Pannon Egyetem
Képfeldolgozás és Neuroszámítógépek Tanszék

Digitális Rendszerek és Számítógép Architektúrák

4. előadás: Vezérlő egységek

Előadó: Vörösházi Zsolt
voroshazi@vision.vein.hu

Jegyzetek, segédanyagok:

- Könyvfejezetek:
  - http://www.knt.vein.hu
    ⇒ Oktatás ⇒ Tantárgyak ⇒ Digitális Rendszerek és Számítógép Architektúrák
    (chapter05.pdf)
- Fóliák, óravázlatok .ppt (.pdf)
- Feltöltésük folyamatosan

Vezérlő egységek általánosan

- A számítógép vezérlési funkcióit ellátó *szekvenciális* egység.
- **Feladata**: az operatív tárban lévő gépi kódú utasítások értelmezése, részműveletekre bontása, és a szekvenciális (sorrendi) hálózat egyes funkcionális részeinek vezérlése (vezérlőjel- és cím-generálás)
- Vezérlő egység tervezési lépései:
  - megfelelő technológia, és rendszerkomponensek kiválasztása
  - komponensek összekapcsolása a működési sorrendnek megfelelően
  - RTL leírás alkalmazása az akciók ill. adatátvitel pontos leírására
  - adatút (data-path) megtervezése (*legfontosabb*!)
  - kívánt vezérlő jelek azonosítása, meghatározása

Adatút (Data-path) tervezés szempontjai

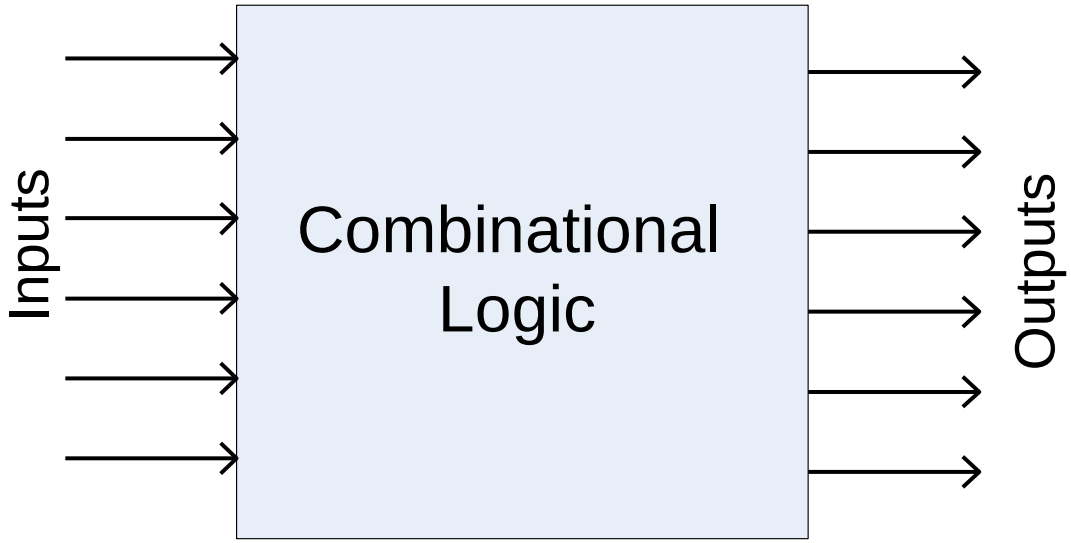
- Gazdaságosság (költség)
- Interfész szükséglet (protokollok)
- Sebesség
- Energia (disszipált teljesítmény)
- Dinamikai tartomány (számrendszerek)
- Rugalmasság (többcélúság)
- Kezelhetőség (probléma, hiba során)
- Környezet (pl. ipari v. irodai használat?)

Vezérlő egységek fajtái:

- Huzalozott (klasszikus módszerrel):
  - Mealy-modell,
  - Moore-modell (Példa: FIR szűrő tervezése – FSM állapotgép segítségével).
    - Multiplexeres / késleltetéses / Shift-regiszteres megvalósítások
- Mikroprogramozott (reguláris vezérlési szerkezettel):
  - Horizontális mikrokódos vezérlő,
  - Vertikális mikrokódos vezérlő.
- Programozható logikai eszközök (PLD):
  - PLA, PAL, PROM, CPLD,
  - FPGA!

Kombinációs hálózatok

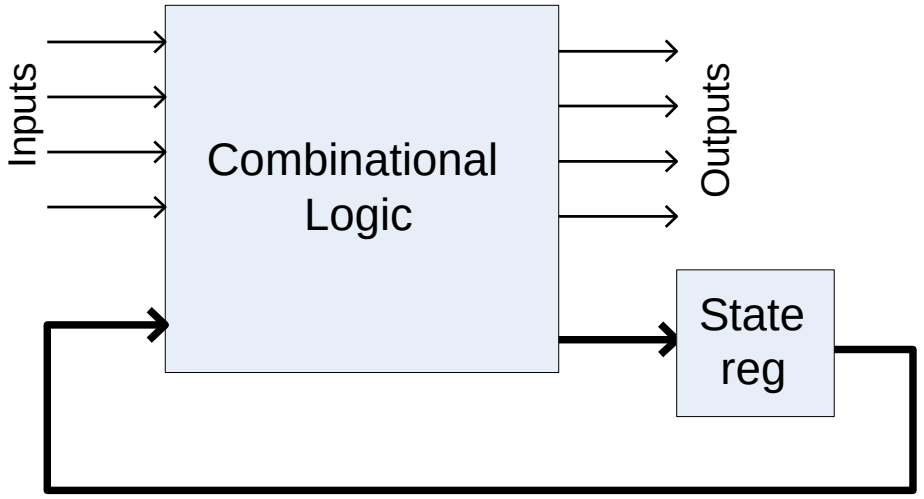
- **(K.H.) Kombinációs logikai hálózat**ról beszélünk: ha a mindenkori kimeneti kombinációk értéke csupán a bemeneti kombinációk pillanatnyi értékétől függ (tároló „kapacitás”, vagy memória nélküli hálózatok).



