciók
- XC4005 196 CLB ~ 5000 kapu

PicoBlaze alkalmazási példa



FPGA Soft Cores

Lágy processzor mag

- Teljes mértékben az általános erőforrások felhasználására épül
- HDL RTL szintű leírásból, szintézis útján
- Az architektúra paraméterei fejlesztési időben módosíthatók
  - Memória méret
  - Utasítás szerkezet
  - I/O kiegészítő funkciók

Tipikus példák

Oktatási mintapéldák

- GNOME (4 bit adat), DWARF (8 bit)

Egyedi, nyílt kódú fejlesztések

- MicroRISC, MiniRISC, OpenRISC, > 25 db

Valódi, szabvány processzorok verziói

- 8 bit, 8051, PIC16xx, Z80
- 16 bit ????
- 32 bit MIPS, DLX

Gyári processzor struktúrák

Altera Nios, újabb verzió Nios II

- Kezdetben két változat: 16 és 32 bites architektúra, azóta komoly evolúciós fejlődés
- 50-150-300 MHz működési sebesség
- Speciális busz struktúra (Avalon)
- Teljes HW és SW támogatás
  - Konfigurációs minták
  - Kiegészítő rendszer elemek
  - Fordítási eszközök
  - Debug és hibakeresési eszközök

Gyári processzor struktúrák

Xilinx MicroBlaze

- 32 bites architektúra, jelenleg v8
- 100-150 MHz sebesség
- Korábban IBM Core Connect busz struktúra (OPB)
- Jelenleg ARM AMBA AXI komm. hálózat
- Teljes HW és SW támogatás
  - Konfigurációs minták
  - Kiegészítő rendszer elemek
  - Fordítási eszközök
  - Debug és hibakeresési eszközök

Gyári processzor struktúrák

Lattice Mico32

- Teljes 32 bites Harvard RISC felépítés
- Szokásos sebesség (~100MHz), tulajdonságok
- Whisbone buszstruktúra
- Megfelelő HW és SW támogatás
  - Konfigurációs minták
  - Kiegészítő rendszer elemek
  - Fordítási eszközök
  - Debug és hibakeresési eszközök

Gyári processzor struktúrák

Microsemi RISC-V

- Új szereplő a területen, globálisan jelentős szereplő lehet később
- Alapvetően valódi VLSI CPU, de természetesen megjelent FPGA lágyprocesszorként is
- 32 bites Harvard RISC felépítés
- Hagyományos ARM AHB/APB buszstruktúra+AXI4
- Megfelelő HW és SW támogatás
  - Konfigurációs minták, Kiegészítő rendszer elemek
  - Fordítási eszközök, Debug és hibakeresési eszközök

Felső kategória

FPGA-ba ágyazott valódi processzorok

A legnépszerűbb típusok:

- ARM, MIPS, PowerPC (lényegében kihaló…)
- Előny: Garantált viselkedés és teljesítmény
- Szabvány, fix, eredeti áramkör, a szokásos konfigurációkban
- Nem használ FPGA erőforrást, de rendelkezik gyors, nagy sávszélességű belső kapcsolattal

SoC eszközök

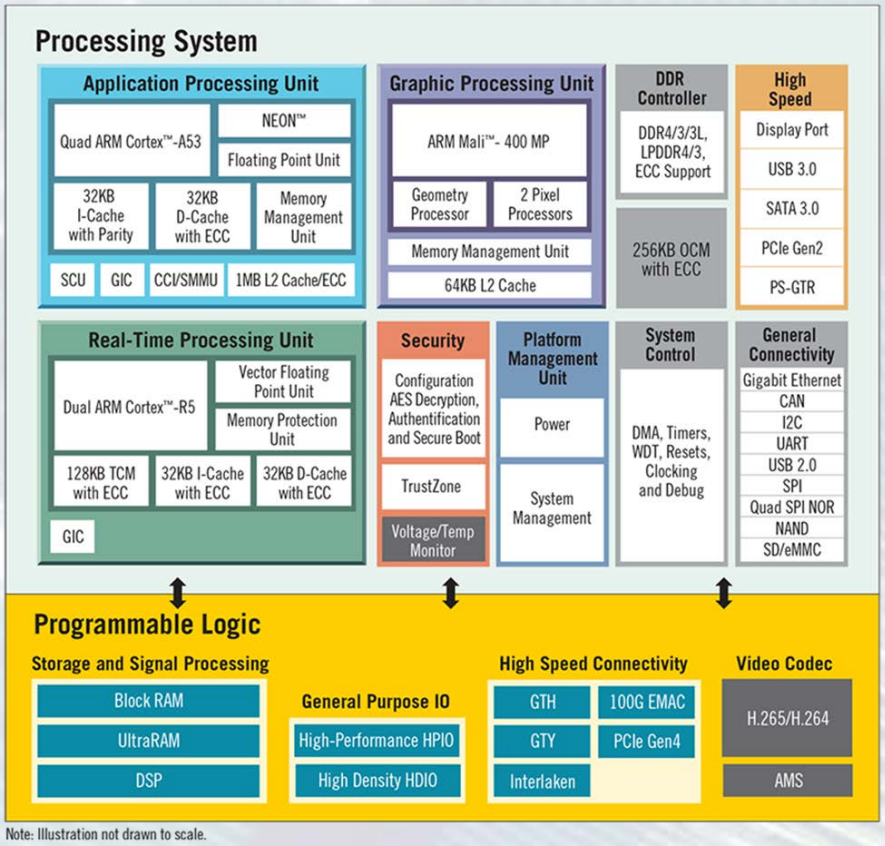
Új kategória:

- Nem önálló CPU mag az FPGA mezőben

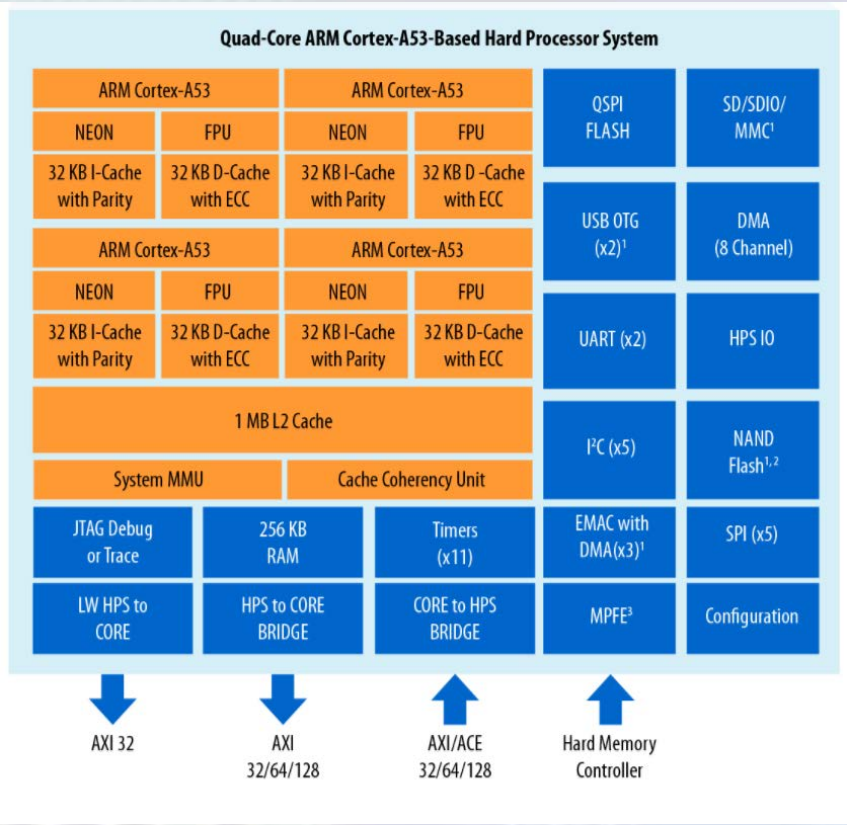
Nagyteljesítményű (többmagos) CPU rendszer, teljes perifériakészlettel

- Kiegészítve/egybeépítve egy FPGA programozható eszközzel
- Elsődleges a CPU rendszer
- DE az FPGA teljes együttműködése jelentős hatékonysági, flexibilitási, alkalmazási előnyöket biztosít (Xilinx, Intel/Altera)

