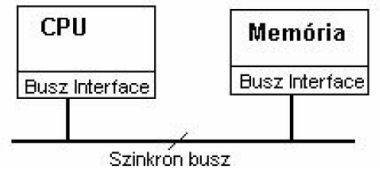
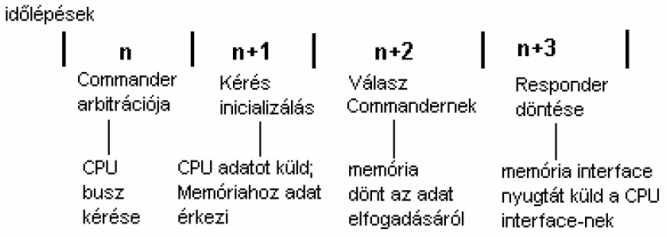
2018/A

JELLEMEZZE A SZINKRON ÍRÁSI PROTOKOLLT (LÉPÉSEK + FELADATOK)! (2P)

\+ Közös órajellel működnek a buszra csatlakozó egységek. Tehát a műveleti időt az órajelciklus határozza meg. Mivel nincs szükség párbeszédre (pl. handshaking), gyorsabb az aszinkron működésénél. Jel: Master=Commander, a Slave=Responder.
\- Az órajelet mindig a leglassabb (legtávolabb lévő) egységhez kell igazítani.



Írási ciklus: (CPU ⇒ MEM)



Szinkron adatátvitel 4 fő lépése:

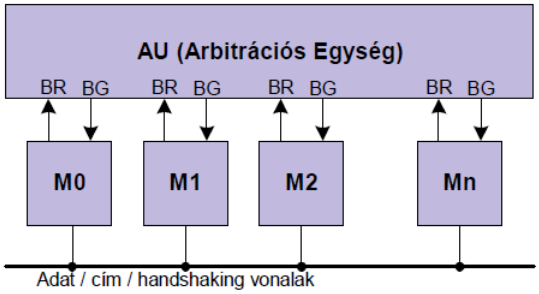
1. Commander arbitrációja, kiválasztása
2. átvitel megkezdése (de még nem fogad)
3. válasz a kérésre, döntéshozás
4. nyugta, Responder veszi az adatot

RAJZOLJA FEL ÉS RÖVIDEN ISMERTESSE AZ ARBITRÁCIÓT ÉS FAJTÁIT (ÁBRÁK). (2P)

- I/O művelet esetén **több Master is meg akarja szerezni** a busz irányítását
- **előre definiált algoritmusok** alapján → melyik **Master végzi a következő átvitelt**
- **arbitrációs eljárás = döntési folyamat** – aktuális adatátvitellel párhuzamosan
- adatátvitel előtt eldől, melyik következő Master adhat
- **BR**: Bus Request (busz kérése) – Master bocsátja ki
- **BG**: Bus Grant (igénylés elfogadása, engedélyezés) – központi **AU** (Arbitration Unit) bocsájtja ki
- **M_1,…M_n :** Masterek

Egy tetszőleges I/O művelet esetén (aszinkron v. szinkron buszos átvitel) több Master (commander) egység is meg akarja szerezni egyszerre a busz irányítását. Különböző előredefiniált algoritmusok segítségével egyértelműen azonosítható, hogy a „versenyhelyzetben" a következő átvitelt (a következő ciklusban) melyik Master fogja megvalósítani. Az *arbitrációs eljárás* egy döntési folyamat (mechanizmus), amely az aktuális adatátvitellel párhuzamosan zajlik le, és még az adatátvitel befejezése előtt eldől, hogy melyik következő master adhat. A Mastereket M_1...M_n-el jelöljük, a BR: Bus Request (busz kérése, igénylése) a Master által, míg a BG: Bus Grant (igénylés elfogadása, engedélyezés) jelet a központi AU: Arbitration Unit (arbitrációs egység) bocsátja ki.
Az arbitrációnak 3 típusa van: párhuzamos, soros (daisy chain), lekérdezéses (polling).

a.) *Párhuzamos:*

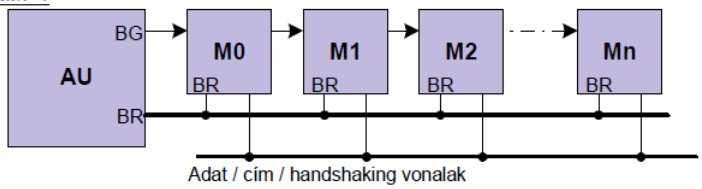


Ez a **leggyorsabb**, **legdrágább** módszer és az M-ek száma korlátozott.

Az **arbitráció lehet**:

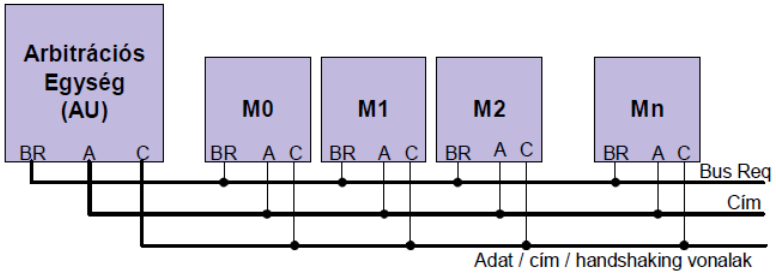
- FIFO
- round robin
- prioritásos alapú

b.) *Soros „Daisy Chain”:*



- **bármennyi** eszközt **sorba** köthetünk (nincs felső korlát)
- **arbitrációs idő egyenes arányban** van a **sorba kapcsolt M-ek** számával
- **BG vonal sorba** van kötve
- **M_0 legmagasabb prioritású**t kérdezzük meg **először**, igényelt-e → Ha igen, minden esetben **megkapja a buszt**.

c.) *Polling / Lekérdezéses technika:*



- mindegyik Master **közös BR buszon igényelhet**
- az **AU minden ciklusban** megnézi, ki a **legnagyobb prioritású** igénylő (FIFO, round robin)

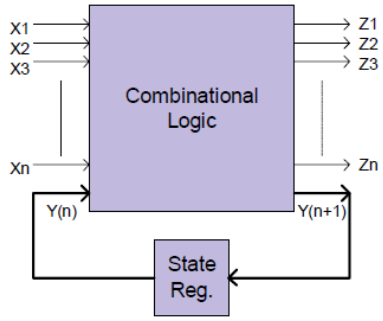
DEFINIÁLJA A MEALY AUTOMATA MODELLT (HALMAZOK ÉS LEKÉPZÉSEK)! (2P)

A sorrendi hálózatok egyik alapmodellje. (A KH. = kombinációs hálózatok esetén a kimenetet csak a bemenetek függvényeként kaptuk meg). Azonban a valóságban minden egyes eszköznek van bizonyos minimális késleltetése, és a kimeneten az eredmény véges időn belül jelenik meg, vagy a korábbi értékek visszacsatolódnak a bemenetre. A sorrendi hálózatokat megvalósító vezérlő egységeknél kimenetek nem csak a bemenetek pillanatnyi, hanem a korábbi állapotoktól is függenek. (N memóriaelem esetén 2^N lehetséges állapotot tud a rendszer elfogadni.)
Három halmaza van: X – a bemenetek, Z – a kimenetek, Y – az állapotok halmaza. Visszacsatolni az állapotregisztert a késleltetés miatt kell.
Két leképezési szabály van a halmazok között:

$\delta(X_n, Y_n) \rightarrow Y_{n+1}$: következő állapot meghatározása

$\mu(X_n, Y_n) \rightarrow Z_n$: kimenet meghatározása

Azonban problémák merülhetnek fel az állapotok és bemenetek közötti szinkronizáció hiánya miatt (változó hosszúságú kimenetet kaphatunk). Ezért alkalmazzuk legtöbb esetben a következő Moore féle automata modellt.



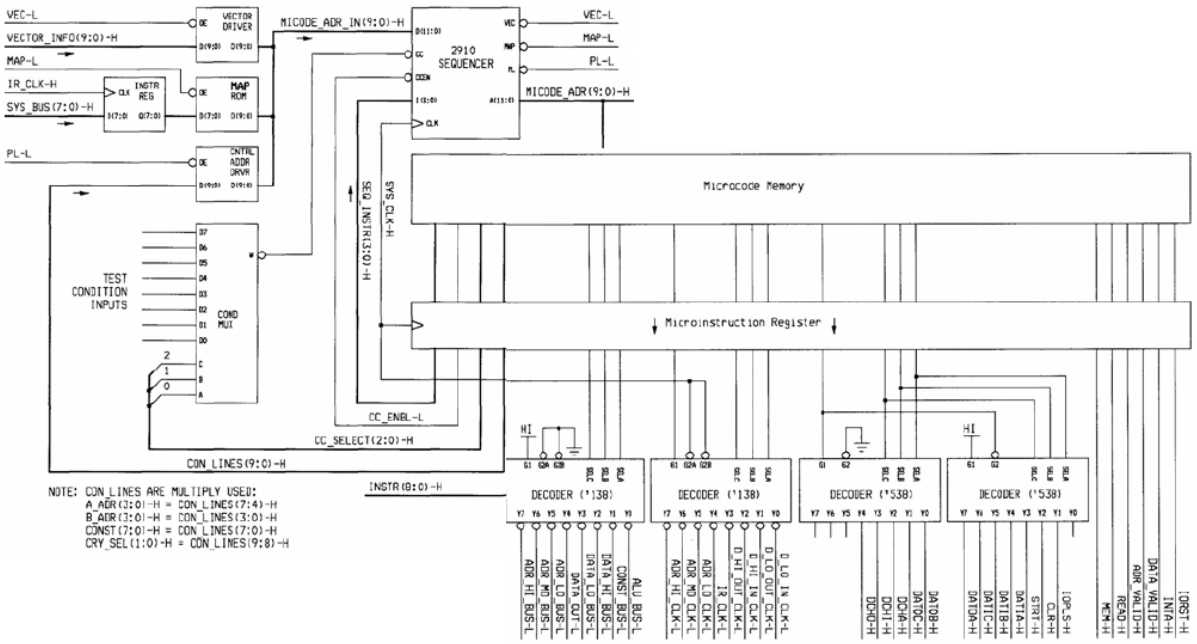
MIKROPROGRAMOZOTT VEZÉRLŐEGYSÉG FELADATA, VERTIKÁLIS MIKROPROGRAMOZÁS (ÁBRA, ELŐNYÖK, HÁTRÁNYOK). (3P)

