Digitális rendszerek és számítógép architektúrák

II. ZH típusfeladatok + megoldás

# Szinkron írási / olvasási protokoll, aszinkron handshaking

* ***Jellemezze a szinkron olvasási protokollt – mikor mi történik?***

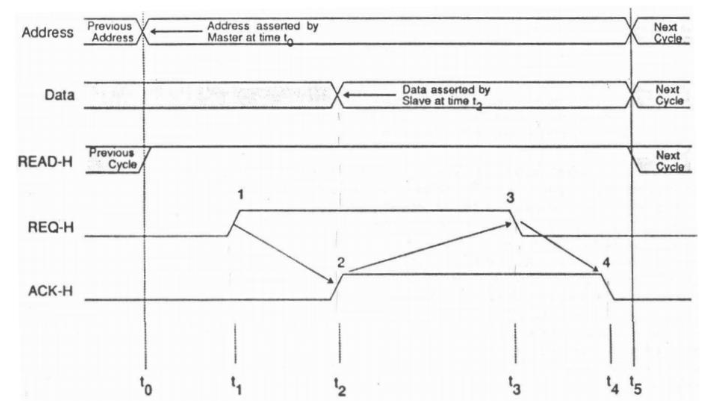
Olvasási ciklus: (CPU <- MEMÓRIA)  
n: Commander arbitrációja, CPU busz lekérése  
n+1: kérés inicializálása (CPU olvasási kérést küld, memóriához adat érkezik)  
n+2: válasz a Commandernek, memória dönt az olvasási igényről  
n+3: responder döntése (memória nyugtát küld a CPU-nak)

* ***Jellemezze a szinkron írási protokollt!***

Közös órajellel működnek a buszra csatlakozó egységek -> a műveleti időt az órajelciklus határozza meg. (Master/Slave) Az órajelet mindig a leglassabb egységhez kell igazítani.  
Írási ciklus: (CPU -> MEMÓRIA)  
n: Commander arbitrációja / kiválasztása   
n+1: adatátvitel megkezdése, memóriához adat érkezik  
n+2: memória dönt az adatok elfogadásáról, válaszol a Commandernek  
n+3: Slave/Responder nyugtát küld vissza

* ***Rajzolja le az aszinkron hand-shaking folyamatot írás esetén!***

t0: master megadja a kívánt slave címét  
t1: master vár egy bizonyos ideig mielőtt beállítja a request vonalat (master kér adatot a slavetől)  
t2: slave veszi a kérést, nyugtázza és beállítja magas szinre a nyugtázó vonalat  
t3: master megkapja az adatot a slavetől, felszabadítja a request vonalat  
t4: a slave érzékeli, hogy a master felszabadította a request vonalat, így felszabadítja a nyugtázó vonalat  
t5: a master felszabadítja a címvonalat

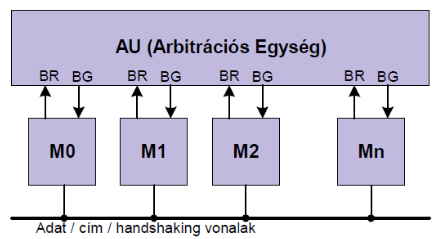


# Arbitráció fajtái

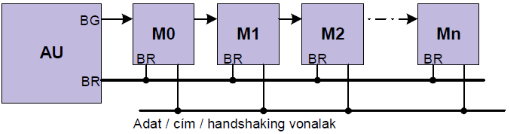
* ***Rajzolja fel és röviden ismertesse az arbitrációt és fajtáit!***

I/O művelet esetén több master is egyszerre akarja a busz irányítását. Az arbitrációs eljárás egy döntési folyamat, amely az aktuális adatátvitel befejezése előtt eldől, hogy melyik következő master adhat.  
Fajtái: párhuzamos, soros, lekérdezéses

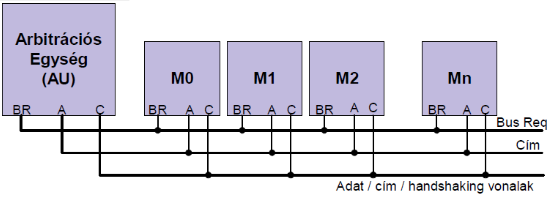
* **Párhuzamos**: Ez a leggyorsabb, legdrágább módszer és az M-ek száma korlátozott. Az arbitráció lehet: FIFO, round robin, prioritás alapú



* **Soros**: Bármennyi eszközt sorba köthetünk (felső korlát nincs). Az arbitrációs idő egyenes arányban van a sorba kapcsolt M-ek számával. BG vonal sorba van kötve. M0 legmagasabb prioritásút kérdezzük meg először igényelt-e. Ha igen, minden esetben megkapja a buszt.