SoC eszközök

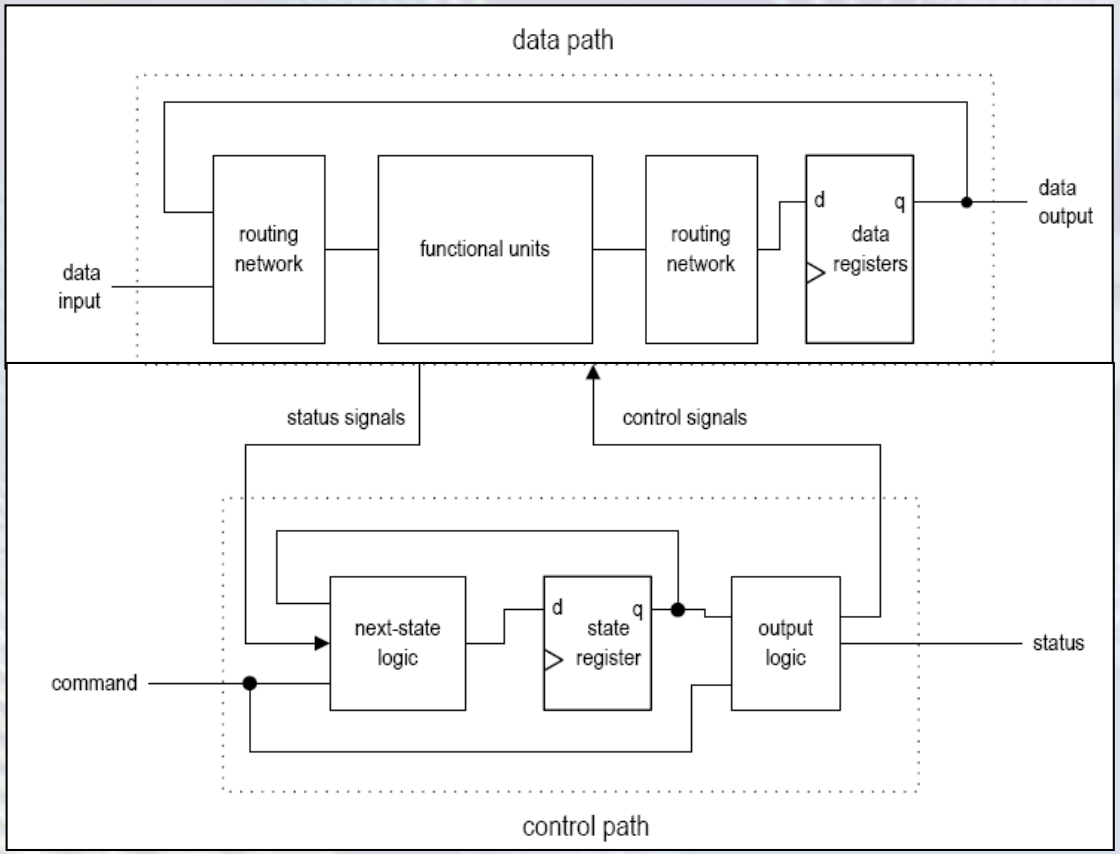


Note: Illustration not drawn to scale.



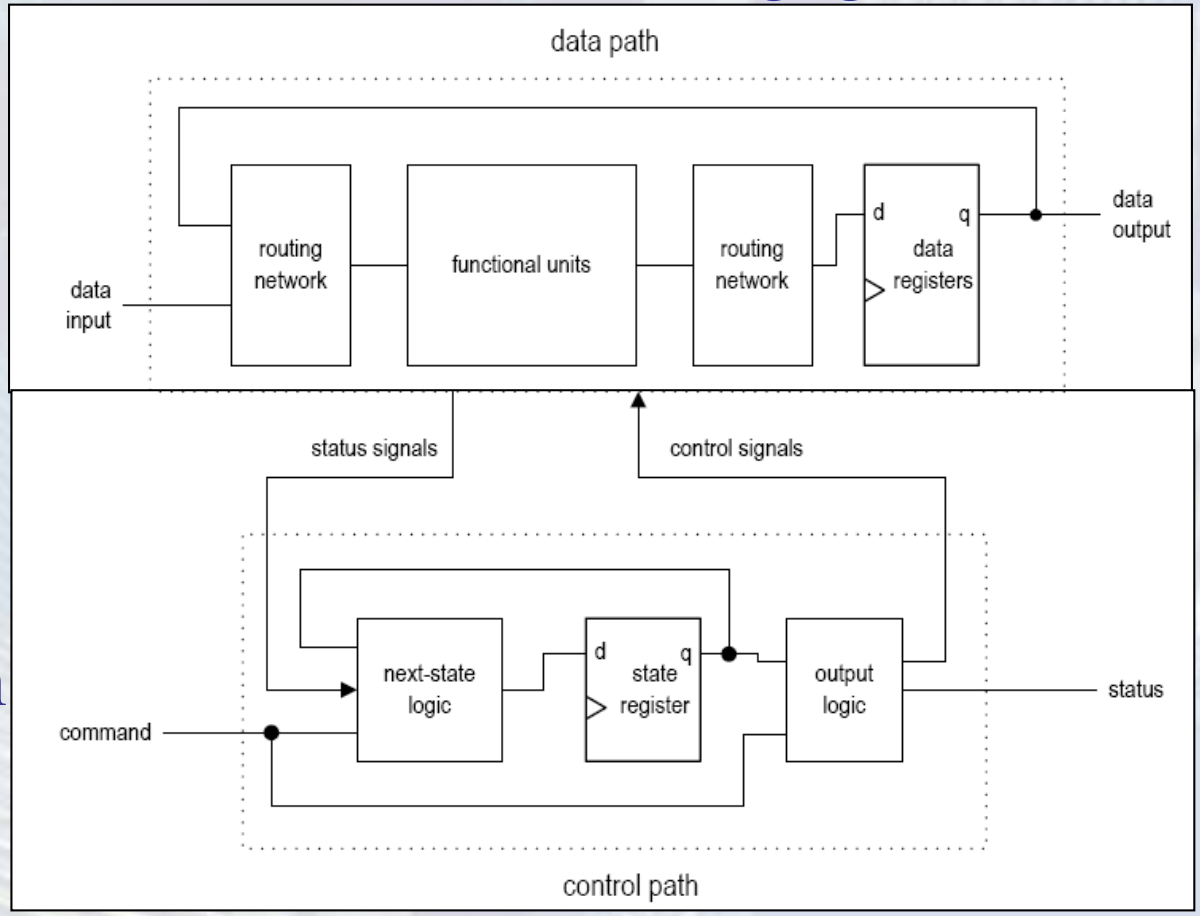
Processzor tervezési alapok

- **ADATSTRUKTÚRA + VEZÉRLÉS**
- **Vezérlés: Minden feladatra egységes általános elv**
  - De természetesen egyedi állapotdiagram, jelek, stb.
- **Adatstruktúra:**
  - Egyedi felépítés, feladatorientált kialakítás
  - Megmaradt a HW párhuzamos, időben konkurens működése



Processzor tervezési alapok

- **ADATSTRUKTÚRA + VEZÉRLÉS**
- **Adatstruktúra: Ahány feladat, annyi architektúra**
  - Lehetne itt is általánosítani? Valószínűleg igen…
  - Feladjuk a párhuzamos működés előnyét az egyszerű tervezhetőség érdekében
  - Sok elemi művelet időben sorban egymás után



Processzor adatstruktúrák

- **Milyen legyen az általános adatstruktúra?**
- **Komponensek: adattárolók, műveletvégzők, huzalozás**
  - Adattárolók:
    - Regiszterek: bármelyik adat bármikor elérhető
    - **Regisztertömb**: korlátozott számú adat érhető el (1,2,3)
    - **Stack**: csak a stack „teteje” érhető el (1,2)
  - Műveletvégzők:
    - Egyedi elemek, közvetlen egyedi bemenetekkel
    - **Többfunkciós egység**, funkcióválasztással és bemeneti operátor kiválasztással (ADD/SUB/COMP, AND/OR/XOR)
    - **Általános státuszjelek** a művelet eredményéről (Z/C/N/V)
  - Huzalozás:
    - **Bemenet/ kimenet kiválasztás/aktiválás**
    - **Belső adatutak, operandus kiválasztás**

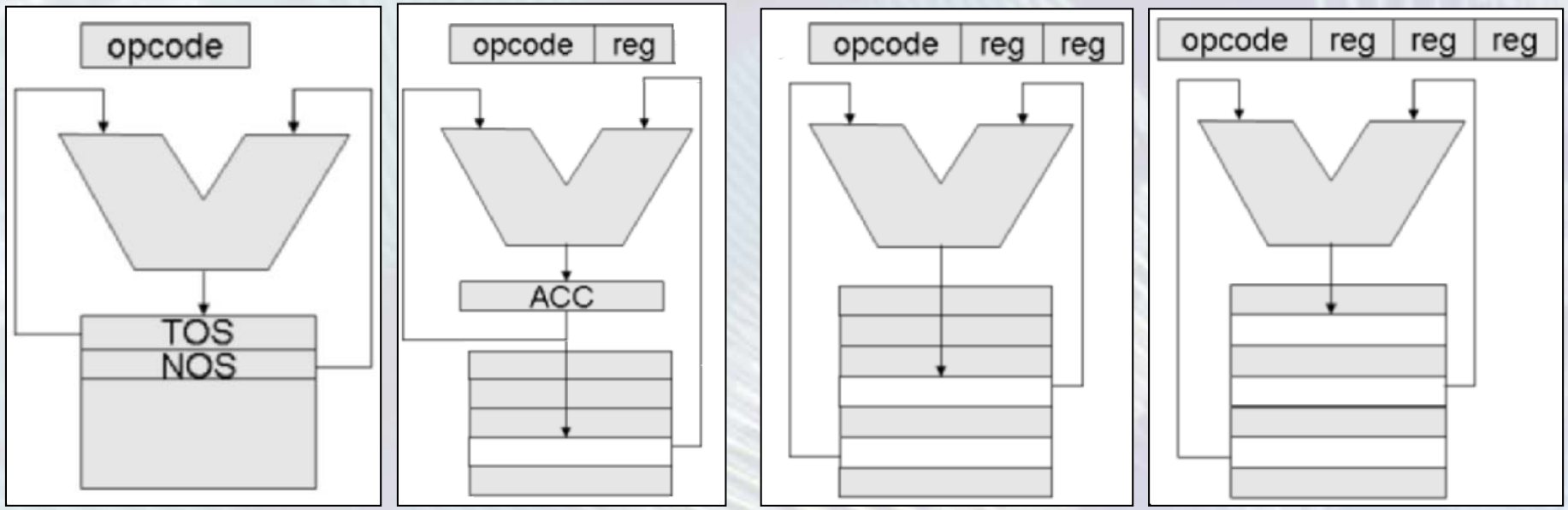
Processzor adatstruktúrák

- **Az általános adatstruktúra?**
- **Szabványosítás:**
  - Adatméret: minden adat azonos méretű pl. 8 bit
  - Egyedi jelek lehetnek egy bitvektor elemei, beolvasunk 8 bitet, maszkoljuk a kívánt pozíciót
  - Az eddigi külső vezérlőjelek (pl. START) is adatbemenetként kezelhetőek, beolvashatóak, tesztelhetőek és a teszt eredménye szerint használhatóak a vezérlési feladatra
  - A kimeneti jelek hasonlóan egységesen kezelhetők, 8 bites bitvektorok közvetlenül, egyedi bitek bitpozíció beállítással kiadhatók
  - A vezérlő egység kimenete (pl. READY) is így kezelhető