Sorrendi hálózatok:

- **(S.H.) Sorrendi (szekvenciális) logikai hálózat**ról beszélünk**:** ha a mindenkori kimeneti kombinációt, nemcsak a pillanatnyi bemeneti kombinációk, hanem a korábban fennállt bementi kombinációk és azok sorrendje is befolyásolja. (A *szekunder /másodlagos kombinációk* segítségével az ilyen hálózatok képessé válnak arra, hogy az ugyanolyan bemeneti kombinációkhoz más-más kimeneti kombinációt szolgáltassanak, attól függően, hogy a bemeneti kombináció fellépésekor, milyen értékű a szekunder kombináció, pl. a State Register tartalma)

Vezérlő egységek alapjául szolgáló hálózat!



Időzítő – vezérlő egység:

- Az **időzítő (ütemező)** határozza meg a vezérlő jelek előállításának *sorrendjét*.
- Egy **időzítő-vezérlő** egység általános feladata az egyes funkciók megvalósítását végző áramköri elemek (pl. ALU, memória elemek) összehangolt működésének biztosítása.
- Az időzítő-vezérlő áramkörök ***szekvenciális*** *rendszerek* – mivel az áramköri egységek tevékenységének egymáshoz viszonyított *időbeli sorrendiségét* biztosítják – melyek az aktuális *kimenet* értékét a *bemenet*, és az *állapotok* függvényében határozzák meg.

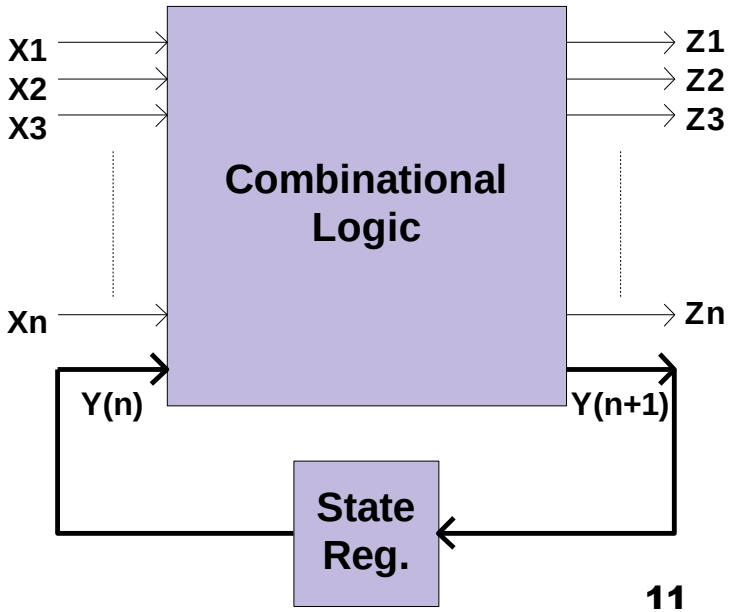
Az időzítő-vezérlő lehet:

- **huzalozott**: áramkörökkel, *dedikált* összeköttetésekkel fizikailag megvalósított (Mealy, Moore, MUX-os modellek alapján, illetve programozható PLD-k)
- **mikroprogramozott**: az adatútvonal (data-path) vezérlési pontjait memóriából (ROM) kiolvasott *vertikális*- vagy *horizontális*-mikrokódú utasításokkal állítják be

Huzalozott vezérlő egységek

1.) Mealy-modell

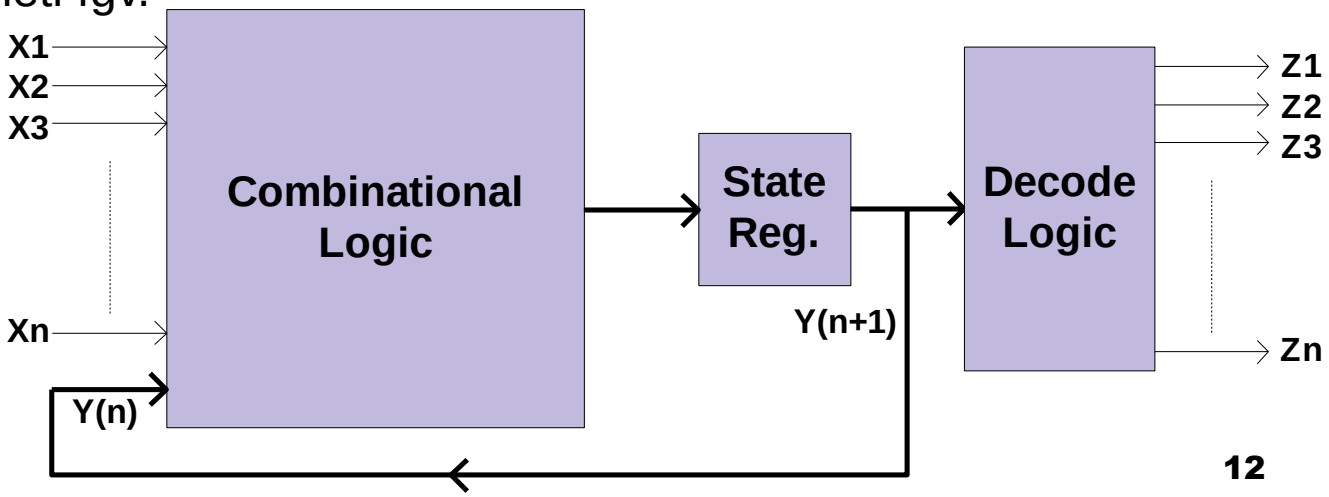
- A sorrendi hálózatok egyik alapmodellje. Késleltetés: a kimeneten az eredmény véges időn belül jelenik meg! Korábbi értékek visszacsatolódnak a bemenetre: kimenetek nemcsak a bemenetek pillanatnyi, hanem a korábbi állapotoktól is függenek. Problémák merülhetnek fel az állapotok és bemenetek közötti szinkronizáció hiánya miatt (változó hosszúságú kimenetet - dekódolás). Ezért alkalmazzuk legtöbb esetben a második, Moore-féle automata modellt.
- Három halmaza van: (Visszacsatolni az állapotregisztert a késleltetés miatt kell)
  - X – a bemenetek,
  - Z – a kimenetek,
  - Y – az állapotok halmaza.
- Két leképezési szabály a halmazok között:
  - $\delta(Xn,Yn) \rightarrow Yn+1$: következő állapot fgv.
  - $\mu(Xn,Yn) \rightarrow Zn$: kimeneti fgv.