* **Polling/Lekérdezéses**: Mindegyik master közös BR buszon igényelhet. Az AU minden ciklusban megnézi ki a legnagyobb prioritású igénylő. (FIFO, round robin)



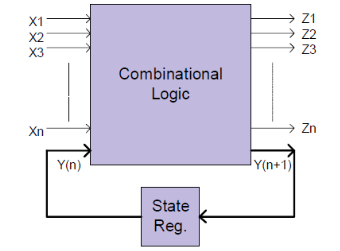
# Mealy és Moore automata modell

* ***Definiálja a Mealy automata modellt!***

A sorrendi hálózatok egyik alapmodellje. A kimenetek nem csak a pillanatnyi bemenettől, de a korábbi állapotoktól is függenek.   
Három halmaza van: bemenetek (X), kimenetek (Z), állapotok (Y)

Két leképezési szabály van e halmazok között:  
Következő állapot meghatározása:

Kimenet meghatározása:



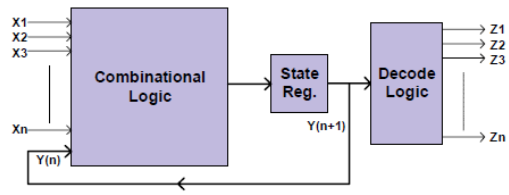
* ***Definiálja a Moore automata modellt!***

A kimenetek közvetlenül csak a pillanatnyi állapottól függenek.   
Három halmaza van: bemenetek(X), kimenetek (Z), állapotok (Y)

Két leképezési szabály van e halmazok között:

Következő állapot meghatározása:

Kimenet meghatározása:



# Mikroprogramozott vezérlőegység feladata, vertikális és horizontális mikroprogramozás (ábra, előnyök, hátrányok)

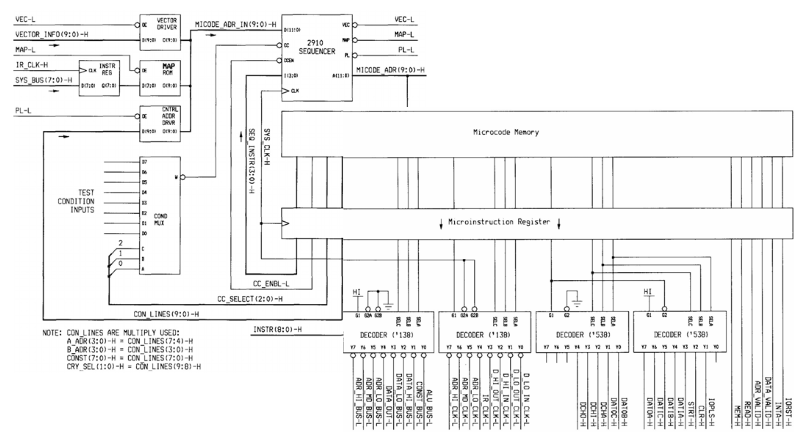
* ***Mikroprogramozott vezérlőegység feladata, vertikális mikroprogramozás (ábrák, előnyök, hátrányok).***

**Mikrokód:** gépi kódú utasításokat legalacsonyabb szintű áramköri utasítások sorozatára leképző köztes kód.

**Mikroprogramozott vezérlőegység**: az adatútvonal vezérlési pontjait memóriából (ROM) kiolvasott vertikális vagy horizontális utasításokkal állítják be.

**Vertikális mikrokód:   
 -** nem a sebességen van a hangsúly, hanem a takarékosságon az erőforrásokkal.  
 - egy időben csak a szükséges biteket kezeljük, egymástól nem teljesen függetlenül, mivel egyszerre csak egyet állítunk be  
 - a jelet dekódolni kell, mely több időt is igénybe vehet, ha a dekódolásnál több mikro utasítás szükségeltetik a memóriát „vertikális” irányban növeljük.  
 - a kiválasztott biteket megpróbáljuk minimális számú vonalon keresztül továbbítani

**Előny**: Takarékos az erőforrásokkal  
**Hátrány**: Lassú

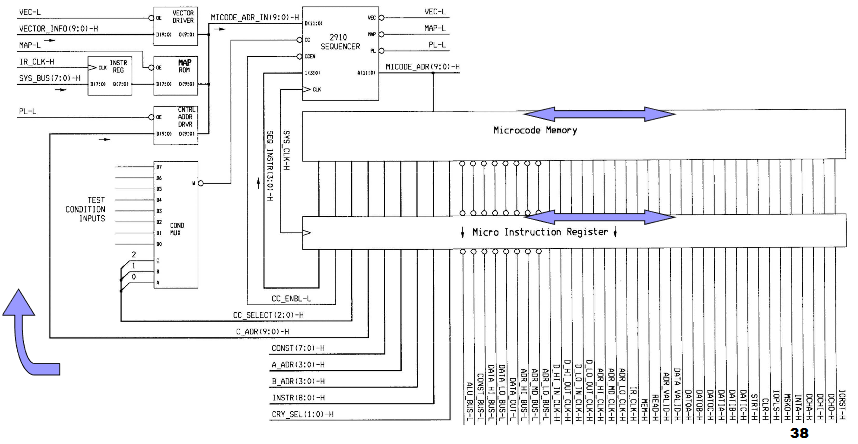


*Megj: Asszem ez az ábra itt kurvára nem vicc és mindjárt sírok.*

* ***Mikroprogramozott vezérlőegység feladata, horizontális mikroprogramozás (ábrák, előnyök, hátrányok).***

**Horizontális mikrokód:**  
- Minden egyes vezérlőjelhez saját vonalat rendelünk, ezáltal horizontálisan nő a mikrokód. Minél több funkciót valósítunk meg vezérlőjelekkel, annál szélesebb lesz a kód.  
- Ez e leggyorsabb mikrokódos technika, mert minden bit független egymástól, és egy mikrokóddal többszörös utasítás is megadható.  
- Nagy az erőforrás igény és fogyasztás