FELADATA:

- memóriában lévő **gépi kódú program utasításainak értelmezése**
- **részműveletekre bontása**
- ezek alapján: **funkcionális egységek vezérlése**
- **ütemező-vezérlő** végzi a **vezérlő jelek sorrendi előállítását** → **mikroprogramozott**: **adatútvonal vezérlési pontjai**t a memóriából **mikrokódú utasításokkal** állítjuk be

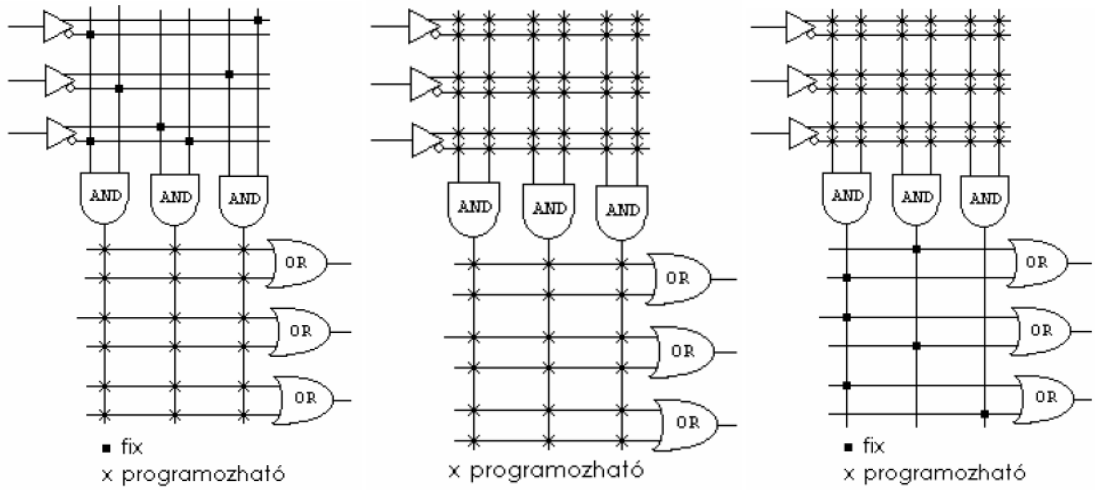
Vertikális mikrokód:

- **takarékosabb** (fogyasztás, mikrokódban bitek számával) → **lassabb**
- egy időben **csak korlátozott** számú **bit**eket kezel - egymástól nem teljesen függetlenül → egyszerre csak egyiket állítjuk be
- jelek dekódolása → **több idő**t vesz igénybe – N bites busz $\log_2(N)$ számú bittel
- műveletek **párhuzamos végrehajtása korlátozott**
- több mikroutasítás → vertikális irányban növeljük a memóriát



RAJZOLJA FEL EGY-EGY PROM, PLA ÉS PAL BELSŐ FELÉPÍTÉSÉT. (2P)

PROM **PLA** **PAL**



- **PROM**: AND fix, OR programozható (EEPROM, EPROM esetén) . 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.

MINIMUM HÁNY LÁBRA VAN SZÜKSÉG EGY 64KX32-ES 1 CS-SEL ÉS KÖZÖS I/O-VAL RENDELKEZŐ SRAM MEGVALÓSÍTÁSÁHOZ? (2P)

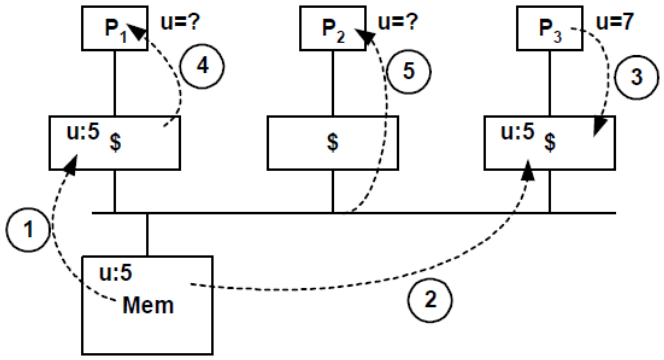
- 32 db I/O
- 16 db cím [2^{16}=64*1024=64k]
- 1 db R/W
- 1 db VDD (betáp)
- 1 db GND (föld)
- 1 db CS

Összesen: 32+16+4= ~52

MI A CACHE KOHERENCIA PROBLÉMA? MUTASSON RÁ EGY EGYSZERŰ PÉLDÁT! (2P)

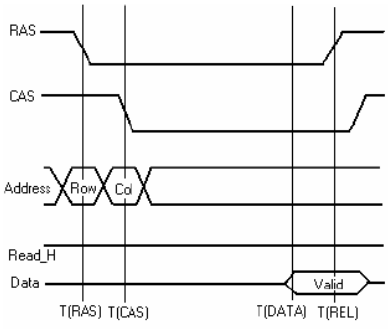
- ha a P processzorok saját $ cache memóriával rendelkeznek
- ugyanazon osztott memória blokk tartalma megjelenik egy vagy több processzor saját cache memóriájában
- történik egy írási tranzakció, a többi processzor pedig még a régi cache-beli tartalommal dolgozik → nem aktualizálódnak a cache memóriák értékei

Osztott cache használat esetén léphet fel, amikor a P processzorok saját $ cache memóriával rendelkeznek. Probléma abból adódhat, amikor ugyanazon osztott memória blokk (u location) tartalma megjelenik egy vagy több processzor saját cache memóriájában, és például történik egy írási tranzakció (egyik processzor módosítja a főmemória tartalmát), a többi processzor pedig még a régi cache-beli tartalommal (másolat) dolgozik, tehát nem aktualizálódnak a cache memóriák értékei.
Példa:



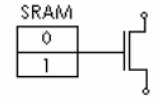
ISMERTESSE A DRAM (DINAMIKUS MEMÓRIA) OLVASÁSI FOLYAMATÁT (IDŐDIAGRAM). (2P)

Olvasási ciklus:
A DRAM-oknak több címvonalra van szüksége, mivel a címek két részre, egy sor- és egy oszlop-azonosítóra (RAS-CAS) vannak felosztva. Miután a RAS jelet alacsony szintre állítja be (lefutó élre vezérelt) a sorcím megjelenik a címvonalon. Ezt kitartja rövid ideig, majd a CAS jelet állítja be alacsony szintre, és az oszlopcím jelenik meg. Ezután válik elérhetővé a memória (setup time). Majd a memória elérési idejétől (T access time) függően kis idő múlva az adat érvényessé válik (Valid). Amikor az RAS felszabadul (T (REL)) a kimeneten az adat visszatér a háromállapotú (tri-state) feltételhez. Read_H végig magas aktív, hiszen olvasási ciklust hajt végre.



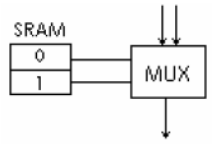
ISMERTESSE, HOGY HOGYAN PROGRAMOZHATÓ EGY PROGRAMOZHATÓ VLSI (RAJZ + ELŐNYÖK ÉS HÁTRÁNYOK, JELLEMZŐ TULAJDONSÁGOK). (3P)

Hogyan programozhatók a VLSI alkatrészek? Programozási technikák a következők:

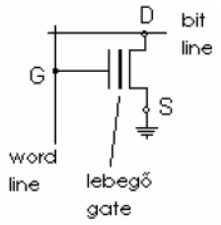


a.) SRAM-os: az SRAM cellára egy áteresztő tranzisztor van csatolva. A tranzisztor vagy kinyit (vezet), vagy lezár. Az SRAM értéke, ami egy bitet tárol ('0' vagy '1') letölthető. Összeköttetéseket, vagy MUX-ok állását is eltárolja.
Tulajdonságai:

- végtelen sokszor újraprogramozható (statikus RAM)
- táp kikapcsolása után az SRAM elveszti tartalmát
- bekapcsoláskor (inicializáláskor) a programot be kell tölteni, fel kell programozni
- az SRAM cellára egy áteresztő tranzisztor van csatolva. A tranzisztor vagy kinyit (vezet), vagy lezár. Az SRAM értéke, ami egy bitet tárol ('0' vagy '1') letölthető. Összeköttetéseket, vagy MUX-ok állását is eltárolja.
- 1 bit tárolása az SRAM-ban (min. 6 tranzisztorból áll)
- sok tranzisztor (standard CMOS), nagy méret, nagy disszipáció
- nem kell frissíteni az SRAM-ot
- nagy 0.5-2 kΩ átmeneti ellenállás
- nagy 10-20 fentoF parazita kapacitás

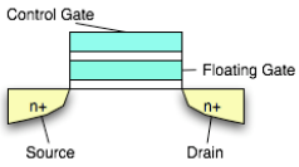


b.) Multiplexeres: az SRAM-ban tárolt '0' vagy '1' bitet használunk a Multiplexer bemeneti vonalának kiválasztásához. (Működése hasonló az SRAM celláéhoz.)



c.) Floating Gate (lebegő gate): Két-gates tranzisztort használunk, amelynek középső gate-je a lebegő gate. A másik gate fix. INTEL alkalmazza. Programozható összeköttetéseknél, csomópontokban használatos.
Tul:

- programozása: töltéseket viszünk fel a word-line felől a lebegő gate-re, kinyit a tranzisztor (a control gate befolyásolja a működését)
- többször törölhető (kis ablakon keresztül UV fénnyel)
- írás: nagy feszültség hatására töltést viszünk fel a lebegő gate-re
- kikapcsoláskor is megőrzi tartalmát, töltések nem sülnek ki (akár 99 évig)
- nagy 2-4 kΩ átmeneti ellenállás
- nagy 10-20 fentoF parazita kapacitás

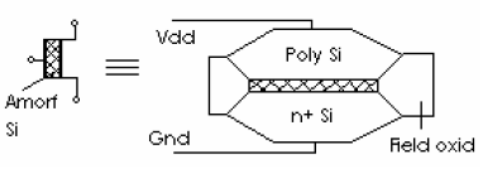


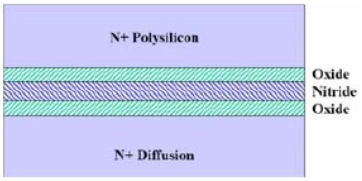
d.) Antifuse:

A tranzisztor Gate-jét amorf kristályos Si alkotja, amelyet relatíve nagy feszültség (kb 20-30V) hatására átkristályosítunk, így vezetővé válik végérvényesen. Texas Instruments alkalmazza.