2.) Moore-modell

- A kimenetek közvetlenül csak a pillanatnyi állapottól függenek (bemenettől függetlenek). Tehát a kimenetet nem a bemenetekhez, hanem az állapotoknak megfelelően szinkronizáljuk.
- Három halmaza van:
  - X – a bemenetek,
  - Z – a kimenetek,
  - Y – az állapotok halmaza.
- Két leképezési szabályok
  - $\delta(Xn,Yn) \rightarrow Yn+1$: köv. állapot fgv.
  - $\mu(Yn) \rightarrow Z$: kimeneti fgv.

Input ⇒ Next-State ⇒ Present-State ⇒ Output



Adatút (data-path) tervezése FIR szűrőhöz

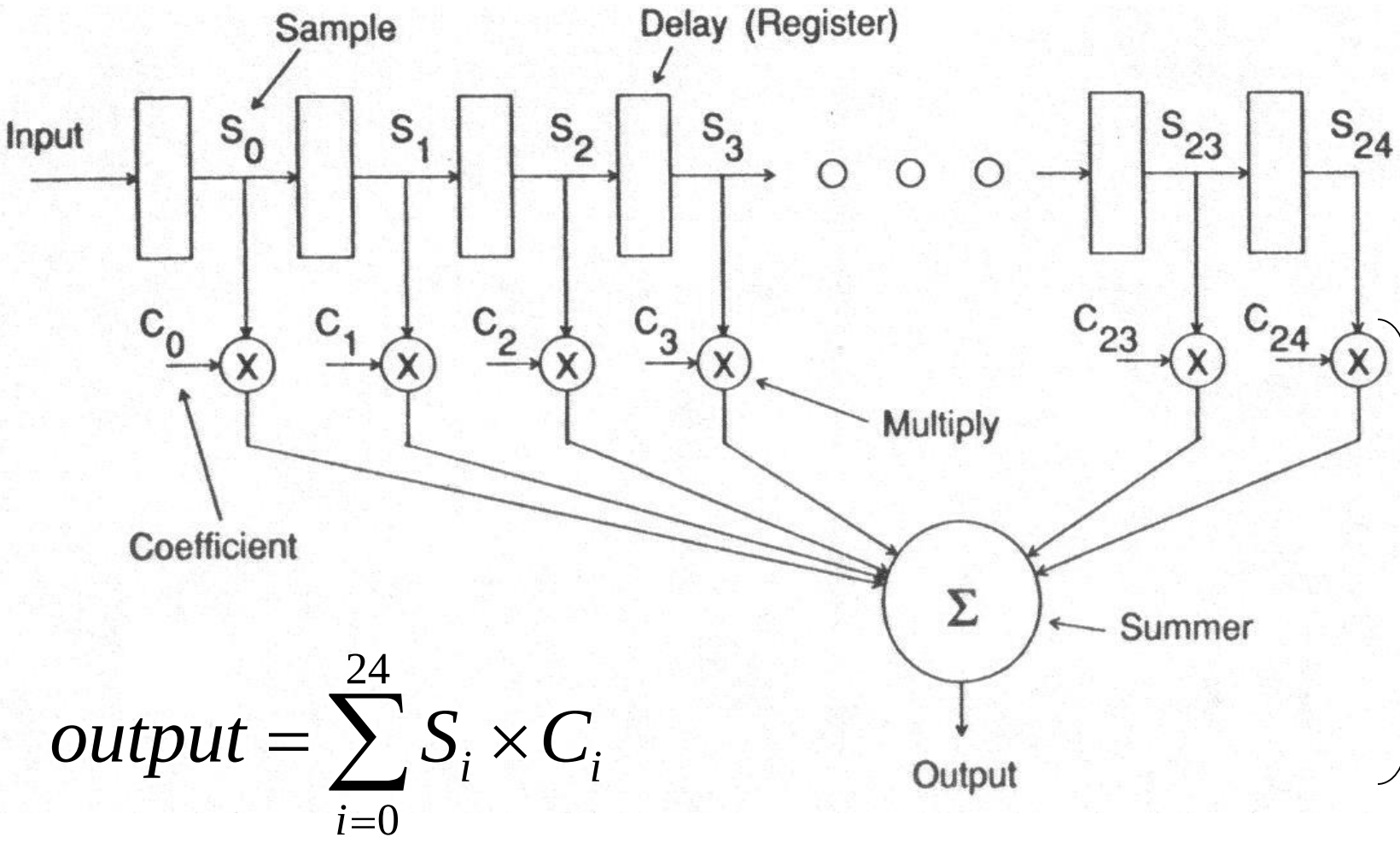
Adatút tervezés - FIR szűrő

- **FIR = Finite Impulse Response** szűrővel egy digitális rendszerben könnyen kiszámíthatók pl. skaláris/ *belső szorzat*, mátrixműveletek stb. Alkalmazzák a digitális jelfeldolgozásban (DSP) áteresztő szűrőknél. **Feladat:** tervezzünk egy FIR szűrőt, amely 25 együtthatót képes kezelni, és a kimenetét képező eredményt a következő egyenlettel kapjuk meg:

$$output = \sum_{i=0}^{24} S_i \times C_i$$

- ahol *Si* a bemeneti adatfolyam i. mintája, ill. *Ci* az i. konstans értéke. A/D konverterrel állítja elő a mintákat, ill. D/A konverterrel alakítja vissza analóg jellé, további feldolgozás szempontjából elfogadható formába. Minden egyes mintát meg kell szorozni a neki megfelelő együtthatóval (koefficiens), majd a szorzatokat össze kell adni. Egyenként 25 mintát veszünk késleltetve, (elvileg 25 különböző szorzó áramkör kellene). De ezt a feladatot a MAC egység látja el.

Példa: FIR szűrő - feldolgozás



MAC:

Multiply & Accumulate

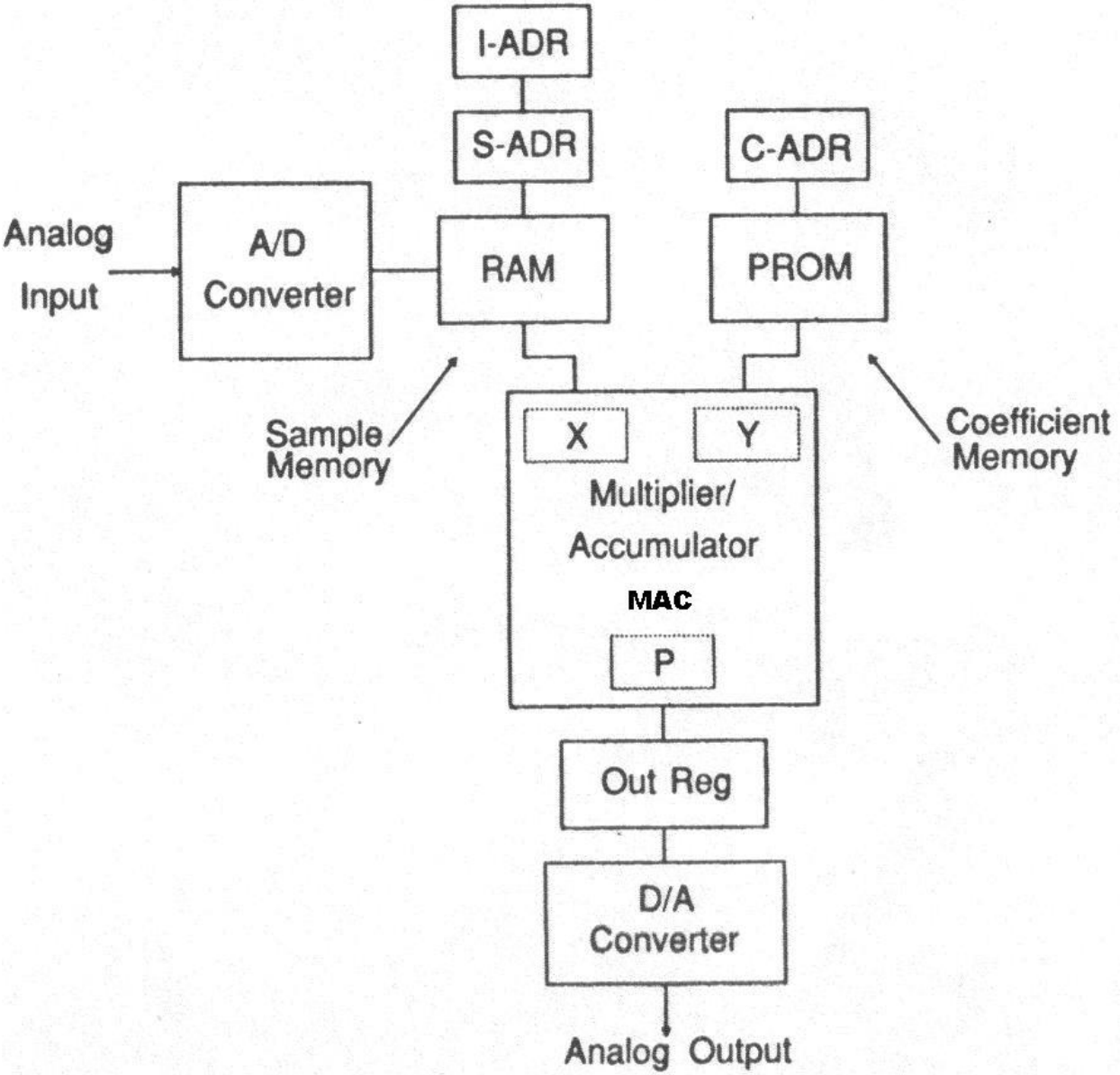
(1 clk)