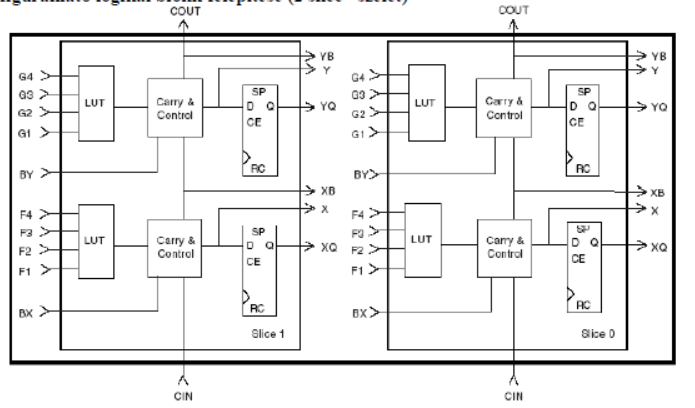
Előny: Rendkívül gyors  
Hátrány: Magas erőforrás igény



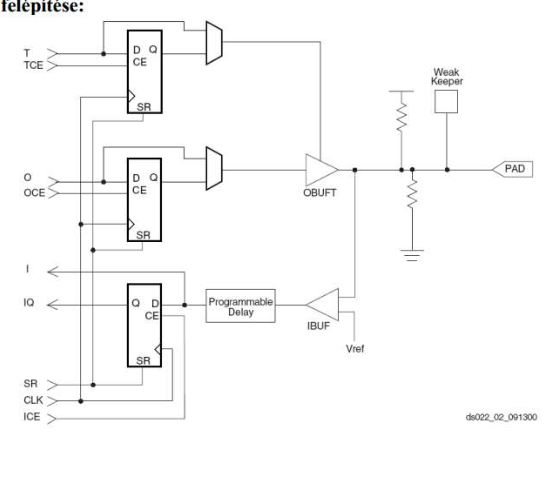
# XILINX FPGA felépítése

* ***Rajzolja fel a XILINX FPGA felépítését (CLB és I/O)!***

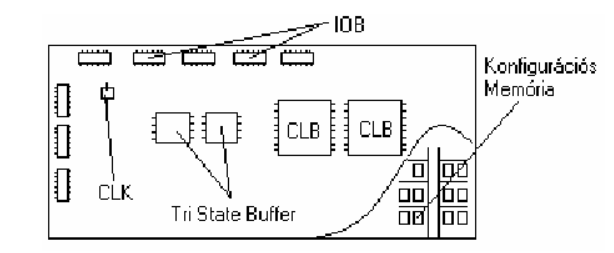
A XILINX FPGA áramkörök alapvető építőelemei:  
**CLB: Konfigurálható Logikai Blokk.**   
 - szükséges logikai kapcsolatok megvalósítása egy logikai tömbben  
 - tartalmaz 2 db. D Flip-Flopot és 2 db. független 4 bemenetű LUT-t

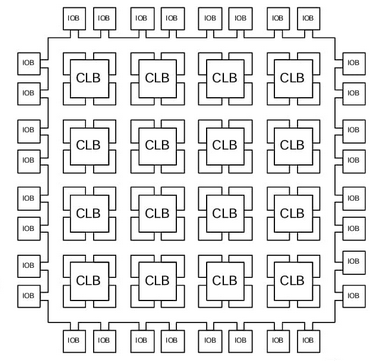


**IOB: Programozható Be és Kimeneti blokk**   
 - kapcsolat a belső vonalak és a tokozás lábai között  
 - belső puffere van, nem kell az adatokat külső lábakon tartani  
 - kiement lehet negált vagy ponált



**A XILINX FPGA általános felépítése:**





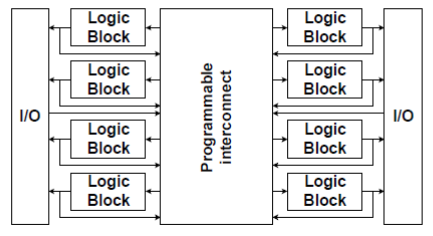
# Vezérlő egység programozható makrocellákkal

* ***Rajzolja fel egy-egy PROM, PLA és PAL belső felépítését!***
* ***Vezérlőegység programozható makrocellákkal (fajta, ábrák, jellemző tulajdonságok)***

**PROM**: AND fix, OR programozható. OR kapuk kimeneténél tároló regiszterek, amik visszacsatolódnak a bemenetekre.  
**PLA**: AND és OR programozhatók. OR kapuknál D-tárolók, amik visszacsatolhatnak a bemenetekre.  
**PAL**: AND kapuk programozhatók, OR kapuk fixek. Gyorsabb, mint a PLA. Kimeneten D tárolók, visszacsatolhatnak

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **PROM** | **PLA** | **PAL** |

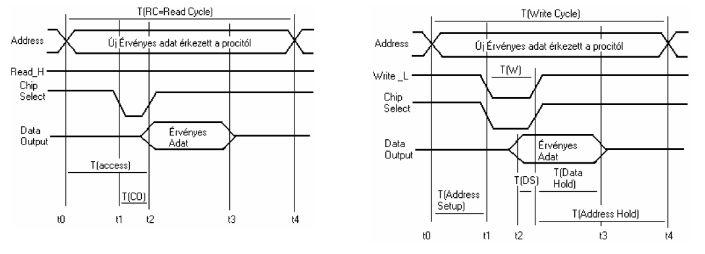
**CPLD**: Logikai blokkban PAL/PLA, regiszterek.