Processzor adatstruktúrák

- **Típustervek az általános adatstruktúrára**

STACK **ACC** **2REG** **3REG**



- **Összehasonlítás, elemzés, hatékonyság vizsgálat egy egyszerű példa alapján, figyelembe véve az utasításszó méretét**

Processzor adatstruktúrák

- **Mintapélda: Fibonacci sorozat generálása**

$$F_n = F_{n-1} + F_{n-2}\text{ , és } F_0 = 0, F_1 = 1$$

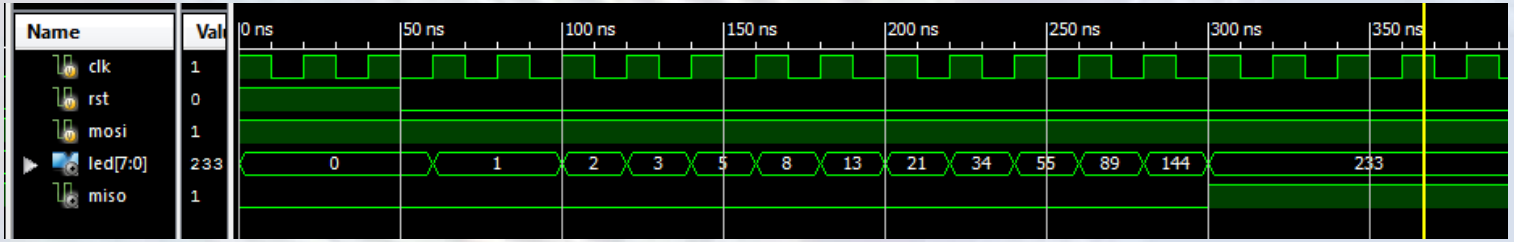
8 biten : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233

- **Vizsgált verziók:**
  - Hardver: Verilog HDL, 2 db REG + ADD
  - Stack alapú + program
  - Akkumulátor alapú + program
  - 2 Regisztercímes + program
  - 3 Regisztercímes + program
- **Szempontok: sebesség, kódméret**

Processzor adatstruktúrák

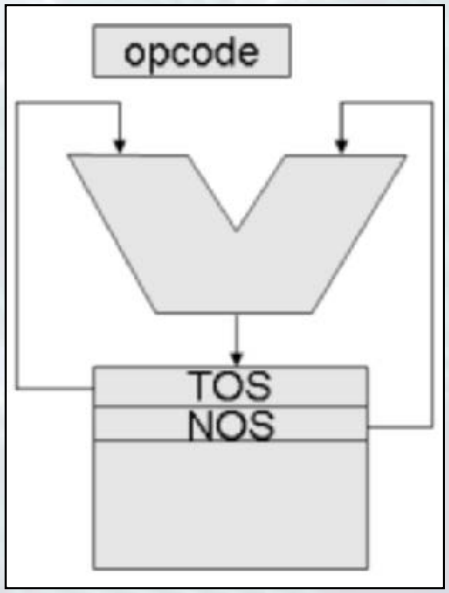
- **Verilog HDL hardver realizáció**
  - **HW költség: 2 db 8 bites regiszter + 1 db 8 bites ADD**
  - **Ha engedélyezett, minden órajelre egy új F_n érték**

```
reg [7:0] current, next;            // Belső változók a Fibonacci sorozat elemeihez
wire en = mosi;                     // Léptetés engedélyezése

always @ (posedge clk)              // Minden működtető órajelre lép egyet
begin
   if (rst)                         // RESET estén inicializálás
      begin
         current <= 8'b0;
         next <= 8'b1;
      end
   else
      if (en)                       // Ha engedélyezett a lépés
      begin
         current <= next;           // megújítja az aktuális értéket
         next <= current + next;    // és kiszámolja az következőt
      end
end
```



Processzor adatstruktúrák

- **Szoftver realizációk !!!!!!PSEUDÓ KÓD!!!!!**
  - **Stack alapú adatstruktúrával**

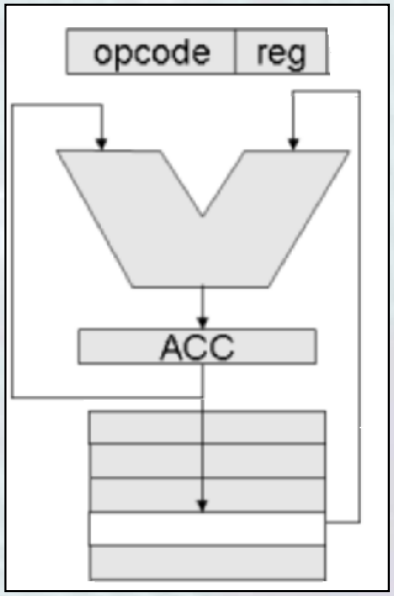



```
;*****************************************************************************
;*  Fibonacci sorozat stack alapú architektúrával PSEUDÓKÓD                  *
;*  Az első néhány elemre, bináris aritmetikával                             *
;*****************************************************************************
DEF LD  0x80                ; LED kimeneti regiszter          (írható/olvasható)

    CODE

;*****************************************************************************
;* A program kezdete.       3 előkészítő utasítás + 3 utasítás a ciklusban
;*****************************************************************************
start:
    push    #0              ; Az F(0) elem a stack tetején, ez mindig az aktuális F(n)
    mov     LD, top         ; Első elem kijelzése
    push    #1              ; Az F(1) elem a stackbe, ez lesz a következő elem F(n+1)
                            ; Stack tetején 1, alatta 0
loop:
    mov     LD, top         ; Kijelezzük az aktuális elemet
    add                     ; top = top + (top-1) és automatikus push
    jmp     loop            ; Iteráció ismétlése frissített F(n), F(n+1) elemekkel
```

- **Program méret: 6 (keskeny) utasításszó**
  - Utasítás formátum: műv_kód (nincs regisztercím)
- **Végrehajtás: 3 utasításonként egy új érték**

Processzor adatstruktúrák

- **Szoftver realizációk !!!!!PSEUDÓ KÓD!!!!!**
  - **Akkumulátor alapú adatstruktúrával**

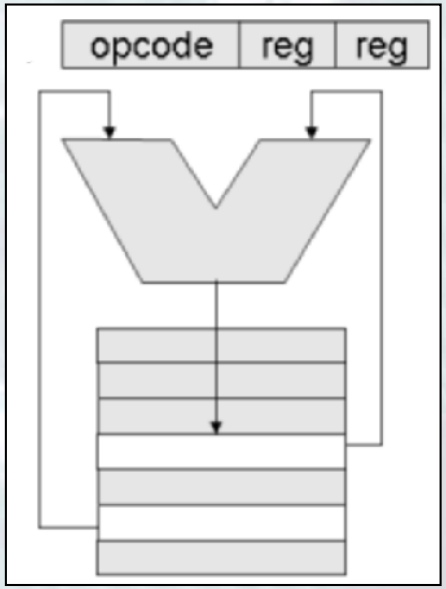



```
;*************************************************************************
;*  Fibonacci sorozat  akkumulátoros architektúrával                     *
;*  A demonstráció miatt az akkumulátor az r7 regiszter
;*  Az első néhány elemre, bináris aritmetikával                         *
;*************************************************************************
DEF LD  0x80                ; LED kimeneti regiszter          (írható/olvasható)

    CODE

;*************************************************************************
;* A program kezdete.       4 előkészítő utasítás + 8 utasítás a ciklusban
;*************************************************************************
start:
    mov     r7, #0          ; ACC beállítása 0-ra
    mov     r0, r7          ; Az F(0) elem értéke, ez lesz mindig az aktuális F(n)
    mov     r7, #1          ; ACC beállítása 0-ra
    mov     r1, r7          ; Az F(1) elem értéke, ez lesz a következő elem F(n+1)
loop:
    mov     r7, r0          ; Aktuális elem másolása ACC-ba
    mov     LD, r7          ; Kijelezzük az aktuális elemet
    mov     r2, r7          ; Aktuális elem átmeneti tárolása
    mov     r7, r1          ; Aktuális frissítése következő elemmel F(n+1)->F(n)
    mov     r0, r7          ; és mentése ACC-ból
    add     r7, r2          ; Következő új értékének számítása F(n+1)+F(n)->F(n+1)
    mov     r1, r7          ; Következő elem átmásolása ACC-ból
    jmp     loop            ; Iteráció ismétlése frissített F(n), F(n+1) elemekkel
```

- **Program méret: 12 (kis méretű) utasításszó**
  - Utasításformátum: ACC = ACC műv REG1 (1 regiszter cím)
- **Végrehajtás: 8 utasításonként egy új érték**

Processzor adatstruktúrák

- **Szoftver realizációk !!!!!PSEUDÓ KÓD!!!!!**
  - **2 címes Regisztertömb alapú adatstruktúrával**