Tul:

- az „átégetés" irreverzibilis folyamat, nem lehet újraprogramozni
- csak egyetlen egyszer programozható
- kis méreten megvalósítható, kis disszipáció
- kis átmeneti ellenállás 300 Ω
- kis parazita kapacitás 1.1-1.3 fentoF
- előállításához sok maszkréteg szükséges, drága technológiát igényel
- Típusai: Amorf Si vagy ONO (Oxid-Nitrid-Oxid)

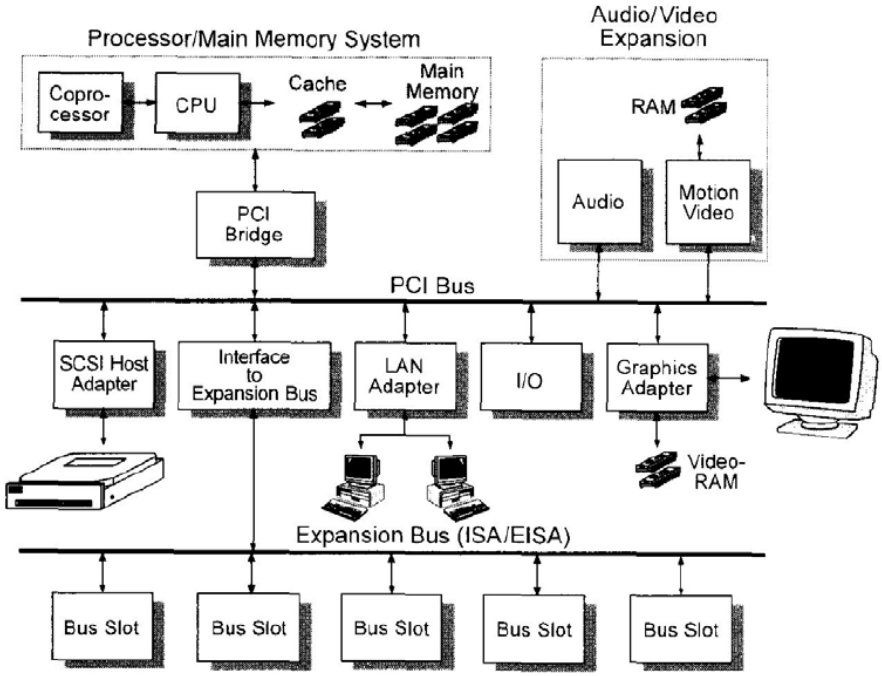




·.) EPROM/EEPROM/FLASH-os: (Lattice)

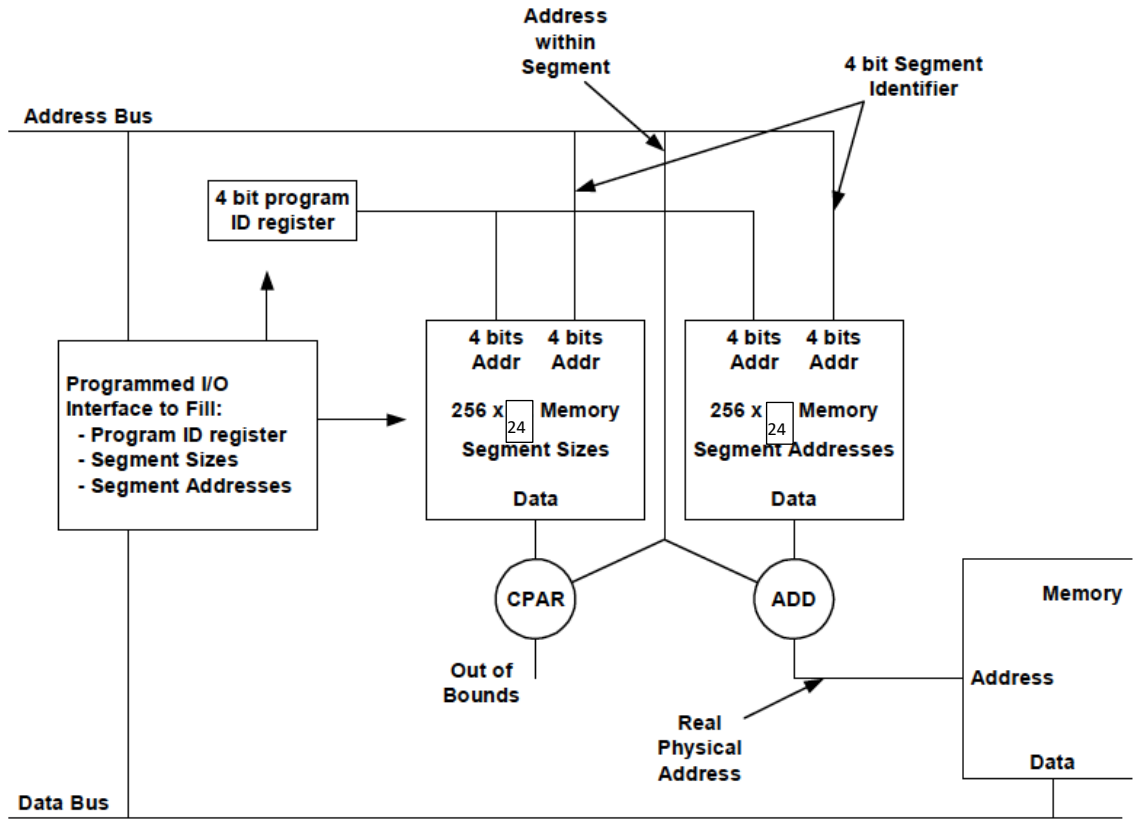
- **A floating-gates technológiát alkalmazza!**
- Megőrzi tartalmát (non-volatile)
- elektromosan írható/törölhető végtelen sokszor (EEPROM, FLASH)
- UV-fénnyel törölhető (EPROM)
- nagy átmeneti ellenállás 2-4 kΩ
- nagy parazita kapacitása
- További gyártástechnológia (sok maszk-réteg alkalmazása szükséges) - drága

RAJZOLJA FEL EGY PCI-EXPRESS BUSZOS SZÁMÍTÓGÉP BLOKKDIAGRAMJÁT. (2P)



RAJZOLJON FEL EGY SZEGMENTÁLT VIRTUÁLIS MEMÓRIA KEZELŐ ÁRAMKÖRT MEMÓRIAVÉDELEMMEL! SZÜKSÉGES PARAMÉTEREK: VIRTUÁLIS CÍM: 32 BIT, SZEGMENSEK SZÁMA: 16, FIZIKAI CÍM: 24 BIT. (3P)

Szegmentálásos címfordító áramkör:

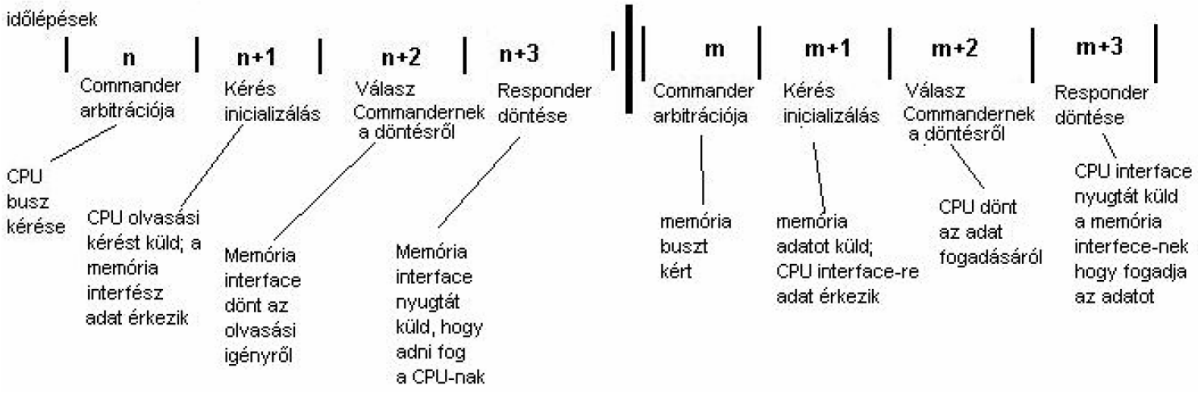


2018/B

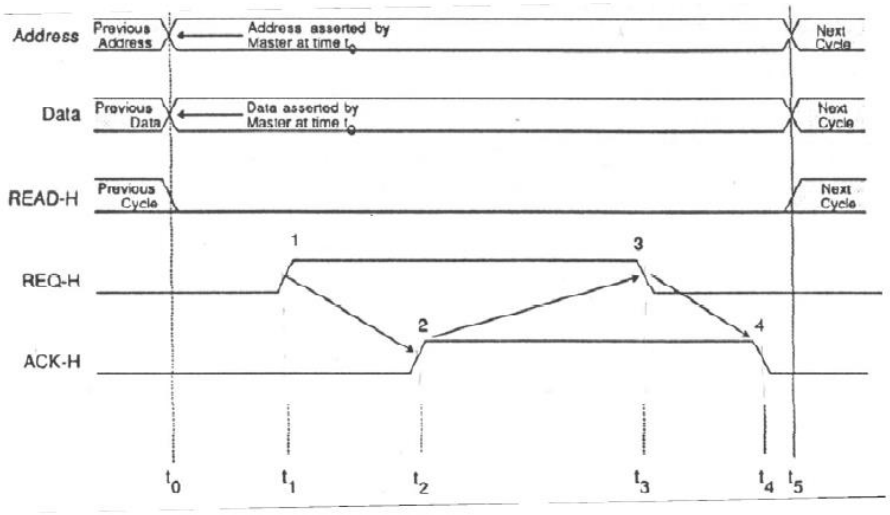
JELLEMEZZE A SZINKRON OLVASÁSI PROTOKOLLT (LÉPÉSEK + FELADATOK)! (2P)