# Dinamikus és statikus memória

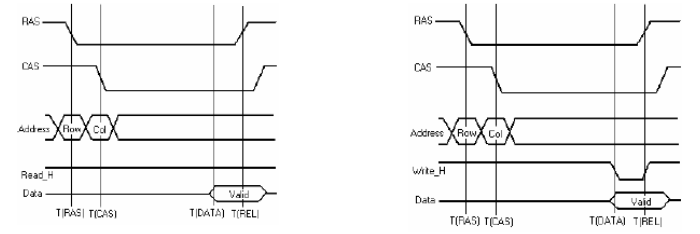
* ***Ismertesse a statikus memória írási olvasási folyamatát! (idődiagram)***

Az olvasási és írási folyamat (ebben a sorrendben):



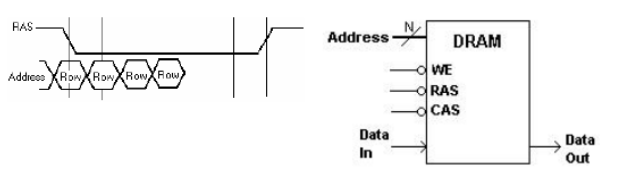
* ***Ismertesse a dinamikus memória írási olvasási folyamatát! (idődiagram)***

Az olvasási és írási folyamat (ebben a sorrendben):



* ***Ismertesse a dinamikus memória frissítési folyamatát (idődiagram)***

Kb. 2-10 milliszekundumomként frissíteni kell, mivel kisméretű kondenzátoron tároljuk az információt, melynek feszültsége idővel exponenciálisan csökken (kisül).   
Egyetlen CAS oszlopcímet adunk ki, majd RAS-sal az összes sorcímet, hogy az összes cella frissítésre kerüljön.



# Minimum hány lábra van szükség…?

Általánosan: 1db R/W +1 db VDD + 1db GND + 1DB CS miatt 4 láb, továbbá amennyi a második tag annyi I/O és log2elsőszám.

* ***Minimum hány lábra van szükség egy 64x32-es 1 CS-sel és közös I/O-val rendelkező SRAM megvalósításához?***

32 db I/O + 6db cím + 1db R/W +1 db VDD + 1db GND + 1DB CS = 42

* ***Minimum hány lábra van szükség egy 128x16-os 1 CS-sel és közös I/O-val rendelkező SRAM megvalósításához?***

16 db I/O + 7db cím + 1db R/W +1 db VDD + 1db GND + 1DB CS = 27

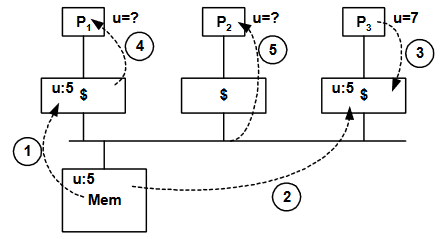
* ***Minimum hány lába van egy 1024x4-es 1 CS-sel és közös I/O-val rendelkező RAM megvalósításához?***

Mivel CS -> SRAM-ról van szó.  
4db I/O + 10db cím + 1db R/W +1 db VDD + 1db GND + 1db CS = 18

# Cache

* ***Mi a cache koherencia probléma? Mutasson rá egy egyszerű példát!***

Osztott cache használat esetén léphet fel, mikor a P processzorok saját $ cache memóriával rendelkeznek. A probléma abból adódhat, mikor ugyanazon osztott memória blokk tartalma megjelenik egy vagy több processzor saját cache memóriájában. Például megtörténik egy írási tranzakció a főmemóriában, a többi processzor még a régi cachebeli tartalommal (másolat) dolgozik, tehát nem aktualizálódnak a cache memóriák értékei.  
Példa:



* ***Definiálja a cache miss és a cache hit fogalmát!***

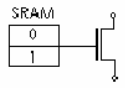
**Cache hit**: Ha a processzor olyan adatot igényel, mely a cache-ben megtalálható, akkor találatról, vagy cache hitről beszélünk. A találatot a cachevezérlő azzal állapítja meg, hogy a bemásolt blokkokhoz tartozó címrészek alapják valamelyik cache blokkban benne van e a processzor által igényelt adat főtárbeli címe.

**Cache miss**: Ha a processzor által igényelt adat nincs meg a cache-ben, ezt tévesztésnek vagy cache miss-nek nevezzük.

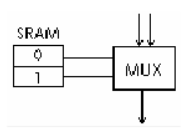
# VLSI

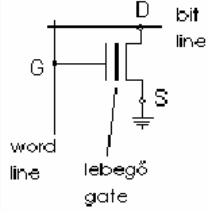
* ***Ismertesse hogyan programozható egy VLSI!***

**SRAM esetén:**

* végtelen sokszor újraprogramozható
* táp kikapcsolása után az SRAM elveszti a tartalmát
* bekapcsoláskor a programot be kell tölteni, fel kell programozni
* a SRAM cellára egy áteresztő tranzisztor van csatolva. A tranzisztor kinyit vagy lezár. Az SRAM értéke, ami egy bit, letölthető.
* 1 bit tárolása az SRAM-ban (min. 6 tranzisztor)
* sok tranzisztor, nagy méret
* SRAM-ot nem kell frissíteni