$$output = \sum_{i=0}^{24} S_i \times C_i$$

Példa: FIR szűrő felépítése

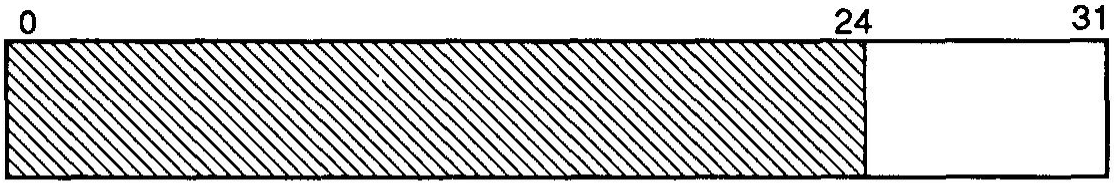
- **IMAC: (Multiplier/Accelerator):** a szorzást, és az összeadást egy órajel ciklusban képes elvégezni. Három regiszterből áll: két bemeneti (X és Y) és egy kimeneti regiszterből (P).
- **Együttható- memória (Coefficient memory):** C_MEM tárolja a Ci konstansok értékeit. Ez egy kisméretű PROM memória.
- **Együttható- memóriacímző regiszter: (C_ADDR)** egy számláló, amely a következő együtthatót azonosítja. 0-ról indul minden egyes iterációkor és az együtthatók címei egyesével inkrementálódnak.
- **Minta-memória (S_MEM):** az éppen aktuális és az azt megelőző 24 mintát tároljuk el. RAM, amely 32 értékből csak 25-t használ.
- **Minta- memóriacímző regiszter: (S_ADDR)** Ez egy számláló, amely az aktuális minta címét inicializálja, ezután pedig a (sorrendben) előző minta címére inkrementálódik (mutat)
- **Kezdő minta címregiszter (I_ADDR):** azonosítja az algoritmus kezdőpontját mindenegyes átmenetnél (akt. iteráció kezdőcíme mindig 1-el kevesebb lesz)
- **A/D konverter: (ADIN)** Minden iterációban ezen az egységen keresztül kapjuk az új adatot.
- **D/A konverter: (DAOUT)** A kimeneti regiszterből kapja az adatot, és analóg jellé alakítja, a további feldolgozás számára elfogadható formában.
- **Kimeneti regiszter: (OUT)** MAC-ből jövő értéket tárolja és a D/A konverternek továbbítja.

Példa: FIR szűrő blokkszintű felépítése



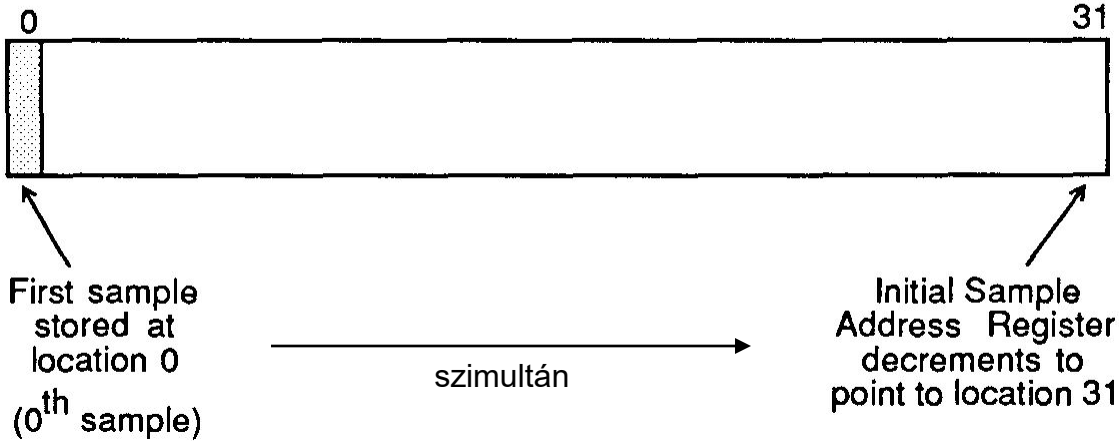
Koefficiens- és minta-adat tárolása

- Mindkettő C_MEM, S_MEM is 32 elemű tárolók



Coefficients stored in first 25 locations of PROM (0…24)

- Első betöltött minta után:

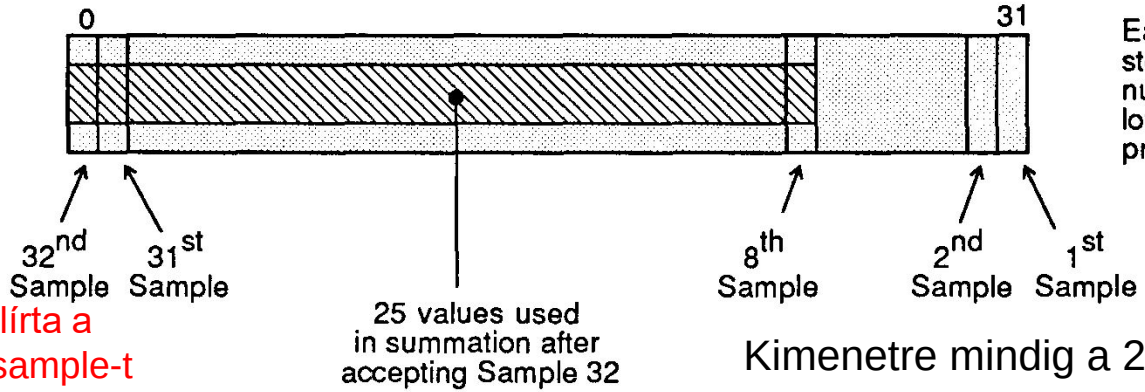


Samples start at location 0 of RAM (0-…)

I_ADR-- dekrementálása (31-…)

Koefficiens és adat tárolás (folyt)

- 33 betöltött minta után:

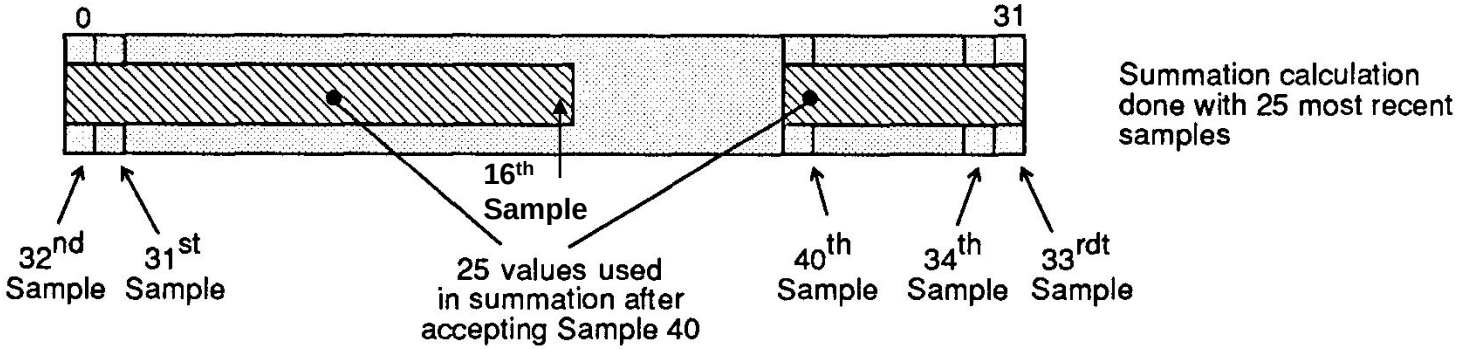


(31,30,…,0)

Felülírta a 0th sample-t

Kimenetre mindig a 25 **legutolsó** érték kerül!

- 40 betöltött minta után:



FIR - RTL leírása

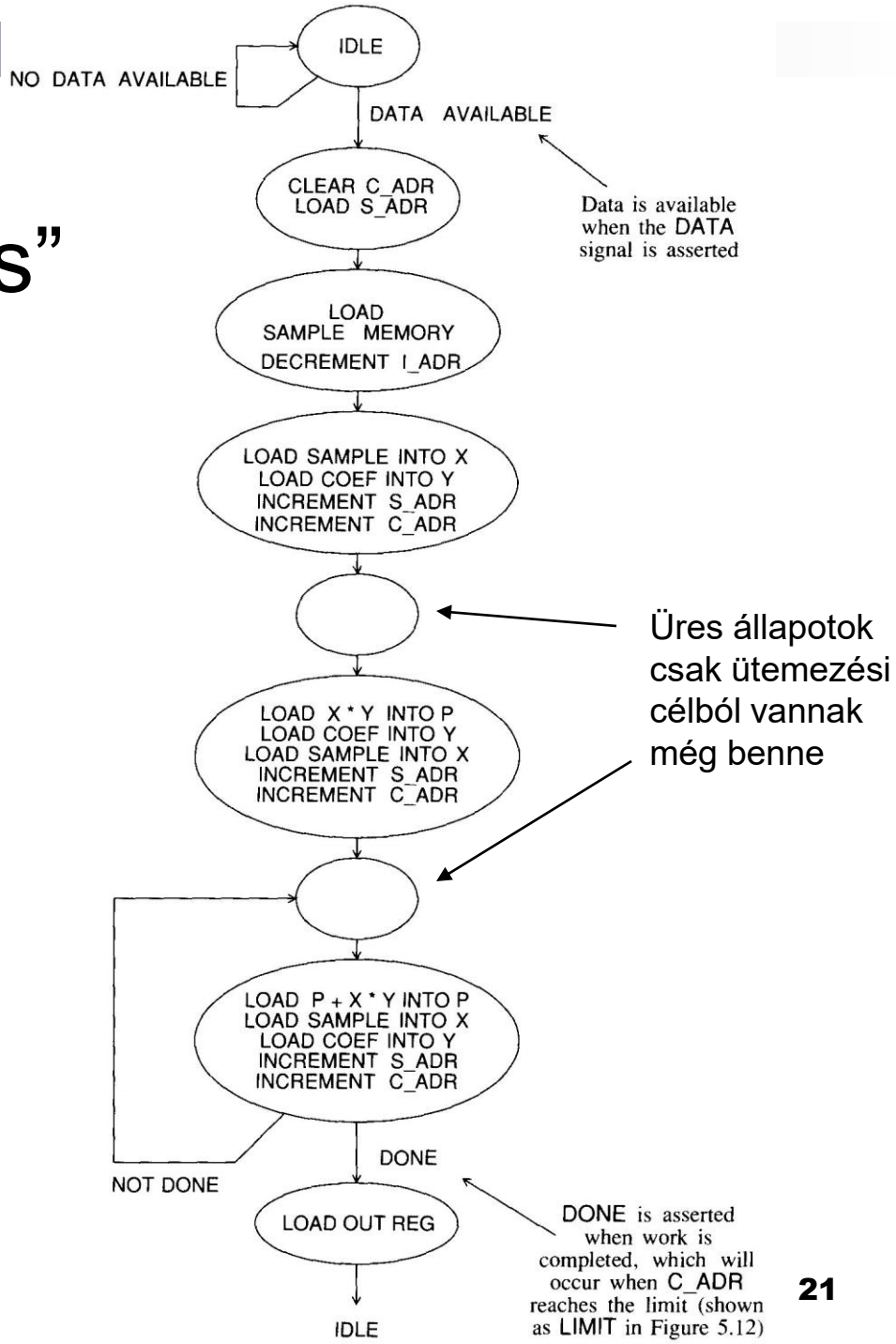
```
start: if (ADIN not ready)   goto start          Check input data.
                                                 Várakozik..., új adat érkezéséig

       0          →  C_ADR                       Clear coefficient address.
       I_ADR      →  S_ADR                       Load sample address.

       ADIN       →  S_MEM[S_ADR]                Data to sample memory.
       I_ADR − 1  →  I_ADR                       Decrement initial address.
                                                 Szimultán betöltés és dekrementálás

       S_MEM[S_ADR]  →  X                        Load sample to X reg.
       C_MEM[C_ADR]  →  Y                        Load coefficient to Y reg.
       S_ADR + 1     →  S_ADR                    Increment sample address.
       C_ADR + 1     →  C_ADR                    Increment coefficient address.
                                                 Első minta (X) és első koefficiens (Y) betöltése a MAC bemenetére

       X × Y         →  P                        Load product register.
       S_MEM[S_ADR]  →  X                        Load sample to X reg.
       C_MEM[C_ADR]  →  Y                        Load coefficient to Y reg.
       S_ADR + 1     →  S_ADR                    Increment sample address.
       C_ADR + 1     →  C_ADR                    Increment coefficient address.
                                                 Szorzat (P) számításával egyidejűleg új mintát (X) és új koefficienst (Y) töltünk be

over:  P + X × Y     →  P                        Add product into P register.
       S_MEM[S_ADR]  →  X                        Load sample to X reg.
       C_MEM[C_ADR]  →  Y                        Load coefficient to Y reg.
       S_ADR + 1     →  S_ADR                    Increment sample address.
       C_ADR + 1     →  C_ADR                    Increment coefficient address.

       if (not done)   goto over                 Repeat for all samples.
                                                 Work is done when C_ADDR is 25.
                                                 Iterációs ciklus, amíg „not Done”, tehát nem érjük el az utolsó mintát!

       P  →  OUT                                 Update output value.

              goto start                         Start over.
```