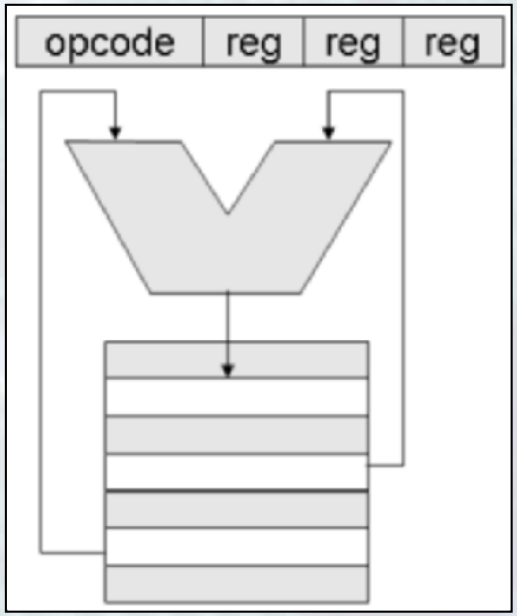



```
;*************************************************************************
;*  Fibonacci sorozat 2R architektúrával                                 *
;*  Az első néhány elemre, bináris aritmetikával                         *
;*************************************************************************
DEF LD  0x80                ; LED kimeneti regiszter          (írható/olvasható)

    CODE

;*************************************************************************
;* A program kezdete.      2 előkészítő utasítás + 5 utasítás a ciklusban
;*************************************************************************
start:
    mov     r0, #0          ; Az F(0) elem értéke, ez lesz mindig az aktuális F(n)
    mov     r1, #1          ; Az F(1) elem értéke, ez lesz a következő elem F(n+1)
loop:
    mov     LD, r0          ; Kijelezzük az aktuális elemet
    mov     r2, r0          ; Aktuális elem átmeneti tárolása
    mov     r0, r1          ; Aktuális frissítése következő elemmel F(n+1)->F(n)
    add     r1, r2          ; Következő új értékének számítása F(n+1)+F(n)->F(n+1)
    jmp     loop            ; Iteráció ismétlése frissített F(n), F(n+1) elemekkel
```

- **Program méret: 7 (közepes méretű) utasításszó**
  - Utasításformátum: REG1 = REG1 műv REG2 (2 reg. cím)
- **Végrehajtás: 5 utasításonként egy új érték**

Processzor adatstruktúrák

- **Szoftver realizációk !!!!!PSEUDÓ KÓD!!!!!**
  - **3 címes Regisztertömb alapú adatstruktúrával**

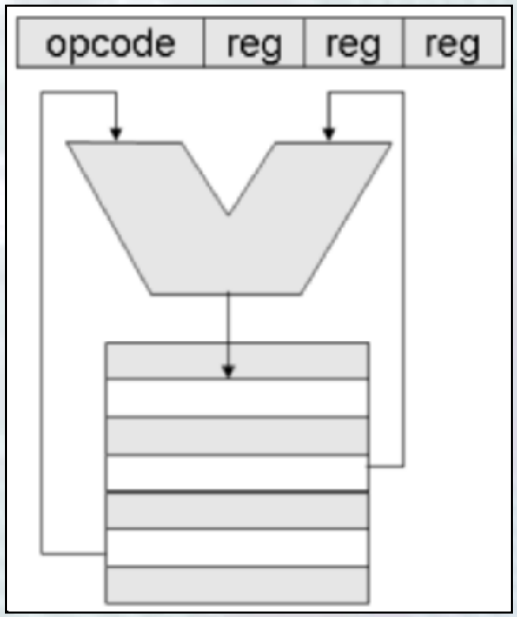



```
;*************************************************************************
;*  Fibonacci sorozat 3R architektúrával                                 *
;*  Az első néhány elemre, bináris aritmetikával                         *
;*************************************************************************
DEF LD  0x80                ; LED kimeneti regiszter          (írható/olvasható)

    CODE

;*************************************************************************
;* A program kezdete.       2 előkészítő utasítás + 5 utasítás a ciklusban
;*************************************************************************
start:
    mov     r0, #0          ; Az F(0) elem értéke, ez lesz mindig az aktuális F(n)
    mov     r1, #1          ; Az F(1) elem értéke, ez lesz a következő elem F(n+1)
loop:
    mov     LD, r0          ; Kijelezzük az aktuális elemet
    add     r2, r0, r1      ; Közvetlenül számoljuk a köv. értéket F(n+1)+F(n)->F(n+2)
    mov     r0, r1          ; Aktuális frissítése következő elemmel F(n+1)->F(n)
    mov     r1, r2          ; Következő új érték frissítése F(n+2) -> F(n+1)
    jmp     loop            ; Iteráció ismétlése frissített F(n), F(n+1) elemekkel
```

- **Program méret: 7 (nagyméretű) utasításszó**
  - Utasításformátum: REG3 = REG1 műv REG2 (3 reg. cím)
- **Végrehajtás: 5 utasításonként egy új érték**

Processzor adatstruktúrák

- **Szoftver realizációk !!!!!!PSEUDÓ KÓD!!!!!**
  - **3 címes Regisztertömb alapú adatstruktúrával, hatékonyabb programkialakítással**




```
;*************************************************************************
;*  Fibonacci sorozat 3R architektúrával, ciklus kiterítéses lineáris kódolással
;*  Az első néhány elemre, bináris aritmetikával                          *
;*************************************************************************
DEF LD  0x80                ; LED kimeneti regiszter          (írható/olvasható)

    CODE

;*************************************************************************
;* A program kezdete.       2 előkészítés + 7 utasítás a ciklusban 3 elemre    *
;*************************************************************************
start:
    mov     r0, #0          ; Az F(0) elem értéke, ez lesz mindig az aktuális F(n)
    mov     r1, #1          ; Az F(1) elem értéke, ez lesz a következő elem F(n+1)
loop:
    mov     LD, r0          ; Kijelezzük az aktuális elemet r0-ból
    add     r2, r0, r1      ; Közv. számoljuk a köv. értéket r2-be F(n+1)+F(n)->F(n+2)
    mov     LD, r1          ; Kijelezzük az aktuális elemet r1-ből
    add     r0, r1, r2      ; Közv. számoljuk a köv. értéket r0-ba F(n+2)+F(n+1)->F(n+3)
    mov     LD, r2          ; Kijelezzük az aktuális elemet r2-ből
    add     r1, r2, r0      ; Közv. számoljuk a köv. értéket r1-be F(n+3)+F(n+2)->F(n+4)
    jmp     loop            ; 3-as iteráció ismétlése
```

- **Program méret: 9 (nagyméretű) utasításszó**
  - Utasításformátum: REG3 = REG1 műv REG2 (3 reg. cím)
- **Végrehajtás: ~2 utasításonként egy új érték !!!!**

Processzor adatstruktúrák

- **A vizsgálatok alapján egy adott mintafeladat különböző feltételekkel realizálható**
- **A megvalósítás fontos jellemzője (költsége): a program teljes memória igénye („memory footprint”)**
- **Asztali vagy nagygépes környezetben nem kritikus**
  - De persze nem elhanyagolható
- **Beágyazott vagy mobil környezetben ma is fontos !!**
  - Pl. Internet of Things eszközök memória korlátja
- **A konkrét esetre, átlagos adatokkal:**
  - 6 bit utasításkód (~50 utasítás), 4 bit reg. cím (16 reg.)
  - Stack: 6 ut.* 6 bit = 36 program memória bit,
  - Akkum:12 ut.* (6+4) bit=120 program memória bit,
  - 2 címes: 7 ut.* (6+4+4) bit = 98 program memória bit
  - 3 címes: 7 ut.* (6+4+4+4) bit = 126 prog. mem. bit

Processzor vezérlés

- **Az általános adatstruktúra jól használható, de képességei korlátosan érvényesíthetők**
- **Egy ütemben egy változó módosulhat**
  - Ez erősen szekvenciális végrehajtást jelent
  - Az ASM működési modell bőven elegendő
  - Nem jelent lényeges előnyt az általános FSM tetszőleges állapotátmeneti képessége

- **ASM állapotátmenetek:**
  - CONT: folytatás (ez az alapértelmezett mód)
  - JUMP: ugrás tetszőleges állapotra (saját magára is)
  - CJMP: elágazás, feltételes ugrás tetszőleges állapotra
    - Feltétel nem teljesülése esetén CONT

Processzor vezérlés

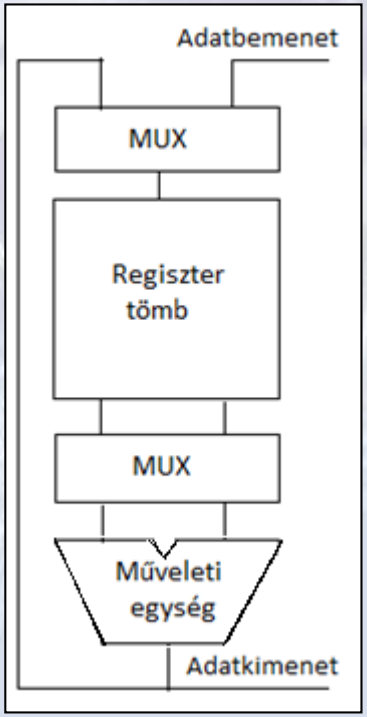
- **ASM algoritmikus állapotvezérlővel**
- **A vezérlő ebben az esetben lehet egy egyszerű számlálón alapuló vezérlőegység: programszámláló (PC, Program Counter)**
- **Az állapotátmenetek vezérlése:**
  - CONT: PC <= PC + 1;
  - JUMP : PC <= LABEL;
  - CJMP : if (COND) then PC <= LABEL
    else PC <= PC +1
- **Ezt a feladatot egy inicializálható, tölthető bináris felfelé számláló tudja biztosítani**
  - RESET-re PC = 0, vagy a csupa „1”, azaz 0xFFFF

Processzor vezérlés

- **ASM algoritmikus állapotvezérlővel**
- **A vezérlőjelek tehát nem közvetlenül a PC értékéből származnak, (annak állapota nem direktben kódolja azokat), hanem a PC tartalma megcímzi és kiolvassa a programmemóriában található utasítás szót (FETCH), és ez tartalmazza a kódolt vezérlőjeleket (annyi biten, amennyi a formátumba belefér), ezt az utasítás dekóder értelmezi (DECODE) és juttatja el az adatstruktúra felé végrehajtásra (EXECUTE).**
- **Tehát az adatstruktúra vezérlőjel generálás folyamata:**
- **PC új érték → FETCH → DECODE → EXECUTE**
  - Ez történik a processzor vezérlő egységében