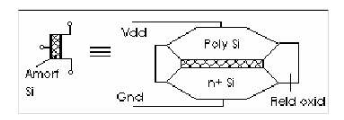
**Multiplexer esetén:**

* A SRAM-ban tárolt bitet használjuk a multiplexer vonalának kiválasztásához

**Floating gate:**

* töltéseket viszünk fel a word-line felől lebegő gate-re, kinyit a tranzisztor (a control gate befolyásolja a működését)
* UV fénnyel többször törölhető
* kikapcsolás után is megőrzi a tartalmát

**Antifuse:**

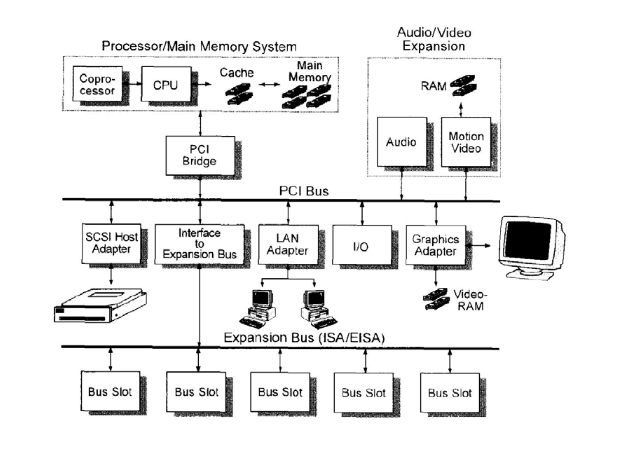
* egyszer programozható
* kis méretű
* nehéz előállítani

**EPROM/EEPROM/FLASH**

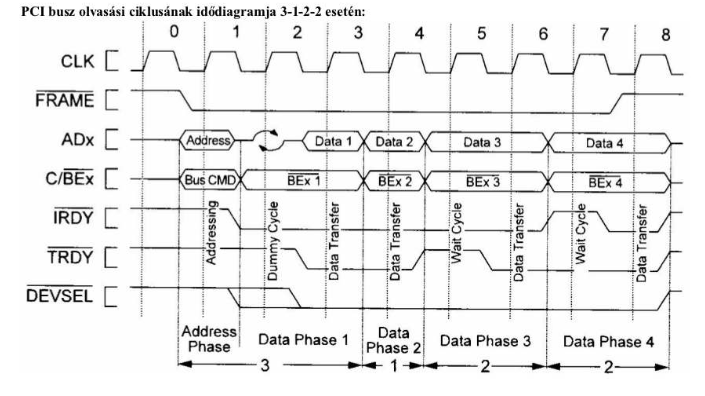
* Floating gate technológia
* megőrzi a tartalmát
* végtelen sokszor írható és törölhető (EEPROM, FLASH)
* UV fénnyel törölhet (EPROM)
* drága

# PCI

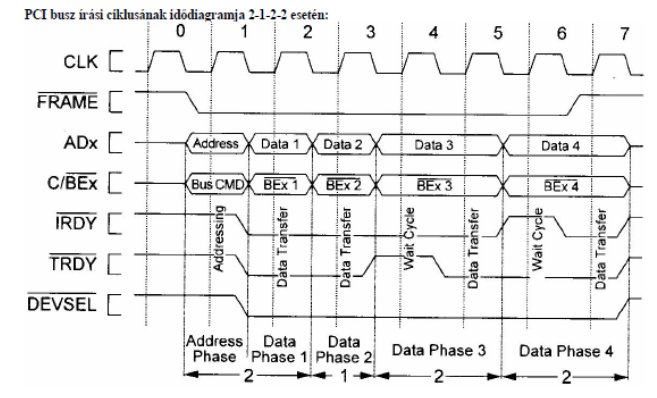
* ***Rajzolja fel egy PCI-Express buszos számítógép blokkdiagramját!***



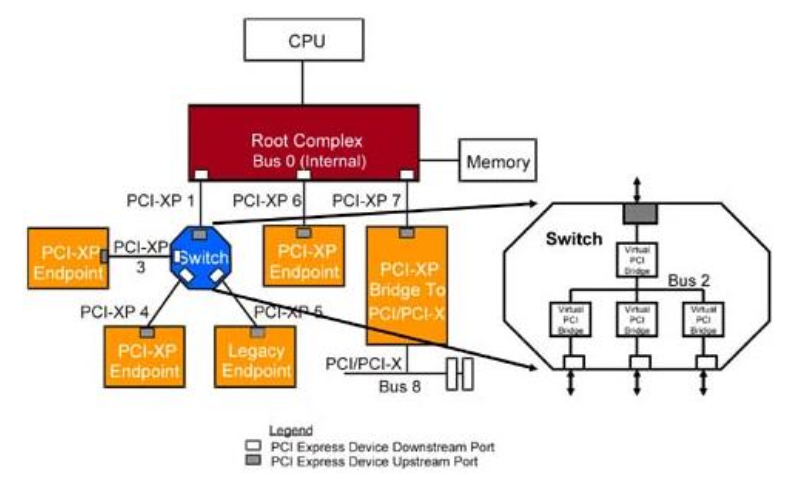
* ***Rajzolja fel egy PCI burst read tranzakció időgramját ideális esetben!***