RTL leírásnak megfelelő „előzetes” állapotdiagram

Vezérlési szekció: adatátvitel nem azonnal, hanem véges idő alatt megy végbe!

Ezt a tervezőnek kezelnie kell tudni (előzetes diagramm).

RTL leírásból képzett diagramnak lehetnek még *üres* állapotai is, azonban később az optimalizálás, véglegesítés során ezek kiesnek – egyszerűsödnek!



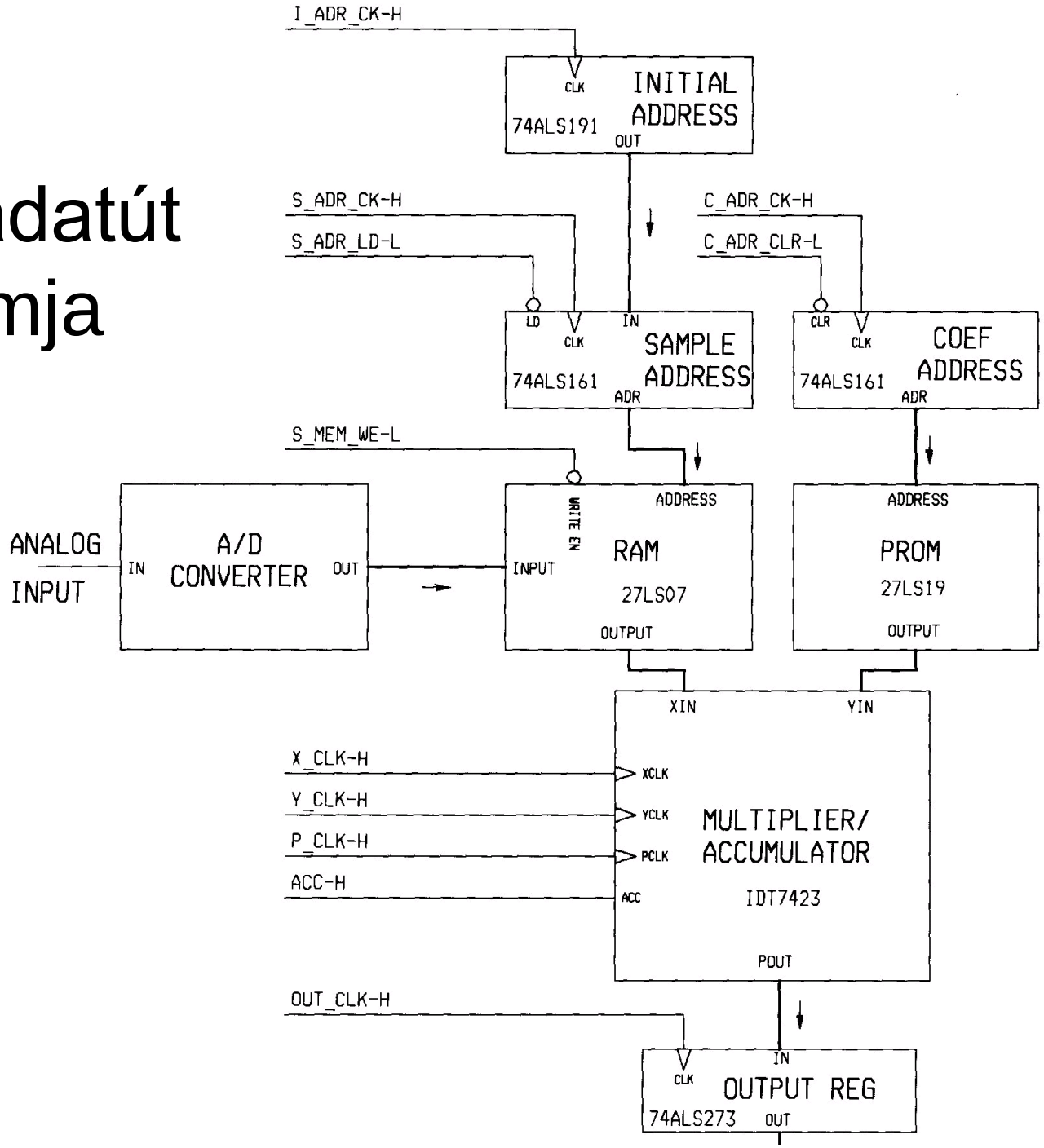
FIR szűrő „részletes” adatút blokkdiagramja

Végleges adatút diagramm:

RTL leírás + előzetes diagramm alapján kapjuk!

Itt már a **vezérlő jelek** is fel vannak tüntetve!

További tervezői feladatunk lesz ezeknek a vezérlő-jeleknek a megfelelő ütemben való generálása!