\+ Közös órajellel működnek a buszra csatlakozó egységek. Tehát a műveleti időt az órajelciklus határozza meg. Mivel nincs szükség párbeszédre (pl. handshaking), gyorsabb az aszinkron működésénél. Jel: Master=Commander, a Slave=Responder.
\- Az órajelet mindig a leglassabb (legtávolabb lévő) egységhez kell igazítani.

Olvasási ciklus: (CPU ← MEM)



RAJZOLJA LE AZ ASZINKRON HAND SHAKING FOLYAMATOT ÍRÁS ESETÉN! (2P)

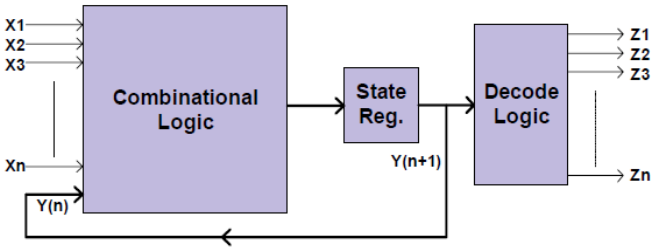


Write: Master ír Slavennek.

- t0-ban master megadja a kívánt slave címét.
- t1-ben master beállítja a request vonalat (minden slave veszi, de csak a toban kiválasztott válaszol)
- t2-ben slave nyugtázza, hogy megkapta az adatokat
- t3-ban master is megkapja a slave nyugtáját, felszabadítja a request vonalakat
- t4-ben slave érzékeli és felszabadítja a nyugtázó vonalat
- t5-ig a master a címvonalat tartja még (a cím esetleges megváltozása miatt)

DEFINIÁLJA A MOORE AUTOMATA MODELLT (3 HALMAZ, 2 LEKÉPZÉS, 1 RAJZ)! (2P)

Itt a kimenetek közvetlenül csak a pillanatnyi állapottól függenek (a bemenettől függetlenek). Tehát a kimenetet nem a bemenethez, hanem az állapotoknak megfelelően szinkronizáljuk. Ezért használunk állapotregisztert.
Három halmaza van: X – a bemenetek, Z – a kimenetek, Y – az állapotok halmaza. Visszacsatolni az állapotregisztert a késleltetés miatt kell.
Halmazok közötti leképezési szabály:
$\mu(Y_n) \rightarrow Z$: kimenet meghatározása!
$\delta(X_n, Y_n) \rightarrow Y_{n+1}$: következő állapot meghatározása
/Input – Next State – Present State – Output/

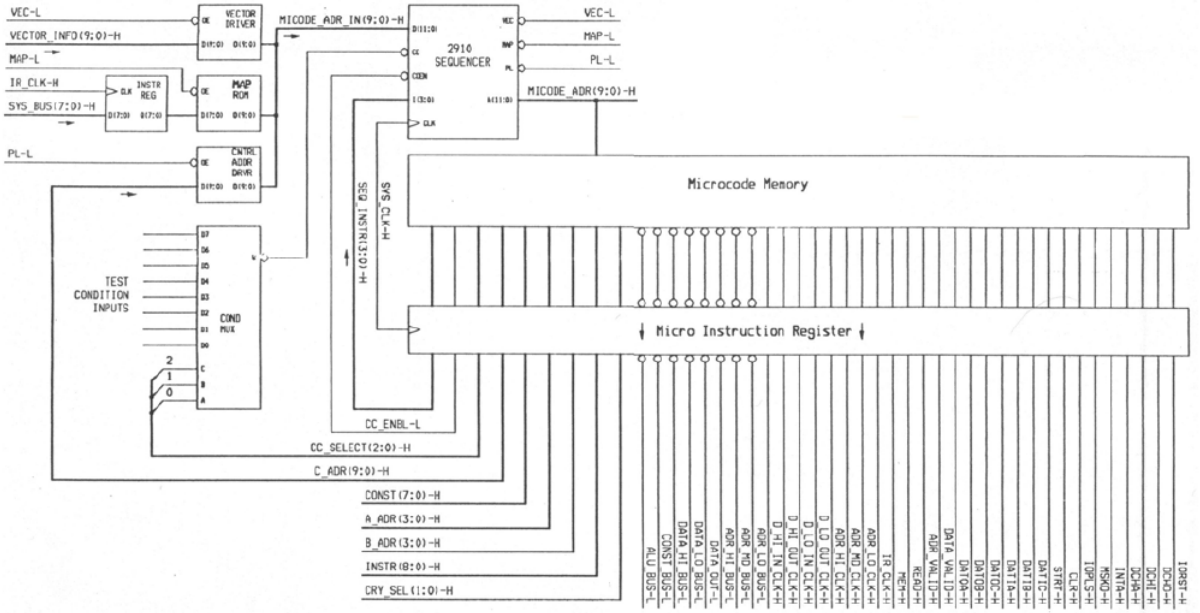


MIKROPROGRAMOZOTT VEZÉRLŐEGYSÉG FELADATA, HORIZONTÁLIS MIKROPROGRAMOZÁS (ÁBRA, ELŐNYÖK, HÁTRÁNYOK). (3P)

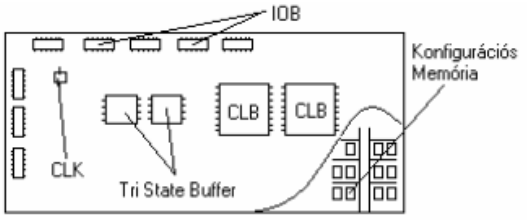
Feladata a memóriában lévő gépi kódolású utasítások részműveletekre bontása és funkcionális egységek vezérlése. Mikroprogramozott, vagyis az adatútvonal vezérlési pontjait (ROM) memóriából kiolvasott vertikális- vagy horizontális mikrokódú utasításokkal állítjuk elő.
Horizontális mikrokód:

- minden egyes vezérlőjelhez saját vonalat rendelünk → megnő a mikrokód
- minél több funkciót valósítunk meg, annál szélesebb lesz a mikrokód
- leggyorsabb mikrokódos technika
- minden bit független egymástól, egy mikrokóddal többszörös utasítás is megadható
- nincs szükség a vezérlőjelek dekódolását végző dekódoló logikára → növekszik a sebesség
- minimálisra csökken a műveletek ciklusideje
- nagyobb az erőforrás szükséglete és fogyasztása

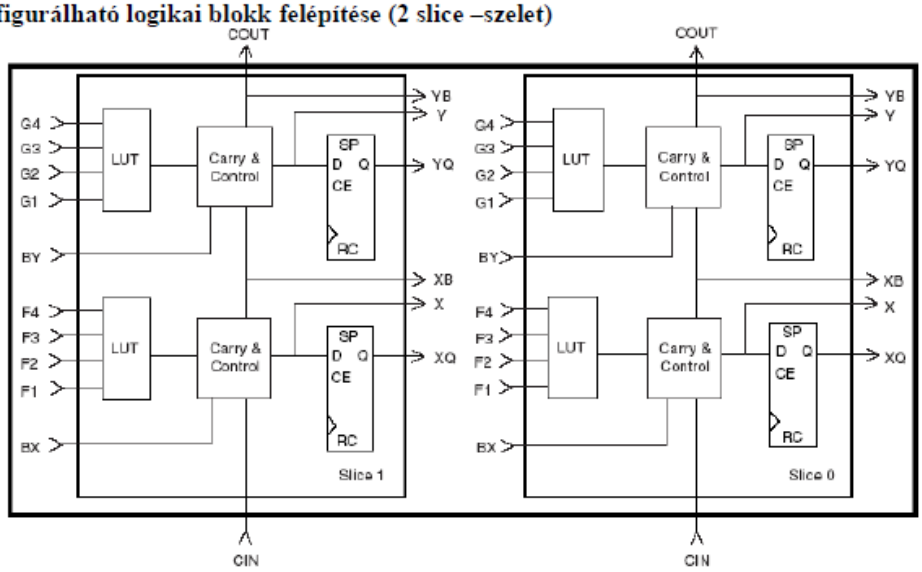
1. Horizontális mikrokód: minden egyes vezérlőjelhez saját vonalat rendelünk, ezáltal horizontálisan megnő a mikro-utasításregiszter kimeneteinek száma, (=horizontálisan megnő a mikrokód). Minél több funkciót valósítunk meg a vezérlőjelekkel, annál szélesebb lesz a mikrokód. Ennek köszönhetően ez a leggyorsabb mikrokódos technika, mivel minden bit független egymástól ill. egy mikrokóddal többszörös (konkurens) utasítás is megadható. Pl: a megfelelő regisztereket (memória, ACC) egyszerre tudjuk az órajellel aktiválni, ezáltal egy órajelciklus alatt az információ mindkét irányba átvihető. Növekszik a sebesség, mivel nincs szükség a vezérlőjelek dekódolását végző dekódoló logikára. Így minimálisra csökken a műveletek ciklusideje. Azonban nagyobb az erőforrás szükséglete, és fogyasztása.