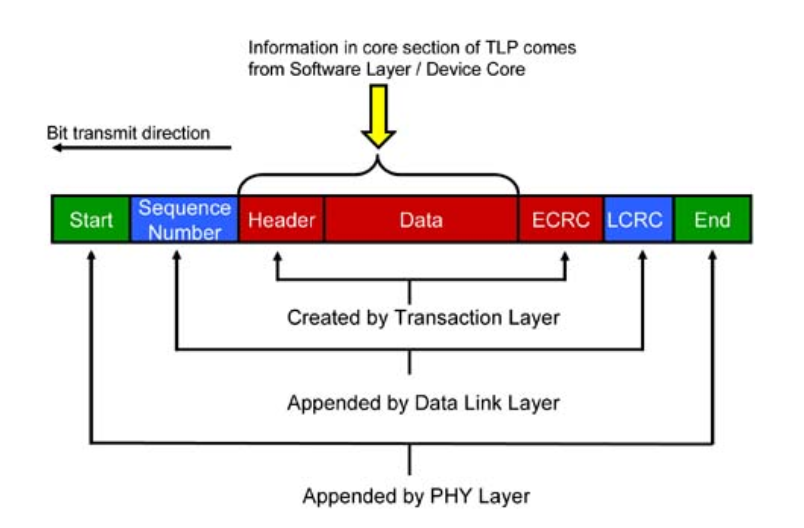
* ***Rajzolja fel egy PCI burst write tranzakció idődiagramját, és a diagramon mutasson példát a várakozási ciklusra mind a két egység részéről!***



* ***Rajzolja fel egy PCI-Express felépítését!***

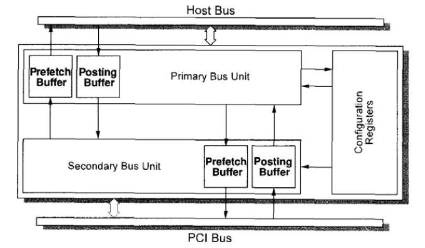


* ***Rajzolja fel egy PCI-Express Transaction Layer Packet felépítését!***



* ***Milyen átviteli technológiát használnak a PCI busz jelei és milyen jelcsoportok találhatók a PCI csatlakozón, továbbá milyen a csatlakozó felépítése?***

Szinkron, PCI 32 bites multiplexált adat/cím jeleket alkalmaz. A csatlakozó 188 lábú oldalanként 94-94 lábbal. Sok GND található a zavarások elkerülése végett. A tápról jön 12V, 5V, 3.3V-os jel is.



* ***Rajzoljon fel egy PCI burst write tranzakció idődiagramját és a diagramon mutasson példát a várakozási ciklusra mindkét egység részéről!***

¯\\_(ツ)\_/¯

* ***Rajzoljon fel egy PCI-Express szerver alapú rendszer blokkdiagramját!***

¯\\_(ツ)\_/¯

# Lapozás, szegmentálás

* ***Rajzoljon fel egy lapozásos virtuális memória kezelő áramkört! Szükséges paraméterek: virtuális cím: 32 bit, lapméret: 4k, fizikai cím: 24 bit.***

¯\\_(ツ)\_/¯

* ***Rajzoljon fel egy szegmentált virtuális memória kezelő áramkört memóriavédelemmel! Szükséges paraméterek virtuális cím: 32 bit, szegmensek száma: 16, fizikai cím: 24 bit.***

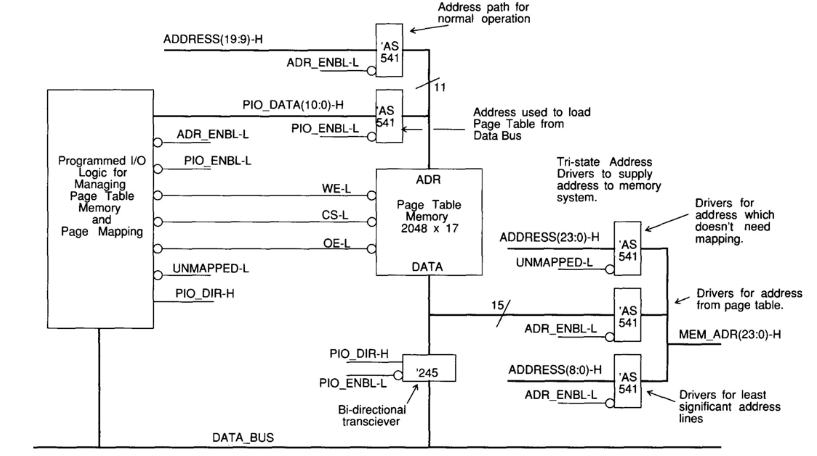
¯\\_(ツ)\_/¯

* ***Hasonlítsa össze a szegmentálást és a lapozást!***

**Lap:**   
 - a program és a memória is fizikailag egyenlő méretű darabokra van osztva.   
 - a fő program, szubrutinok, globális és lokális adatok, verem mind egy-egy azonos méretű lapnak felelnek meg a memóriában  
 - a módszer gazdaságtalan, mert kis adatot is egy nagy méretű lapba kell betölteni.  
 - fizikai cím = Lap báziscíme + Lap eltolás.  
 - a + konkatenációt (összefűzést) jelent. Ezáltal gyorsabb a fizikai cím képzése, mert nem kell összeadandót használni. A hexadecimális címek egy-egy számjegyét 4 bites bináris számmá kódoljuk és összefűzzük  
 - az OS minden futó processhez hozzárendel egy laptáblát  
 - a laptáblából kiolvassuk a lap számát, címét és elérési információit (R/W/X: read, write, execute)  
 - a lapok kis egységek, nagy laptáblát használunk, célszerű memóriában tartani  
 - a lapok mérete általában kisebb mint a szegmensek mérete  
ELŐNY: gyors , HÁTRÁNY: gazdaságtalan