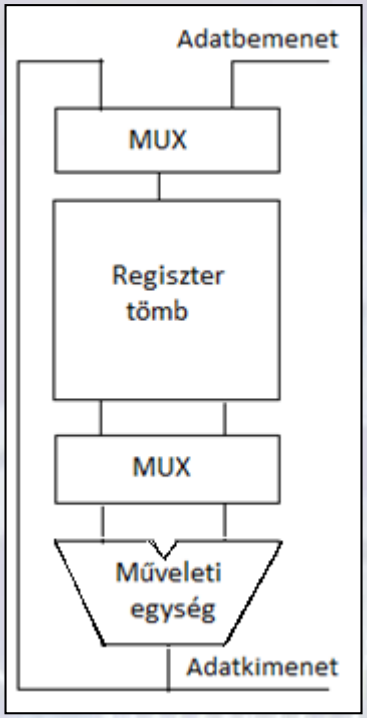
Processzor műveletvégzés

- **Regisztertömb alapú adatstruktúra kiegészítése be-/kimeneti interfésszel (memória, periféria)**
- **Általános tulajdonságok:**
  - Minden adatot először beírunk a regisztertömbbe
  - Műveletet csak a regiszter adatokon végzünk
    - Létezik közvetlen adat programkódból
  - A részeredményeket visszaírjuk
  - A végeredményt kiadjuk
- **Ezt hívjuk LOAD/STORE felépítésnek**
  - A regisztertömb mérete
    - Szélesség: 8/16/32/64 bit
    - Mélység: 16/32/64 regiszter
    - Több regiszter → több címbit (utasításméret)
    - Több regiszter → kevesebb adatmozgatás
    - 32 bites utasításnál jó kompromisszum



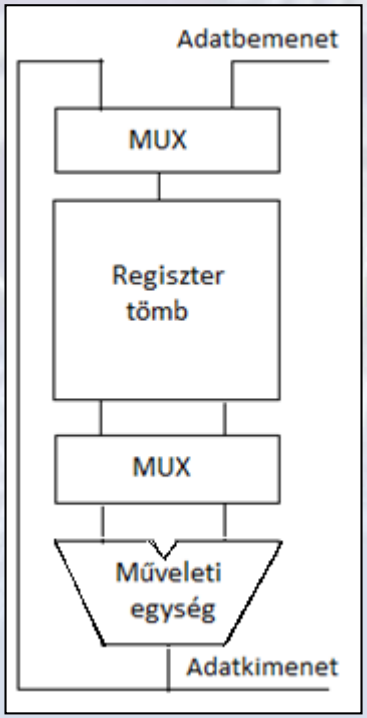
Processzor műveletvégzés

- **Az adatstruktúra műveleti egysége:**
- **ALU: Aritmetikai Logikai Unit**
- **Műveleti képességek (utasítás készlet)**
  - Aritmetikai (ADD, SUB, INC, DEC….)
  - Logikai (AND, OR, XOR, NOT…)
  - Léptetés (SHL, SHR, ASH…)
  - Forgatás (ROL, ROR,…)
  - Feltétel vizsgálat (COMP, TST,…)
  - Egyszerű adatmozgatás (MOV)
- **Minden művelet a szabványos adatméreten**
  - 8 / 16 / 32 / 64 bit, az adott rendszer jellemzője
  - Kisebb adatméret: eredmény maszkolása
  - Nagyobb adatméret: Átvitelbit használatával kiterjesztett műveletvégzés



Processzor műveletvégzés

- **ALU felépítése**
- **1. verzió: Minden feladatra külön áramkör, a kimeneten MUX hálózattal → nem gazdaságos**
- **2. verzió: Sok feladat egyetlen összeadóval + a bemeneten speciális kiegészítés az adatelőkészítésre**
  - Kivonás: Kettes komplemens képzéssel
  - Inkrementálás: „1” hozzáadása (pl. Ci)
  - Dekrementálás: „1” kivonása
  - Léptetés balra: önmagával összeadás
- **Konkrét áramköri megoldások eltérőek**
- **Közös vonás:**
  - Néhány bites vezérlés: FUN
  - Operandus kiválasztás: REG1, REG2
  - Bemenet/kimenet előválasztás



Összegzés

- **Tetszőleges digitális rendszer:**
  - Általános rendszerterv: adatstruktúra + vezérlés
- **Processzoros rendszerek:**
  - Általános processzor adatstruktúra
    + ASM alapú egyszerűsített processzor vezérlőegység
  - **A vezérlési állapot indirekt megadása:**
    - PC (programmemória cím)
      → programtár olvasás (aktuális utasítás)
      → dekódolt vezérlő jelek származtatása és végrehajtás
    - Továbblépés: PC = PC +1 vagy esetleg (feltételes) ugrás
- **Egységesített adatméret és be-/kimeneti interfészek**
- **LOAD/STORE működés, külső vagy memória adatok regiszterbe töltődnek használat előtt**

Egyszerű 16 bites processzor

Jan Gray: XR16

Általános jellemzők

Az FPGA-k komplexitása megengedi mikroprocesszoros alkalmazások megvalósítását

xr16 Pipelined, 16 bites RISC, 33MHz az XC4000 családon belül

C fordító !!!, szimulátor és teljes dokumentáció

- LCC fordító (Princeton) adaptálva az xr16 architekturára
- Local C Compiler/Little C Compiler
  - Alpha, Sparc, MIPS, x86, Microsoft Intermediate Language

Szoftver támogatás

Az LCC fordító adaptálása, illesztése

Eredetileg más processzorokhoz készült

- Az xr16-hoz módosítani kellett az mdf file-t
  - MACHINE DESCRIPTION FILE
- Definiálni és rögzíteni a hívási módszereket
- Regiszterek száma a szokásosnál kisebb, 16
- Alapértelmezett adatméret 16 bit
- Utasítások, utasításcsoportok leírása: ugrások, szorzás/osztás (nincs, szoftver)

Az xr16 utasításkészlete

Célok és korlátok kompromisszumos egyeztetése

- Kompakt 16-bites utasítások
- 16 darab 16 bites regiszter
- C integer műveletek támogatása
- Pipeline, szabályos felépítés -> sebesség
- Bájtos/szavas címezhetőség (load, store)
- Indirekt indexelt címzés (reg)+disp
- Long int támogatás, előjel kiterjesztés, stb.

Utasítások

Általános műveletek

- Leggyakoribb műveletek: load, store, move, call, branch, compare, címbetöltés
- Címszámítás, ellenőrzés
- Indexelt címzés (C fordító hívás átadás)

Jellemzők

- 3 operandusú utasítások dest = src1 op src2/imm
- 4 bites konstans az utasításban (-8 …+7)
- 12 bites konstans speciális prefix módban
- Feltétel kiértékelés külön utasítás

Regiszterhasználati konvenciók a C fordítóhoz

Regiszter	Használat
R0	Mindig 0, nem írható
R1	Assembler munkaregiszter
R2	Függvény eredmény
R3 – R5	Függvény argumentum
R6 – R9	Általános munkaregiszterek
R10 – R12	Változók
R13	Verem mutató (sp)
R14	Megszakítás visszatérési cím
R15	Szubrutin visszatérési cím

Utasítás formátumok

Formátum Neve bitek	15…12	11…8	7…4	3…0
rrr	op	rd	ra	rb
rri	op	rd	ra	imm
rr	op	rd	fn	rb
ri	op	rd	fn	imm
i12	op	imm12		
br	op	cond	disp8	

Teljes utasításkészlet (1)

Hex	Csop	Assembler formátum
0dab	rrr	add rd,ra,rb
1dab	rrr	sub rd,ra,rb
2dai	rri	addi rd,ra,imm
3d*b	rr	{and,or,xor,andn,adc,sbc} rd,rb
4d*i	ri	{andi,ori,xori,andni,adci,sbci,slli,slxi, srai,srli,srxi} rd,imm
5dai	rri	lw rd,imm(ra)
6dai	rri	lb rd,imm(ra)

Teljes utasításkészlet (2)

Hex	Csop	Assembler formátum
8dai	rri	sw rd,imm(ra)
9dai	rri	sb rd,imm(ra)
Adai	rri	jal rd,imm(ra) (ugrás és visszatérés)
B*dd	br	{br,brn,beq,bne,bc,bnc,bv,bnv,blt,bge, ble,bgt,bltu,bgeu,bleu,bgtu} label
Ciii	i12	call func
Diii	i12	imm imm12 (köv. imm. op.[15:4] bitje)
7,E, Fxxx	-	Nem használt, későbbi módosításra (pl. HW szorzó, stb.)

Utasítás végrehajtási kötések

Ha van konstans (imm), akkor a két művelet nem megszakítható (add, sub, shift)

Betöltés/tárolás 2+ ciklus

Ugrás 3 ciklus

4 bites konstans mező lehet

- -8 ..+7 egész (add, sub, log és shift utasításnál)
- 0..15 byte mozgatásnál
- 0..30 páros szó mozgatásnál ill. jal, call utasításoknál

Pszeudó utasítások (1)

Léteznek, mint assembler mnemonik, a logikusabb leírás miatt, de nem külön kódok, meglévő utasítások újabb nevei

A fordító automatikusan ezt teszi a listába

– Honnan tudja, hogy erre illene gondolnia?