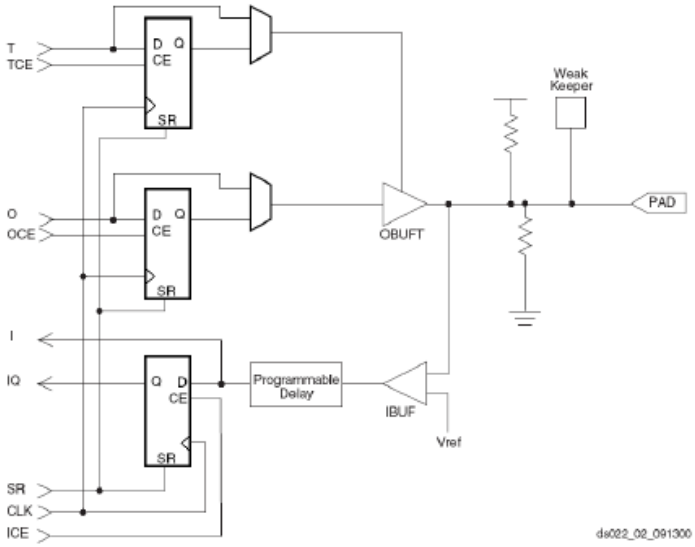
RAJZOLJA FEL A XILINX FPGA BELSŐ FELÉPÍTÉSÉT (CLB ÉS I/O). (2P)



Virtex CLB: Konfigurálható logikai blokk felépítése (2 slice –szelet)



Virtex IOB - I/O Blokk felépítése:



- Virtex – CLB (Konfigurálható Logikai Blokk): Szükséges logikai kapcsolatok megvalósítása egy logikai tömbben.
- IOB Input/Output Blokk (programozható be/kimeneti blokk): Kapcsolat a belső vonalak és a tokozás lábai között. A kimenet lehet negált vagy ponált.

MINIMUM HÁNY LÁBRA VAN SZÜKSÉG EGY 128KX16-OS 1 CS-SEL ÉS KÖZÖS I/O-VAL RENDELKEZŐ SRAM MEGVALÓSÍTÁSÁHOZ? (2P)

- 16 db I/O
- 17 db cím [2^{17}=128*1024=128k]
- 1 db R/W

- 1 db VDD (betáp)
- 1 db GND (föld)
- 1 db CS

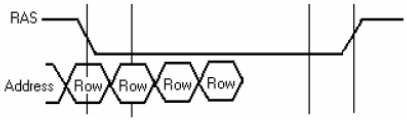
Összesen: 16+17+4= ~37

DEFINIÁLJA A CACHE MISS ÉS A CACHE HIT FOGALMÁT. (2P)

- Cache hit: Ha processzor olyan adatot igényel, mely a cache-ben megtalálható, akkor találatról vagy cache hit-ről beszélünk. A találatot a cachevezérlő azzal állapítja meg, hogy a bemásolt blokkokhoz tartozó címrészek (toldalék vagy „tag") alapján 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.

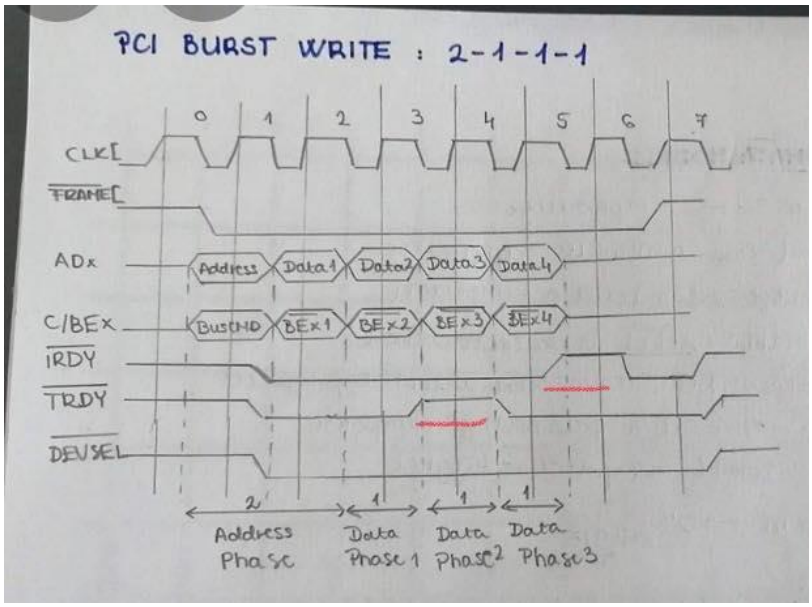
ISMERTESSE A DRAM FRISSÍTÉSI FOLYAMATÁT (IDŐDIAGRAM) (2P)

Dinamikus memória frissítési folyamata! (idődiagram)
Frissíteni kell, mivel kondenzátoron tároljuk az információt, melynek feszültsége idővel exponenciálisan csökken, tehát kisül.



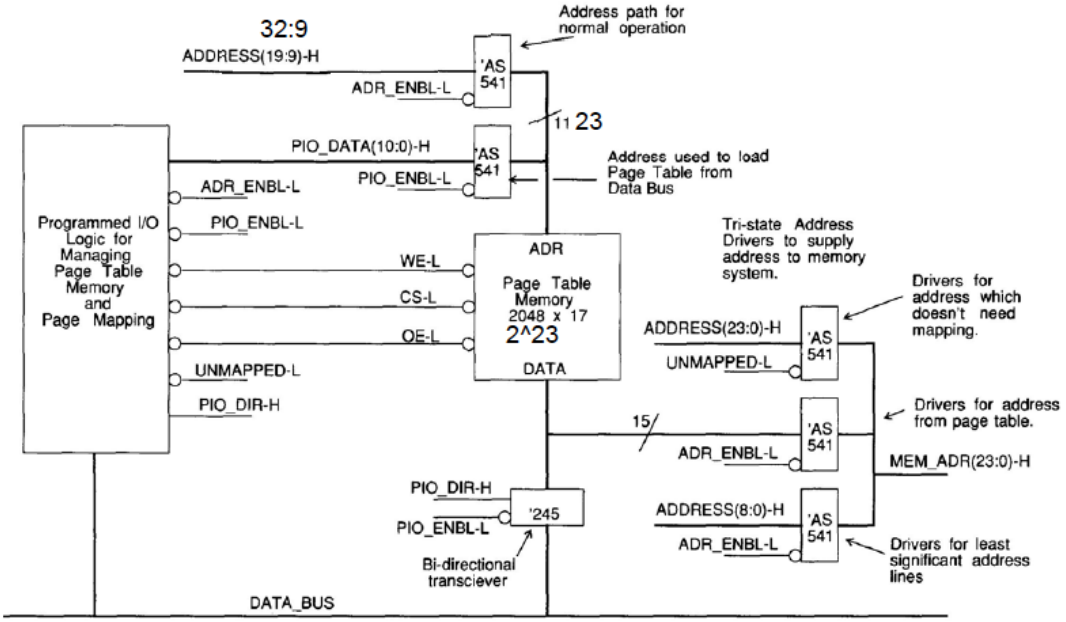
A frissítéskor először például egyetlen CAS oszlopcímet adunk ki, majd a RAS-al az összes sorcímet, hogy az összes cella frissítésre kerüljön. Frissítés történhet a kiolvasási ciklus végeztével is (még a következő beírás előtt). Átlagosan: 2-10ms.

RAJZOLJA FEL EGY PCI BURST WRITE TRANZAKCIÓ IDŐDIAGRAMJÁT IDEÁLIS ESETBEN. (2P)



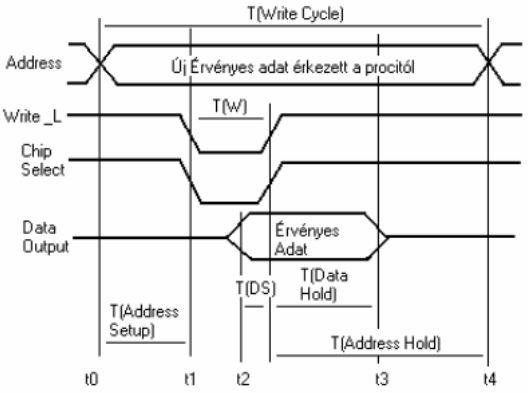
RAJZOLJON FEL EGY LAPOZÁSOS VIRTUÁLIS MEMÓRIA KEZELŐ ÁRAMKÖRT! SZÜKSÉGES PARAMÉTEREK: VIRTUÁLIS CÍM: 32 BIT, LAPMÉRET: 4K, FIZIKAI CÍM: 24 BIT. (3P)

Lapozásos címfordító áramkör:



EGYÉB FELADATOK

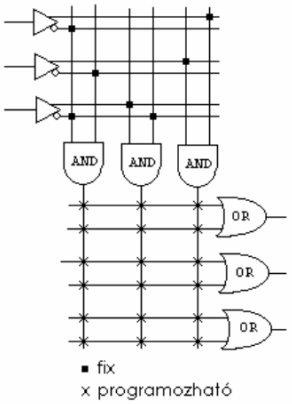
ISMERTESSE A STATIKUS MEMÓRIA ÍRÁSI FOLYAMATÁT (IDŐDIAGRAM)! (2P)