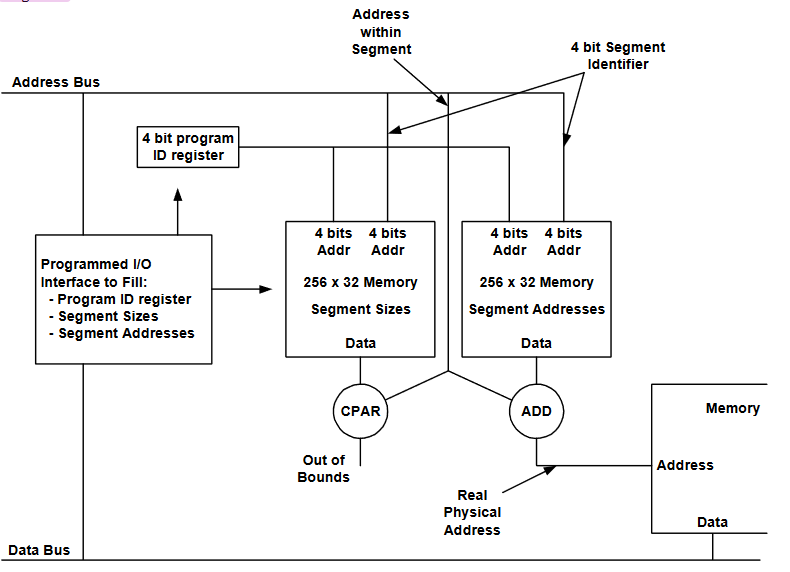
**Szegmentálás:**  
 - szegmens: a program változó méretű logikai egységekre van feldarabolva. A program négy fő szegmense funkciójuk szerint:  
 - 0. szegmens: főprogramok (olvasható, végrehajtható),  
 - 1. szegmens: szubrutinok  
 - 2. szegmens: csak olvasható adatok  
 - 3. szegmens: olvasható / írható adatok  
 - a program a memóriát közvetlenül nem érheti el, csak a szegmensen keresztül.  
- fizikai cím = szegmens bázis címe + szegmensen belüli eltolás. A + itt összeadás!  
- HW-s megvalósítás kell az összeadó miatt, és egy összehasonlítás, hogy ne címezzünk túl a címtartományon. hiba esetén megszakítunk.  
- szegmenstábla: a szegmenstáblából kiolvassuk a szegmens számát, majd a hozzá tartozó szegmens hosszát, címet, továbbá az elérési információt. Kezelése az OS feladata

* ***Jellemezze a lapozásos virtuális memória kezelést – rajz + jellemző paraméterek tárolása!***



Részletesebb leírás: korábbi kérdésnél.

* ***Jellemezze a szegmentált virtuális memória kezelést – rajz + jellemző paraméterek tárolása!***



Részletesebb leírás: korábbi kérdésnél.

* ***Rajzoljon fel egy lapozásos virtuális memóriakezelő áramkört! Szükséges paraméterek: virtuális cím: 64 bit, lapméret: 4k, fizikai cím: 48 bit.***

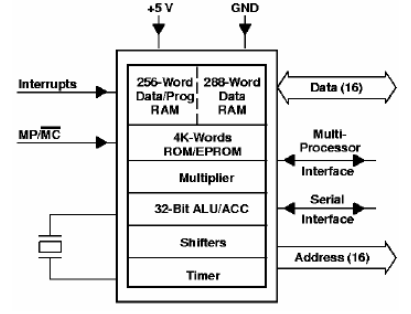
¯\\_(ツ)\_/¯

* ***Rajzoljon fel egy szegmentált virtuális memória kezelő áramkört memóriavédelemmel! Szükséges paraméterek: virtuális cím: 48 bit, szegmensek száma: 16, programok száma: 32, fizikai cím: 32 bit.***