Pszeudó utasítások (2)

Mnemonik	Leképezés
nop	and r0,r0
mov rd, ra	add rd,ra,r0
cmp ra,rd	sub r0,ra,rb
subi rd,ra,im	addi rd,ra,-im
cmpi ra,im	addi r0,ra,-im
lea rd,imm(ra)	addi rd,ra,imm
lbs rd,imm(ra) Byte betöltés előjel kiterjesztésel	lb rd,imm(ra) xori rd,0x80 subi rd,0x80
j addr	jal r0,addr
ret	jal r0,0(r15)

Assembler

Olvassa a C fordító által generált listát

Feldolgozza a címkéket, szimbólumokat

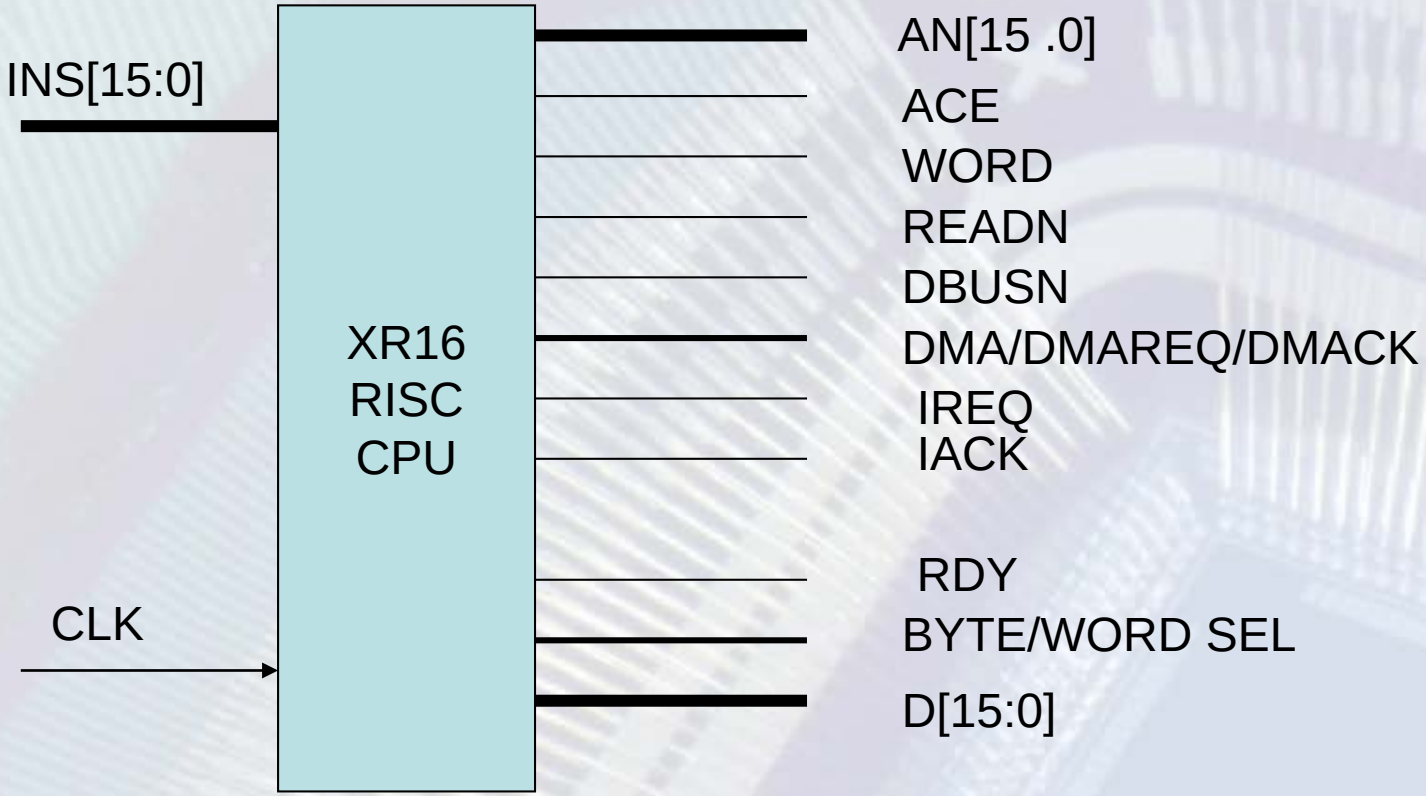
Két menetben dolgozik

- 1. menet Lista beolvasása, referenciák feloldása
- 2. menet Kódgenerálás, közeli-távoli ugrások kezelése

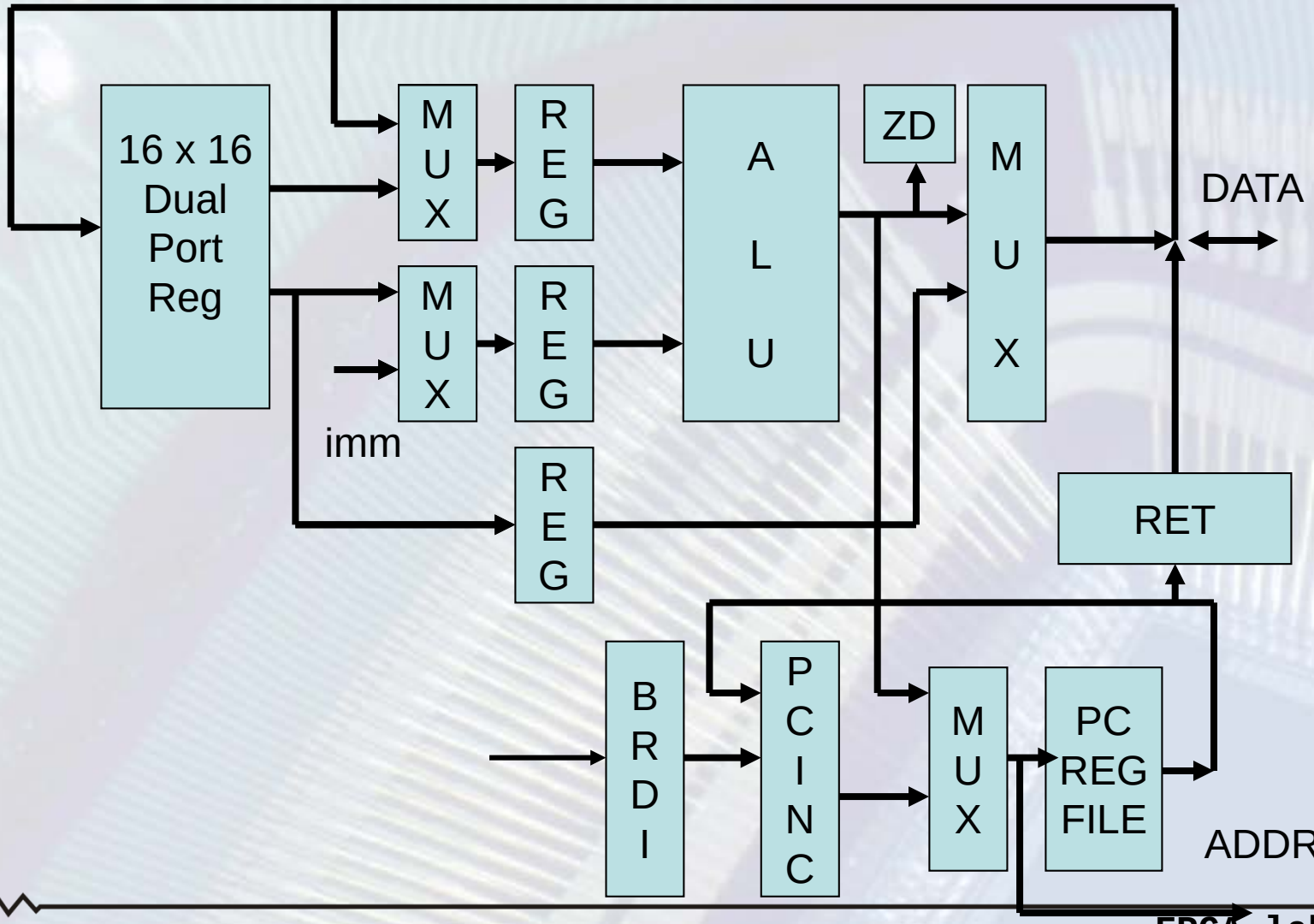
Egy egyszerű utasítás szimulátor is készült

Áramköri felépítés

Egyetlen makro szimbólum



Adatút felépítése



Adatút elemei

Regiszter file 16x16 DP RAM (2RD1WR)

Konstans mező multiplexer

ALU, shifter, effektív cím számláló

Feltétel logika

Programszámláló, ugrási cím képző

PC regiszter egy 16-os készlet egy eleme

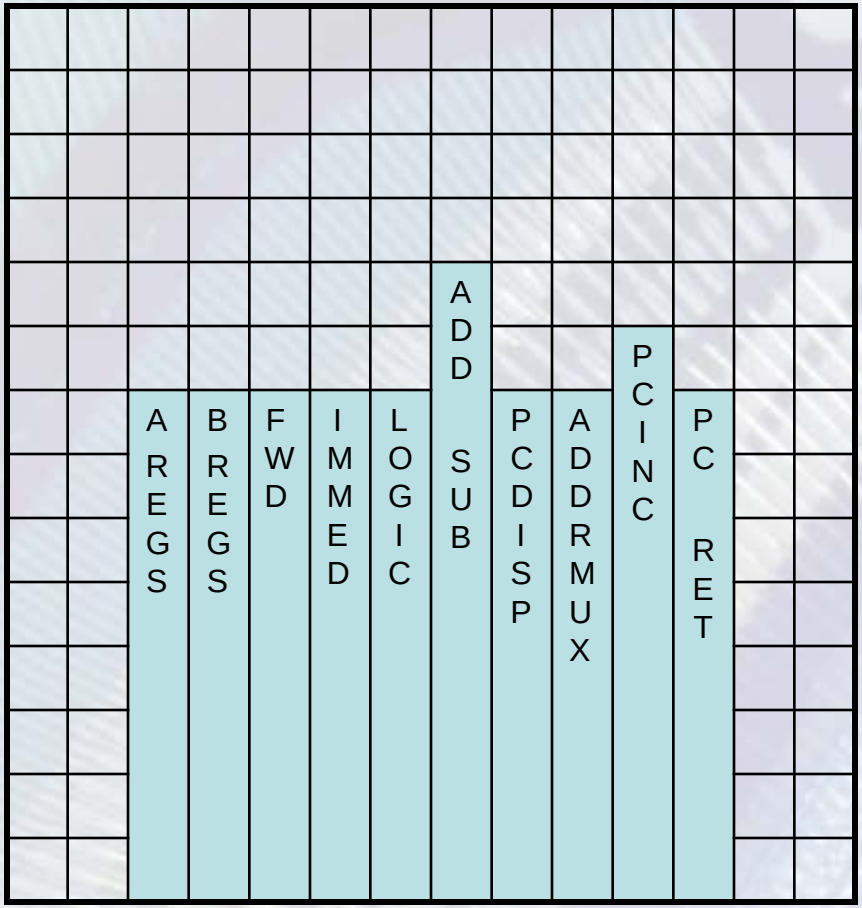
- További címtárolási lehetőség (Pl. DMA, stb.)