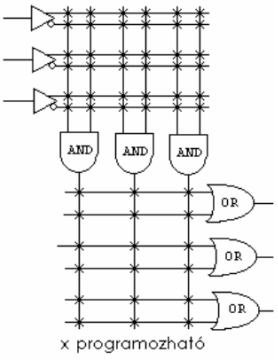
VEZÉRLŐEGYSÉG PROGRAMOZHATÓ MAKROCELLÁKKAL (FAJTA, ÁBRÁK, JELLEMZŐ TULAJDONSÁGOK). (2P)

a., PROM (Programmable Read-Only Memory)

- speciális feladatokra
- **információkat írás után már nem lehet megváltoztatni** (kiv.:EPROM, EEPROM)
- tápfesz. nélkül is **megőrzik** tartalmukat
- cella felprogramozása felhasználó által, biztosíték átégetésével
- cellák négyzetes mátrixban
- címzés: sor-oszlop vonalra 1-et adva → dekódolni kell
- összes lehetséges minterm előállnak OR kapcsolattokként (n bemenet → 2^n kimeneti kombináció)
- **AND=fix, OR=programozható** (EEPROM/EPROM)

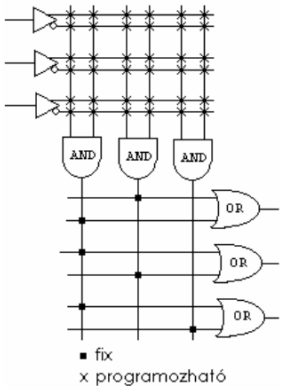


- OR kapuk kimenetelénél tároló regiszterek (**D flip-flop**) – visszacsatolódhatnak a bemenetre



b., PLA (Programmable Logic Array)

- alapja a **diszjunktív normálforma → mintermek VAGY** kapcsolata
- 3-állapotú inverterek, AND, OR kapuk alkotják
- **felső mátrix**: ponált bemeneti változók és negáltjaik közötti ÉS kapcsolat (kereszteződő vezetékek segítségével)
- **alsó mátrix**: ÉS kapuk kimeneteleit felhasználva VAGY függvények
- AND, és OR kapuk **is programozhatóak**
- OR kapuk kimenetelei **D-tárolók** – visszacsatolódhatnak a bemenetre

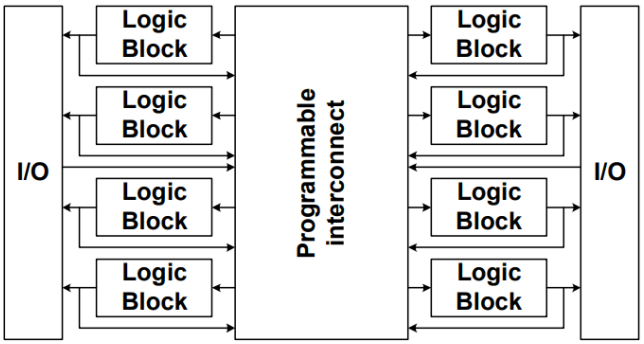


c., PAL (Programmable Array Logic)

- hasonló, mint PLA
- DE! **AND=programozható, OR=fix**
- **gyorsabb**, mint PLA: fix összeköttetések → kevesebb kapcsoló
- kimenetelei **D-tárolók** – visszacsatolódhatnak a bemenetre

d., CPLD (Complex Programmable Logic Device)

- **Logikai blokk**jában:
  - PLA/PAL
  - Regiszterek
- Interconnect (**összeköttetés**):
  - teljes
  - részleges összeköttetés hálózat

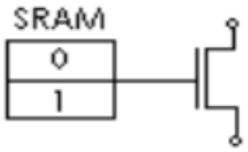


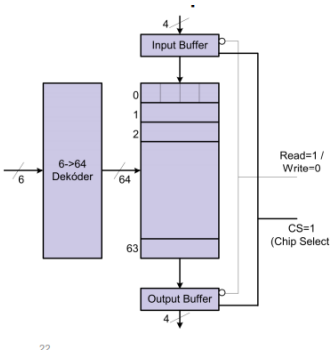
MINIMUM HÁNY LÁBRA VAN SZÜKSÉG EGY 1024X4-ES 1 CS-SEL ÉS KÖZÖS I/O-VAL RENDELKEZŐ RAM MEGVALÓSÍTÁSÁHOZ? (2P)

CS → **SRAM**-ról van szó

- 4 db I/O
- 10 db cím [2^{10}=1024]
- 1 db R/W
- 1 db VDD (betáp)
- 1 db GND (föld)
- 1 db CS
- **Összesen:** 4+10+4= ~18

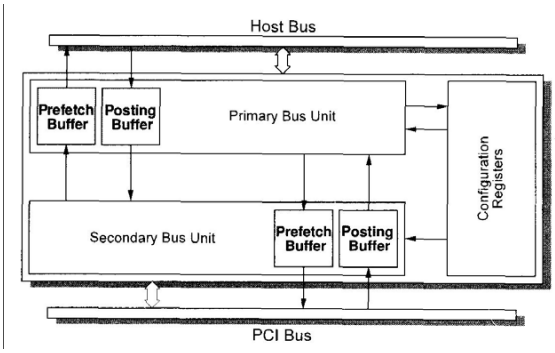
kép: 64 x 4-es SRAM





MILYEN ÁTVITELI TECHNOLÓGIÁT HASZNÁLNAK A PCI BUSZ JELEI ÉS MILYEN JELCSOPORTOK TALÁLHATÓK A PCI CSATLAKOZÓN, TOVÁBBÁ MILYEN A CSATLAKOZÓ FELÉPÍTÉSE? (2P)

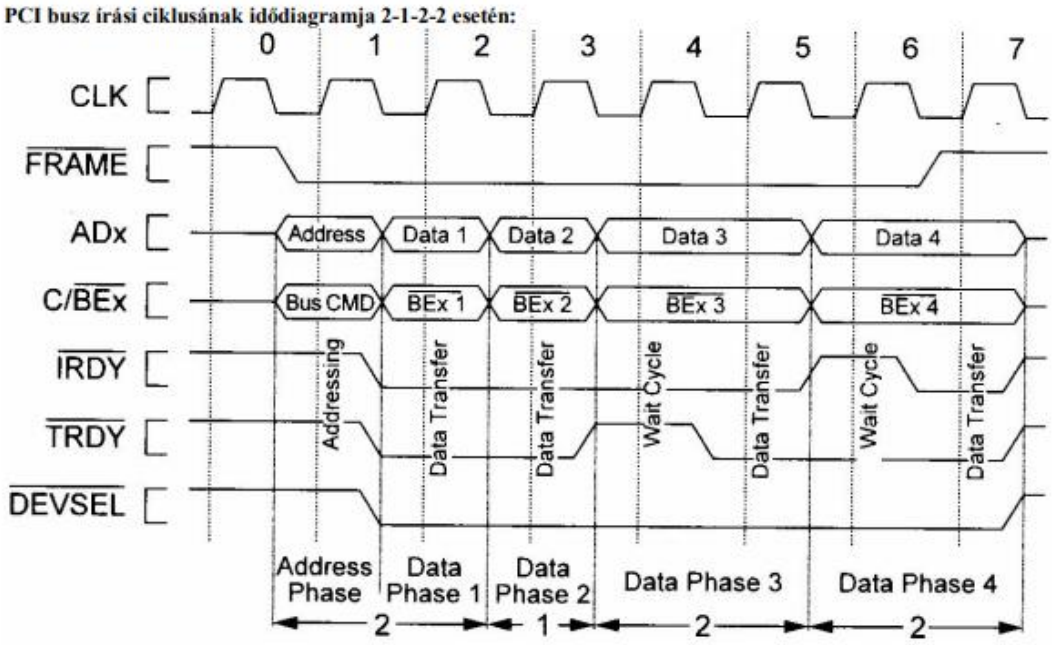
- szinkron kommunikációs protokollt használ
- **Arbitrációs mechanizmust használ**
- PCI: teljesen új, önálló szabvány, többprocesszoros rendszereket is támogat
  33 MHz-es órajel támogatás
- **Kapcsolatot** a processzor és a PCI sin között egy PCI HOST BRIDGE biztosítja
- PCI 32 bites multiplexált adat/cím jeleket alkalmaz (szerverekben: 64 bites)
- **PCI busz jelei**
  - CLK
  - AD0-AD31
  - AD32-AD63
  - IRDY
  - TRDY
  - DEVSEL
  - IDSEL
  - STOP
  - FRAME
  - PAR
  - C/BE3-C/BE0
  - PERR-SERR
  - INTA-INTD
  - REQ
  - GNT
  - TCK, TDI, TDO, TRST
  - RST
- **Jelcsoportok a PCI csatlakozón:**
  - csatlakozó: 188 lábú, oldalankánt 94-94 láb
  - sok GND (földelés) → zavarások elkerülése végett
  - tápról: 12 V, 5 V, 3.3 V-os jel is



PCI BRIDGE

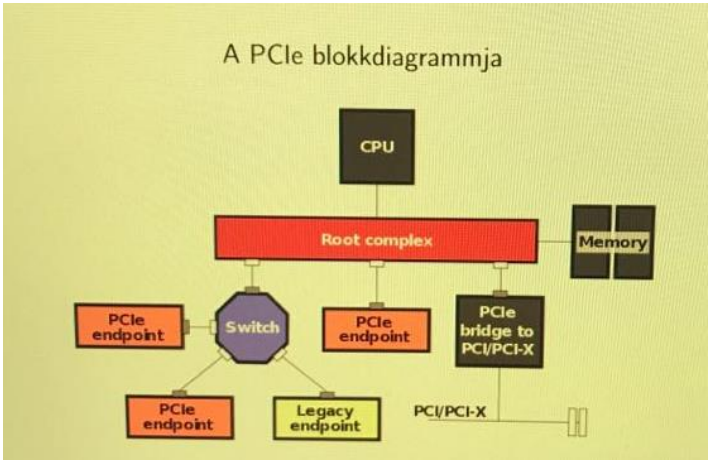
- PCI buszt köti össze a rendszerbusszal, HOST busszal
- konfigurációs regiszter a részét képezi →két egység ezen keresztül (is) megcímezhető
- HOST busszal az elsődleges busz egységhez: Prefetch, Posting bufferek