¯\\_(ツ)\_/¯

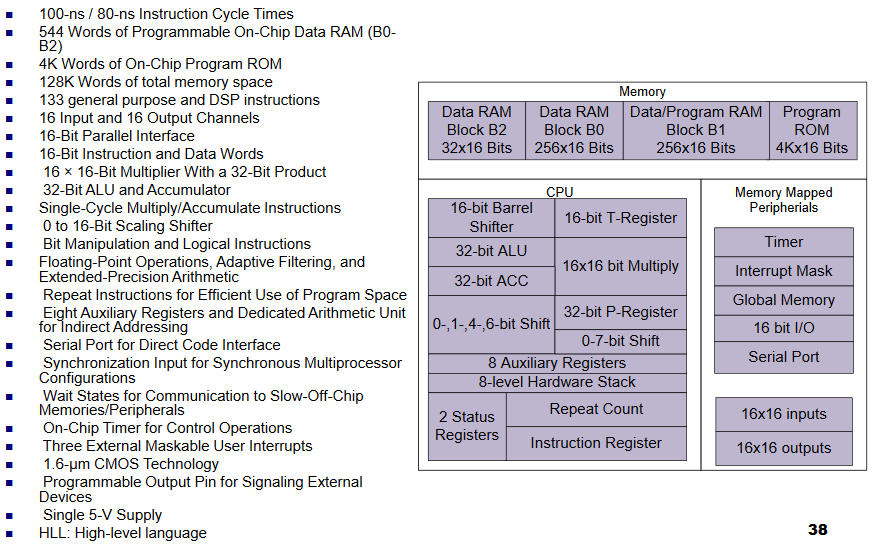
# DSP

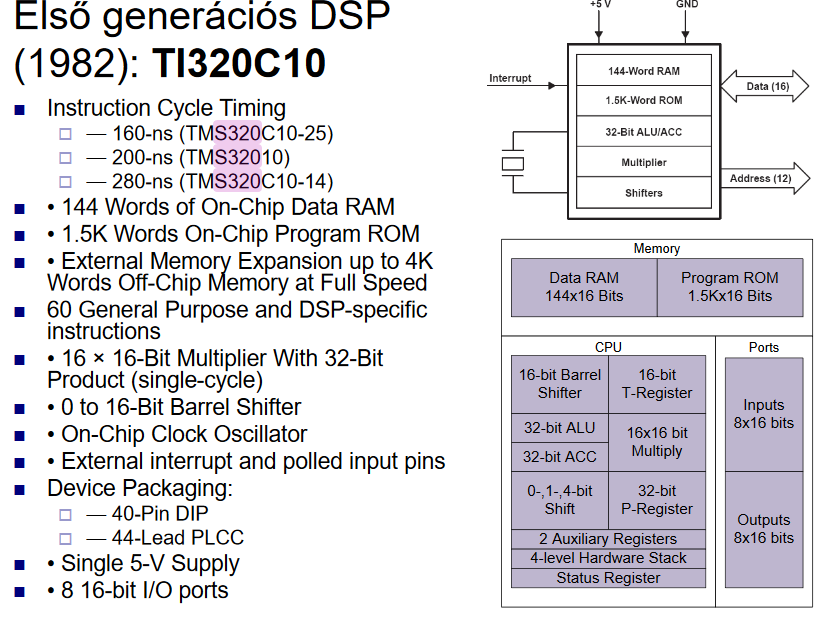
* ***Rajzoljon fel egy fixpontos DSP architektúrát és adja meg néhány jellemző tulajdonságát!***



Tulajdonságai: 40MHz frekvencia, fixpontos, 16 bites, 68 lábú processzor. CMOS technolgóiával készült -> kis teljesítménydisszipáció, 80-100 ns utasításvérehajtási ciklusidő. 1 ciklus alatt szorzás/tárolás, 5V tápellátás, módosított Harvard architektúra, két adatmemóriája van, melyből egyik variálható, 32-bites ALU, blokkos adatátvite, rugalmas, nagysebességű, processzort9mbbe szervezhető chip, párhuzamos architektúra, hatékony utasítás készlet, HW-esen implementáált funkciók, magas szintű C nyelven programozható

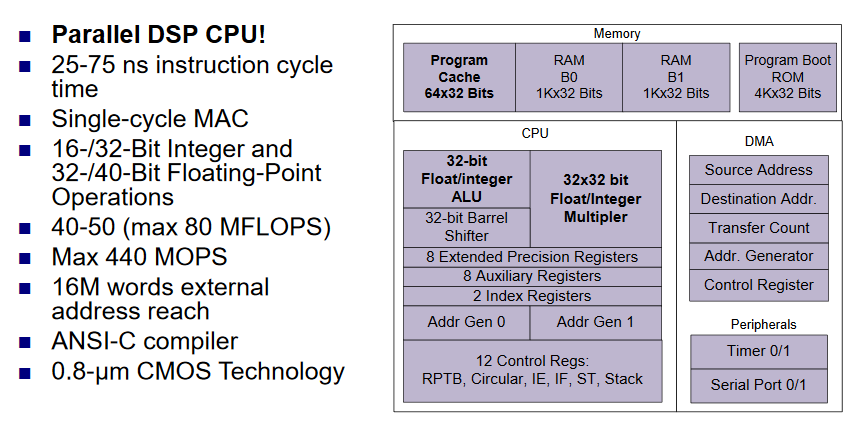
**Talán?**





* ***Rajzoljon fel egy lebegőpontos DSP architektúrát és adja meg néhány jellemző tulajdonságát!***

**Talán?**

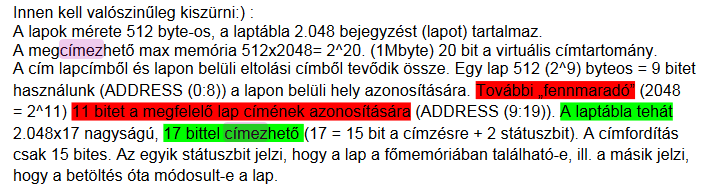


# SCSI

* Határozza meg az SCSI kommunikációs lépéseket és rajzolja fel a kommunikáció folyamatát!

1. command
2. data
3. message
4. status

# Memória címzés



* ***Mekkora memória címezhető meg 25 bittel?***
* ***Mekkora memória címezhető meg 21 bittel?***
* ***Mekkora memória címezhető meg 29 bittel?***

# Egyéb

* ***Hogyan használható egy memóriából több független program?***

Minden programnak saját logikai címtartománya van, amellyel hivatkozni lehet rá.