Operandus kiválasztás

Utasítás típus	Op A	Op B
add rd,**ra**,**rb**	REG ra	REG rb
addi rd,**ra**,**i4**	REG ra	*sign_ext* imm
sb rd, **i4(ra)**	REG ra	zero_ext imm Dout = REG rd
Imm 0x123 addi rd, ra, 4	Nincs REG ra	Imm12 ||0000 Imm12 || imm4
add r3,r1,r2 add r5,r3,r4	REG r1 eredmény	REG r2 REG r4

Adatút megvalósítás

XC4005 FPGA 14x14 CLB (196 CLB)



Vezérlő egység

Adatfeldolgozó egység: 3 szintű pipe-line

- Regfile_output + operandreg + regfile_input
- Vezérlés: következő utasítás (INSTN)
- Feltételjelek: Státusz információ CZVN
- Speciális pipe-line események:
  - Adat versenyhelyzet
  - Külső memória hozzáférés
  - Elágazás
  - Megszakítás

Utasítás végrehajtás

IF: utasítás elővétel

- programmemória olvasás
- beírás utasítás regiszterbe
- PC = PC + 2 (vagy más érték…)

DC: utasítás dekódolás

- Bitmezők dekódolása
- Műveleti operandusok olvasása

EX: utasítás végrehajtás

- A végrehajtó egységek párhuzamosan dolgoznak, egy eredmény lesz visszaírva

Utasításvégrehajtás

3 fokozatú-pipeline

t1	t2	t3	t4	t5
IF1	DC1	EX1		
	IF2	DC2	EX2	
		IF3	DC3	EX3
			IF4	DC4

Pipe-line események

Normál, sorrendi végrehajtás

Adat versenyhelyzet

- Előző eredmény még nincs visszaírva, de operandusa a következő utasításnak
  - Megoldás1:
    - Assembler utasítások átrendezése
    - Végrehajtás felfüggesztése, amíg az eredmény elérhető
  - Megoldás 2:
    - Adat előre csatolás (egy fokozat átugrása)
    - ! Csak az A operandusra!

Pipe-line események

Külső memória hozzáférés

- Külön memória interfész, hozzáférés arbitráció és szinkronizáció
- Két master jellegű funkció: CPU + DMA jellegű képernyő vezérlő
- CPU olvasást indít és vár a kész (RDY) jelre
  - Amíg nincs új utasítás a pipe-line leáll
  - Belső írás engedélyező jelek inaktívak, az utolsó utasítás EX fázisa ismétlődik …

Utasítás végrehajtás

Memória elérés várakozással (a memória gyors, 1 ciklusos, de IF3 alatt éppen más hozzáférés volt)

t1	t2	t3	t4	t5
IF1	DC1	EX1	EX1	
	IF2	DC2	DC2	EX2
		IF3	IF3	DC3
				IF4

t3 üres, pipe-line várakozás akár n ciklus is

Memóriareferenciás utasítások

Betöltés – tárolás (Load – store w/b)

- Egy plusz memória művelettel
  - Versenyhelyzet a következő utasítás elővétellel
  - Valószínűleg az adat operandusa a következő műveletnek is (assembler átrendezés!)
- Minek legyen prioritása?
  - Hajtsuk végre a következő elővételt, és az utasítás legyen készenlétben (speciális tároló regiszter)
  - Fejezzük be az aktív, éppen végrehajtott műveletet

Előző két probléma együttesen is előfordul

Utasítás végrehajtás

Memória elérés + adatmozgatás várakozással

t1	t2	t3	t4	t5	t6	t7	t8	t9
IF1	IF1	DC1	DC1	EX1	EX1	EX1	EX1	
		IF2	IF2	DC2	DC2	DC2	DC2	EX2
				IF3	IF3	IF3	IF3	DC3
								IF4

1 várakozáskérés ciklus a memóriának, ezért minden ciklus 2 ütem, hiszen minden ütemben van memória hozzáférés (utasítás elővétel ill. adat)

Adat és programmemória lehetne akár eltérő időzítésű

Programelágazások

Ugrások, szubrutin hívások

t5	t6	t7	t8	t9
EX2				
DC3	EX3			
IFB	DCB	EXB		
	IF5	DC5	EX5	
		IF6	DC6	EX6
			IFU	DCU

(FELTÉTELES) Ugrástól függően a betöltött, esetleg dekódolt IF5, IF6 már nem kerül végrehajtásra

Pipe-line kiürítés és újraindítás (3 utasítás ciklus!)

Megszakítás

Általában

- Interrupt kérés észlelése
- Ugrás a kiszolgáló rutin belépési pontjára
- Visszatérési cím mentése
- Pipe-line kiürítése
- Kiszolgálás elindítása, befejezése
- Visszatérési cím elővétele
- Pipe-line újratöltése

Megszakítás

Egyszerűen a már elővett utasítás felülírása a jal r14,10(r0) utasítással

- Ugrás 0x0010-re, PC mentésével r14-ben
- Visszatérés jal r0,0(r14), ez egy iret utasítás

Nem lehet IT-t elfogadni

- 1. nem megszakítható utasítások között (imm12 típusú kódok)
- 2. ugrás, elágazás idején, amikor a két utasítás a végrehajtásból törlődik

A teljes belépési időveszteség általában 3 utasítás legrosszabb esetben 5 (+ 3 az iret)

Rendszerkialakítás

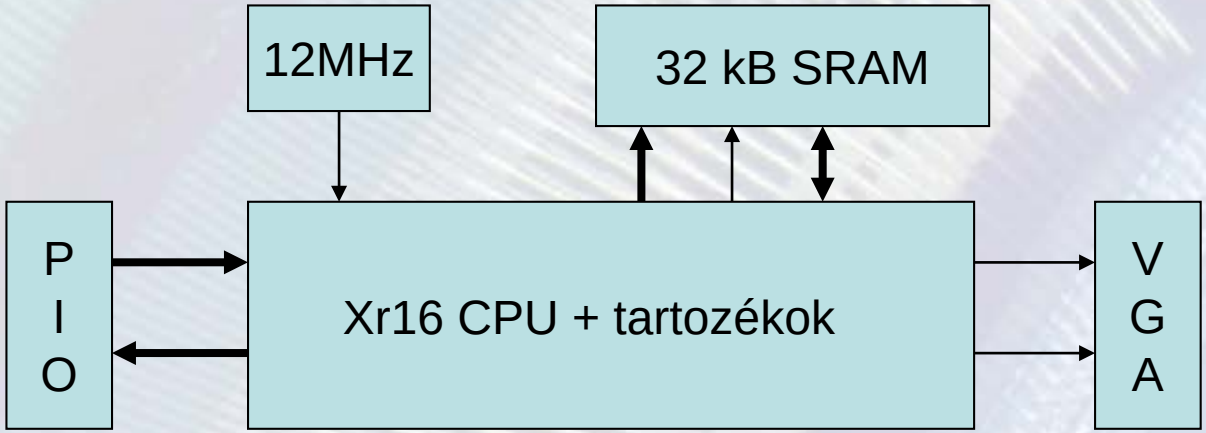
Példamegvalósítás: az XS40 demokártyán

- XC4005XL FPGA 196CLB/~5000 kapu
- 12MHz oscillátor, 32 kB SRAM 15 ns
- Párhuzamos port, VGA port csatlakozók
- 7 szegmenses LED, kapcsolók
- 8031 mikrovezérlő
- Részletek: www.xess.com

SYSTEM-ON-A-CHIP

Az integrált rendszer komponensei:

- CPU + be-/kimeneti perifériák + memória (külső)
- Be-/kimenet: egyszerű párhuzamos port
- Megjelenítés: 576x455 monochrom VGA



SoC buszkialakítás

Milyen busz legyen a chipen belül?