RAJZOLJA FEL EGY PCI BURST WRITE TRANZAKCIÓ IDŐDIAGRAMMJÁT ÉS A DIAGRAMON MUTASSON PÉLDÁT A VÁRAKOZÁSI CIKLUSRA MIND A KÉT EGYSÉG RÉSZÉRŐL! (2P)

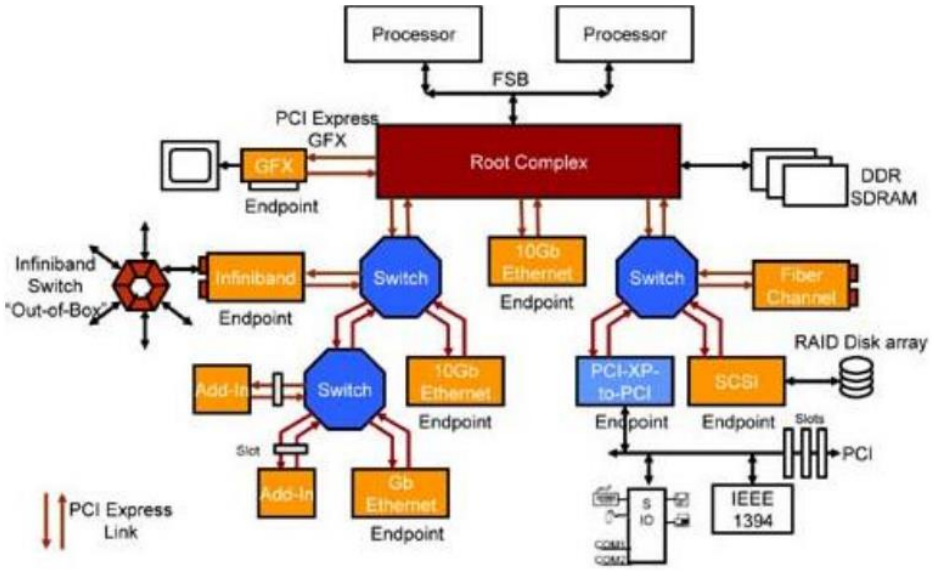


Azért van két várakozási egység, mert egyszer a TRDY foglalt(1), ezért nem lehet adatot írni. A másodiknál pedig az IRDY foglalt(1) ezért nem lehet adatot olvasni.

RAJZOLJA FEL EGY PCI-EXPRESS SZERVER ALAPÚ RENDSZER BLOKKDIAGRAMJÁT! (1P)



ez egy egyszerűbb ábra ugyanarra



HOGYAN HASZNÁLHATÓ EGY MEMÓRIÁBÓL TÖBB FÜGGETLEN PROGRAM? (2P)

multiprogramming: Minden programnak saját logikai/bázis címtartománya van, amellyel hivatkozni lehet rá

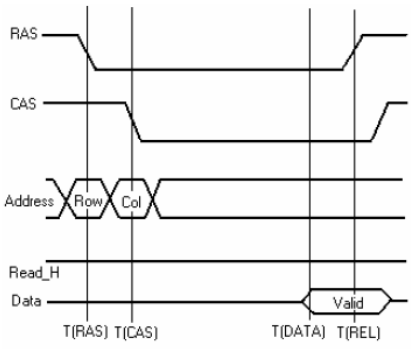
- sok feladat egymás után jön létre
- CPU észleli, ha:
    - folyamat befejeződik/megszűnik
    - vár az I/O-ra
    - CPU leáll
- cél: CPU kihasználtságának a maximalizálása

MEKKORA MEMÓRIA CÍMEZHETŐ 21 BITTEL? (1P)

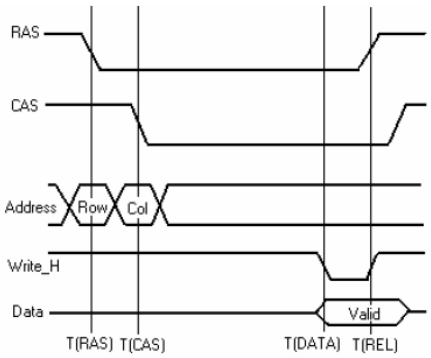
2^21 rekesz van, vagy ha a rekesz minimális mérete 1 bit, akkor a rekesz helyett bitről beszélhetünk.

ISMERTESSE A DINAMIKUS MEMÓRIA ÍRÁSI-OLVASÁSI FOLYAMATÁT (IDŐDIAGRAM)! (2P)

Olvasási ciklus:

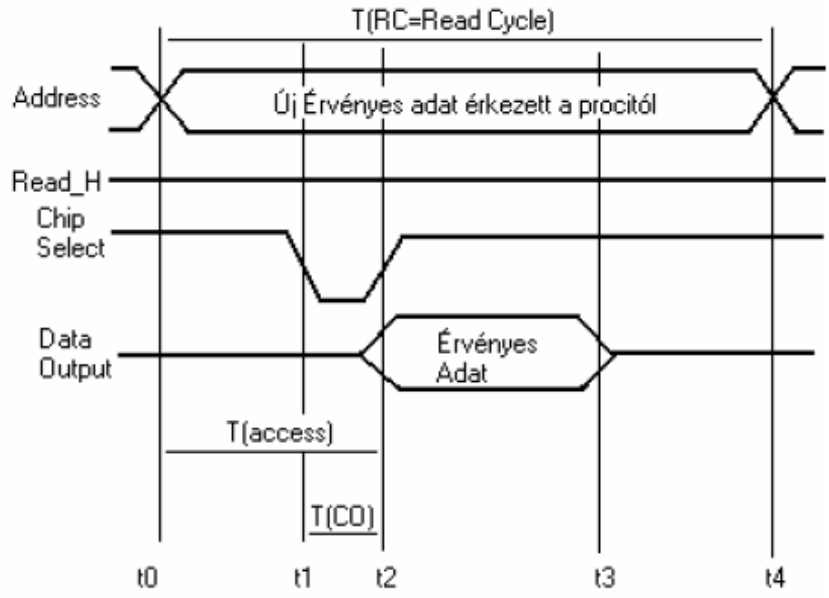


Írási ciklus:

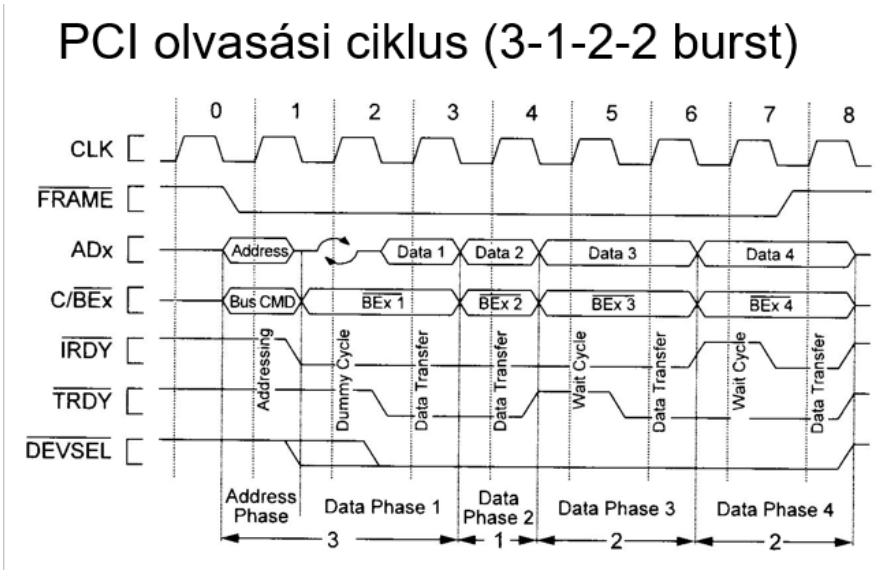


ISMERTESSE A STATIKUS MEMÓRIA OLVASÁSI FOLYAMATÁT (IDŐDIAGRAM) (2P)

Olvasási ciklus:



RAJZOLJA FEL EGY PCI BURST READ TRANZAKCIÓ IDŐDIAGRAMMJÁT ÉS A DIAGRAMON MUTASSON PÉLDÁT A VÁRAKOZÁSI CIKLUSRA MIND A KÉT EGYSÉG RÉSZÉRŐL! (2P)

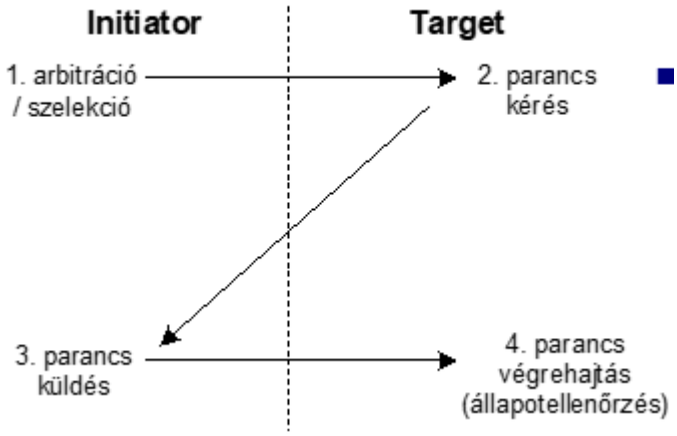


HATÁROZZA MEG A SCSI KOMMUNIKÁCIÓS LÉPÉSEKET ÉS RAJZOLJA FEL A KOMMUNIKÁCIÓ FOLYAMATÁT! (2P)

Az SCSI 4 fő kommunikációs fázisa a következő (handshaking):

reselection- Ha szükséges, az initiator újra kapcsolatot létesít a korábbi targettel, miután egy megszakítás történt (~folytatódhat a művelet)