- Rövid áttekintés, később részletes analízis
- Több master-es rendszer: CPU, DMA
- CPU órajel, busz órajel viszonya: 1:1
- Memória portok/blokkok száma: 1
- Busz méretei: Adat, cím, vezérlőjelek
  - Adat: kívül 8 bit (SRAM), belül 16 bit CPU)
  - Cím: kívül 15 bit, belül 16 bit MEM / IO kiválasztás
- Külső-belső busz szétválasztott? Nem
- Buszművelet átlapolt? Igen

Memória térkép

Egyszerű, memóriába ágyazott címtartomány

Címtartomány	Erőforrás
0000 – 7FFF	Külső 32kB SRAM
0000	RESET vektor
0010	Megszakítás vektor
FF00 – FFFF	I/O vezérlő regiszterek, 8 periféria számára
FF00 – FF1F	0: 16 x 16 bit (szavas) on chip RAM
FF20 – FF3F	1: párhuzamos bemeneti port
FF40 – FF5F	2: párhuzamos kimeneti port
FF60 – FFFF	3:=7 nincs használva

Interfészek

Vezérlőbusz: általános vezérlőjelek csoportja

- WORDN,READN, DBUSN,ACE, DMA,IRQ, stb.
- Ebből a megfelelő lokális vezérlés dekódolása
- 8 féle buszciklus:
  - (EXRAM-I/O) x (RD – WR) x (szó – byte)

Külső memóriához:

- Címbusz: A[14:1], A0, D[7:0], /CE, /WE, /OE

Perifériákhoz:

- SEL[7:0], A[3:1], A0, és a megfelelő CKE vagy BEN érvényesítő jelek

Külső memória kezelése

32kB SRAM, 15ns hozzáférési idő

Interfész: (8 bit, A0 byte szelektor)

CLK= 12MHz -> 83.3 ns

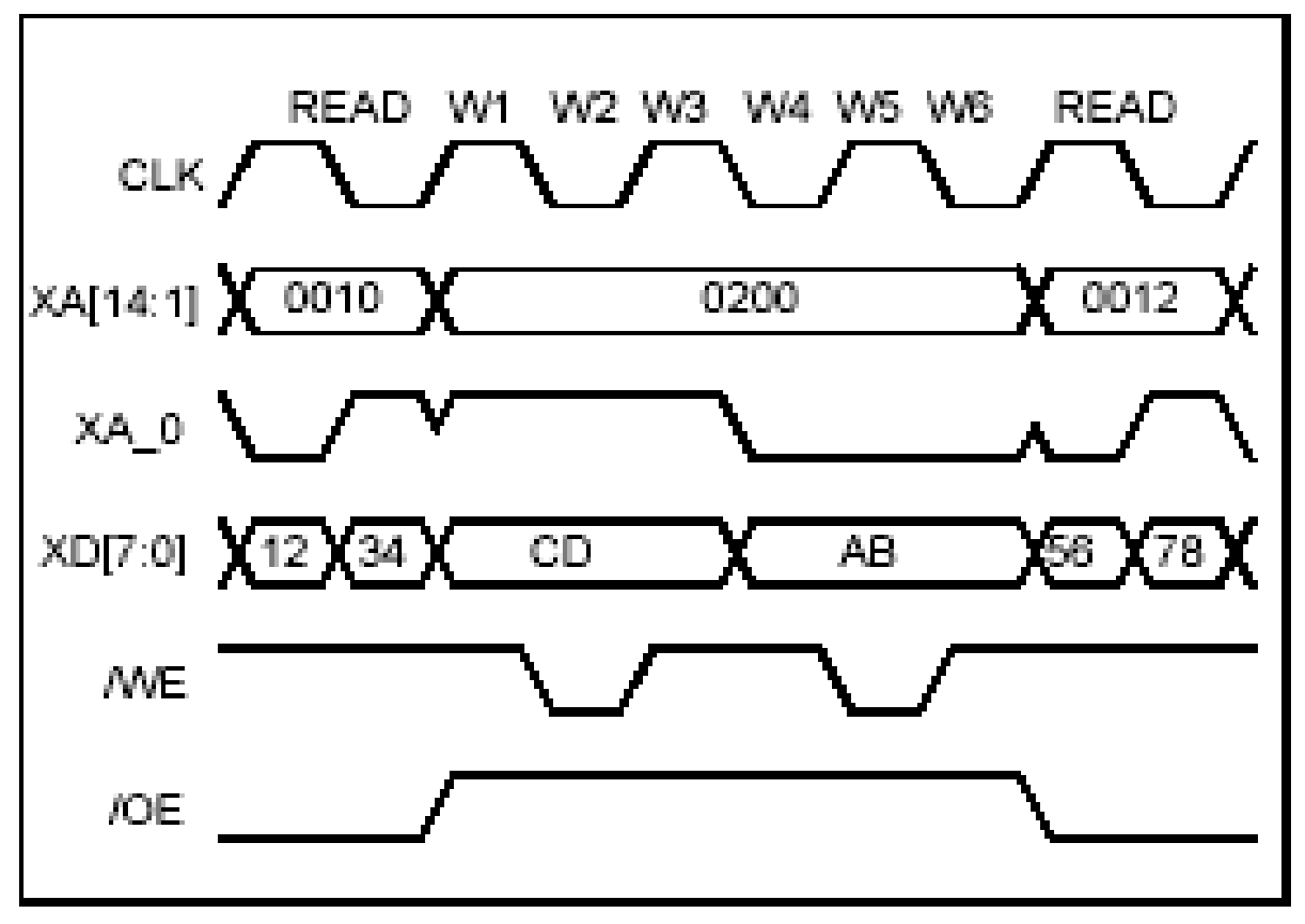
83/2 =41,6 >> 15ns + késleltetetések

Két byte olvasás 1 órajelciklus (2 fél ciklus) alatt

Két byte írás 3 órajelciklus (2x3 fél ciklus) alatt

- Az írás destruktív művelet, csak stabil vezérlőjel állapotok mellett aktivizálható

Külső memória kezelése



Összefoglalás

Az xr16 egy igazán érdekes CPU és SoC

Igényes hardver megvalósítás és szoftver környezet

Interfész lehetőségek vizsgálata fontos lenne

Új verziók: gr0000 család, konfigurálható opciókkal

Teljes infó: www.fgacpu.org

Továbbfejlesztések

Czakó Péter:

Mikrorendszer megvalósítása FPGA környezetben

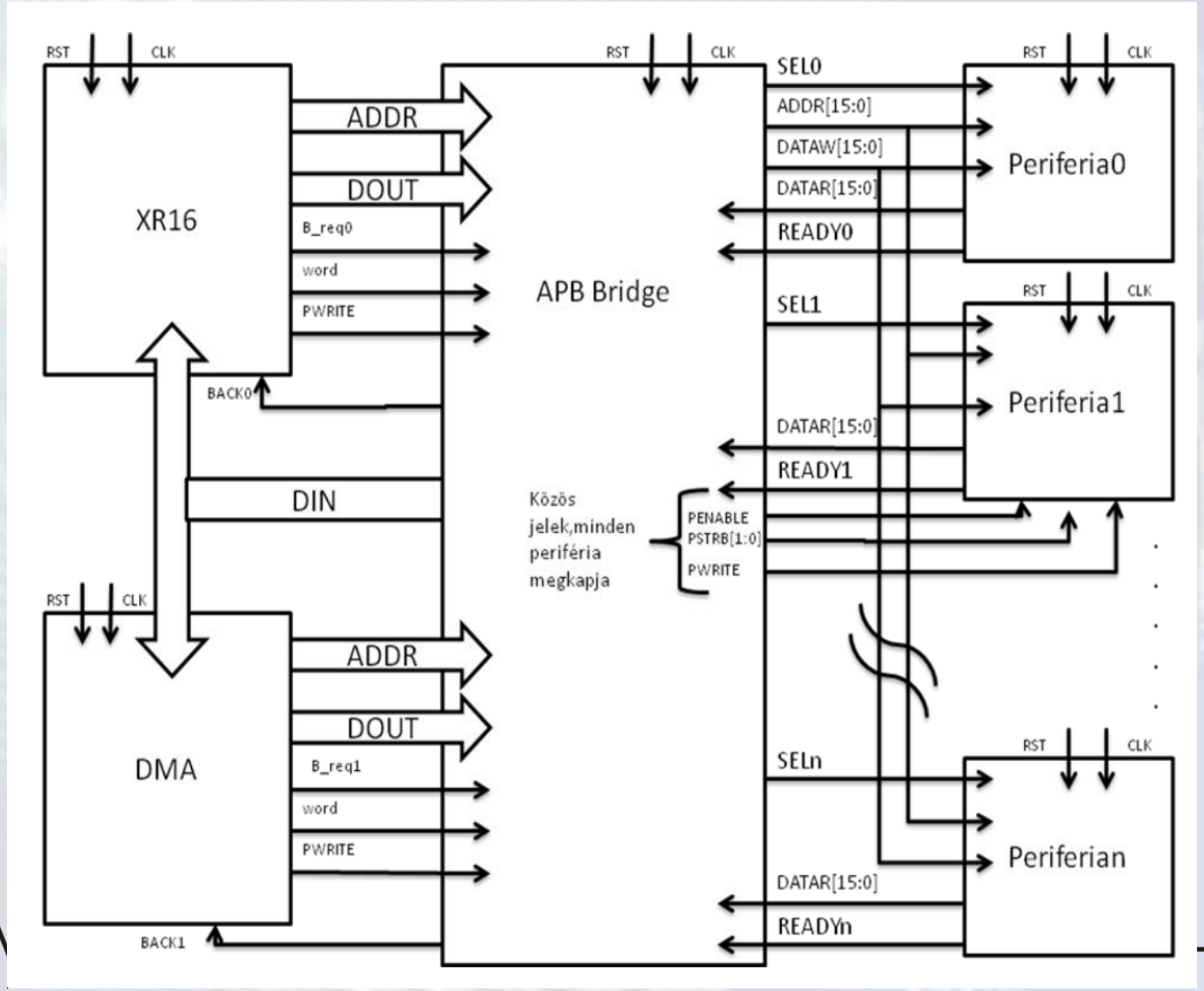
- Önlab + Diplomaterv 2006 - 2007

Berki Szabolcs:

XSOC rendszer megvalósítása APB busszal

- BSc Szakdolgozat 2011

XSOC - APB


RST
CLK
XR16
ADDR
DOUT
B_req0
word
PWRITE
BACK0
DIN
DMA
B_req1
BACK1
APB Bridge
SEL0
ADDR[15:0]
DATAW[15:0]
DATAR[15:0]
READY0
SEL1
READY1
Közös jelek,minden periféria megkapja
PENABLE
PSTRB[1:0]
PWRITE
SELn
READYn
Periferia0
Periferia1
Periferian

XSOC – APB busz működés