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



A teljes SCSI busz fázisok a következők:

- ☐ - Reselection (ha a target megszakítva lett, folytathatja a komm.t az initiatorral)
- ☐ - busz szabad (bus free?)
- ☐ - arbitráció / szelekció (ki adjon?)
- ☐ - üzenetküldés (msg)
- ☐ - parancs elmegy (comm)

- ☐ - adatkapcsolat (data)
- ☐ - állapotellenőrzés (status)
- ☐ - üzenetzárás, kapcsolatbontás

HASONLÍTSA ÖSSZE A SZEGMENTÁLÁST ÉS A LAPOZÁST! (2P)

Lapozás	Szegmentálás
a program és a memória is fizikailag **egyenlő méretű** darabokra van osztva	a program **változó méretű logikai** egységekre van feldarabolva
fizikai cím = lap báziscíme + lap eltolás (offset) címe, ahol a „+” **konkatenációt/összefűzést** jelent	fizikai cím = szegmens báziscíme + szegmensen belüli eltolás (offset) címe, ahol a „+” **összeadást** jelent
laptábla: lap száma, lapcím (hexa), egyes lapok elérési információi	**szegmenstábla:** szegmens száma, szegmens hossza, szegemenscím (hexa), egyes szegmensek elérési információi
ADDR(REAL) = ADDR(BASE) + ADDR(OFFSET)	

JELLEMEZZE A LAPOZÁSOS VIRTUÁLIS MEMÓRIA KEZELÉST (RAJZ + JELLEMZŐ PARAMÉTEREK TÁROLÁSA)! (2P)

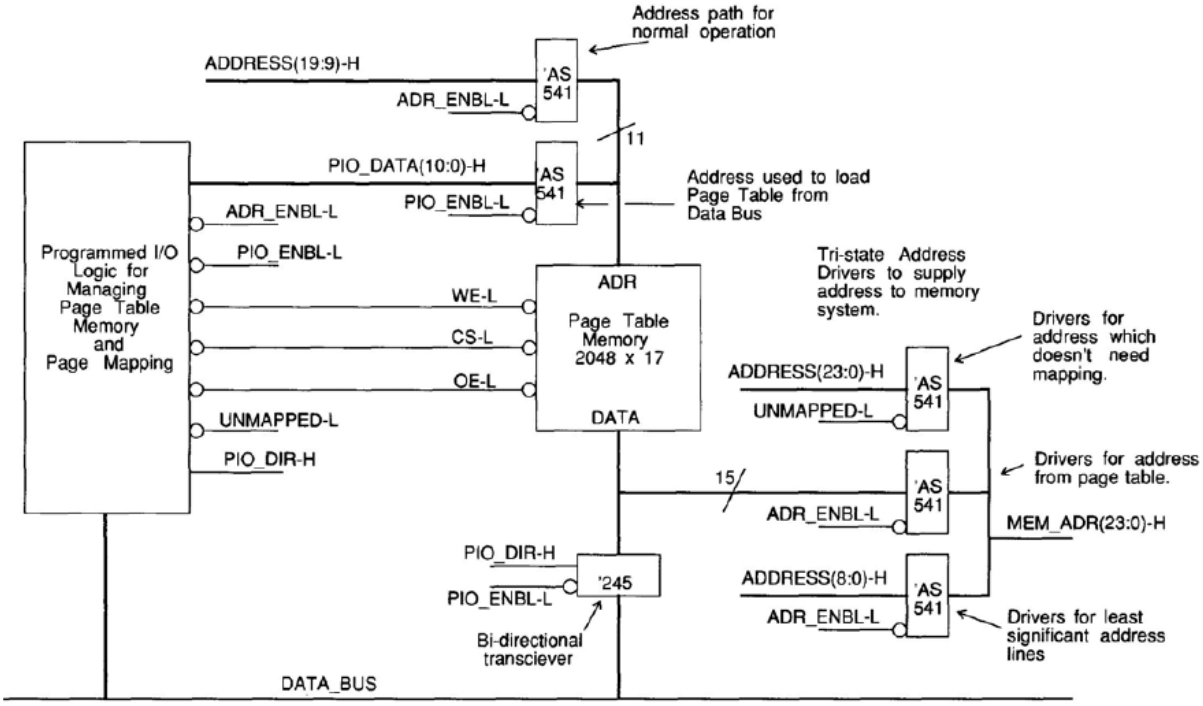
Lap: a program és a memória is fizikailag egyenlő méretű darabokra van osztva (fizikai határok). Főprogram, szubrutinok, globális/lokális adatok, verem mind-mind egy-egy azonos méretű lapnak felelnek meg.

- Hátrány: gazdaságtalan (nem használja ki a rendelkezésre álló lapméretet).
- Előny: gyorsabb fizikai cím leképezés (nem kell összeadót használni).

Fizikai Cím = Lap Báziscíme + Lap eltolás címe (itt a + konkatenáció/összefűzés, NEM összeadás)

Fizikai címet a CPU határozza meg. Lapokat célszerű a memóriában tárolni. A konkatenációt huzalozással oldhatjuk meg. Laptáblából olvassuk ki a lap számát követően a hozzátartozó lapcímet, továbbá az egyes lapok elérési információit: (R/W/X).

Lapozásos címfordító áramkör:



JELLEMEZZE A SZEGMENTÁLT VIRTUÁLIS MEMÓRIA KEZELÉST (RAJZ + JELLEMZŐ PARAMÉTEREK TÁROLÁSA)! (2P)

Szegmens: a program változó méretű logikai egységekre van feldarabolva. A program 4 fő szegmense:

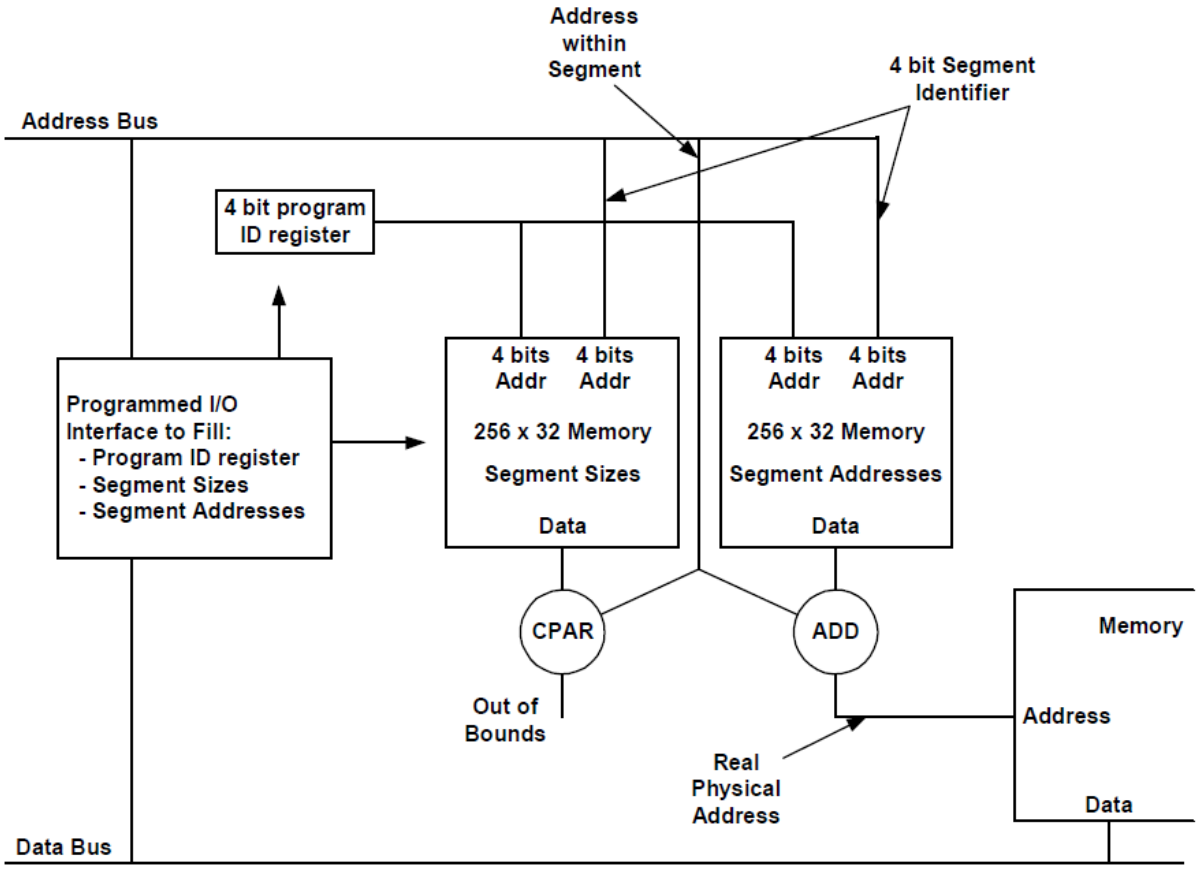
- 0.: főprogramok (olvasható/végrehajtható)
- 1.: szubrutinok
- 2.: csak olvasható adatok
- 3.: olvasható/írható adatok

A program a memóriát közvetlenül nem érheti el, csak a szegmensen keresztül.

FIZIKAI CÍM = Szegmens bázis címe + Szegmensen belüli eltolás címe (ez tényleg összeadás!!!)

Szegmenstábla kezelése az OS feladata. Szegmenstáblából olvassuk ki a szegmens számát követően a hozzátartozó szegmens hosszát, szegmens címet, továbbá z egyes szegmensek elérési információit: (R/W/X) olvasható/írható/végrehajtható. Szegmentálás hátránya: külső töredezettség.

Szegmentálásos címfordító áramkör:



RAJZOLJA FEL A XILINX FPGA-K FEJLESZTŐI RENDSZERÉNEK FELÉPÍTÉSÉT!