Vezérlő egység tervezése egyszerű állapotgép (FSM) segítségével – FIR szűrőhöz

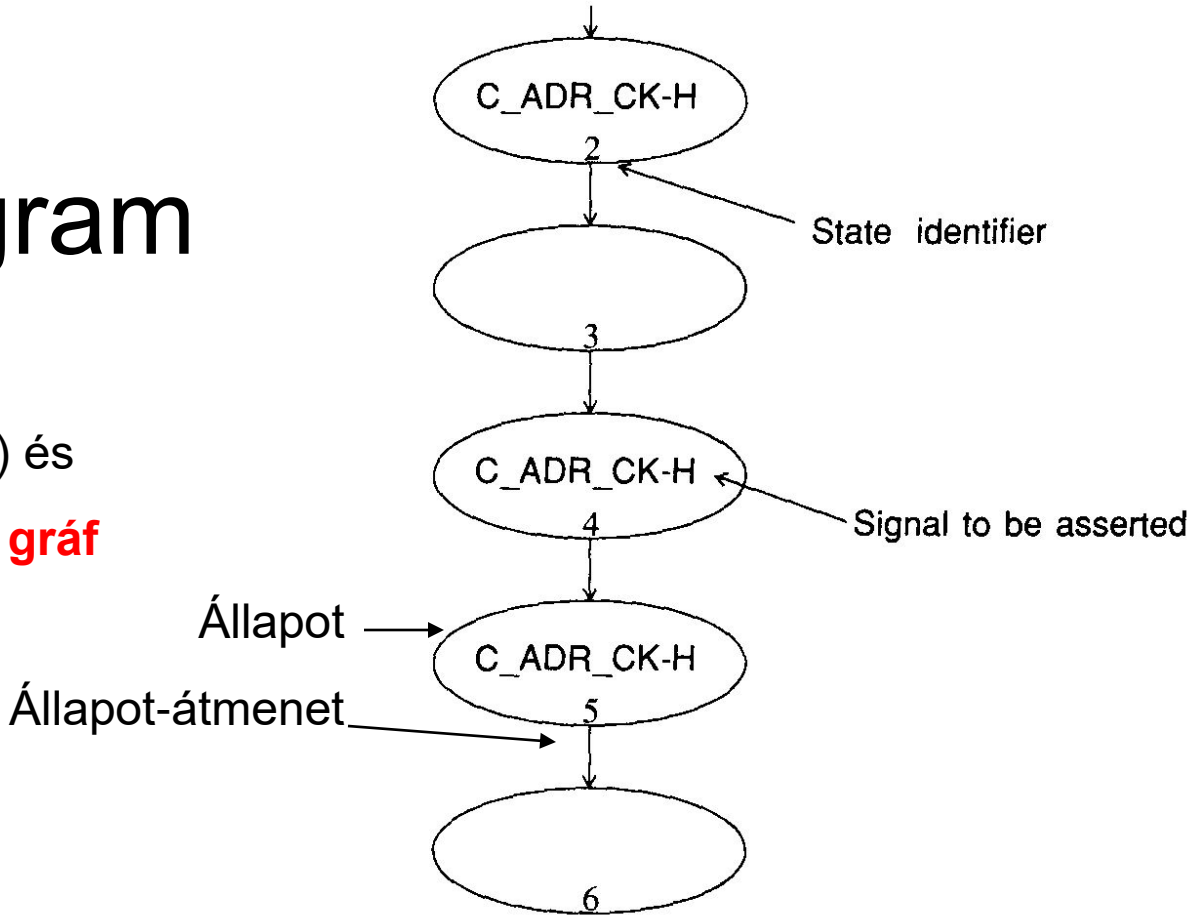
Tervezői feladatok (FIR szűrő):

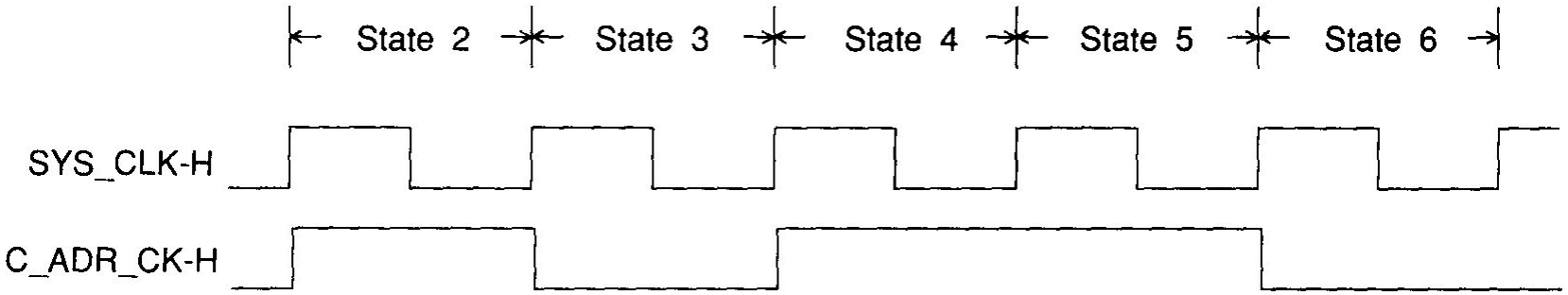
- Eddig rendelkezésre áll:
  - RTL leírás,
  - Előzetes és
  - részletes blokk diagram
- Most: a vezérlési rész tervezése következhet
  - 1.) Vezérlő jelek beállítása (assertion levels)
  - 2.) „Mapping” (hozzárendelés): a Moore modellt vesszük alapul, amelyhez a Véges Állapotú Automatát (FSM) próbáljuk illeszteni / hozzárendelni.

Példa: Állapot diagram

Gráf reprezentáció:

- (DFG: Adat-folyam gráf) és
- **CFG: Vezérlés-folyam gráf**



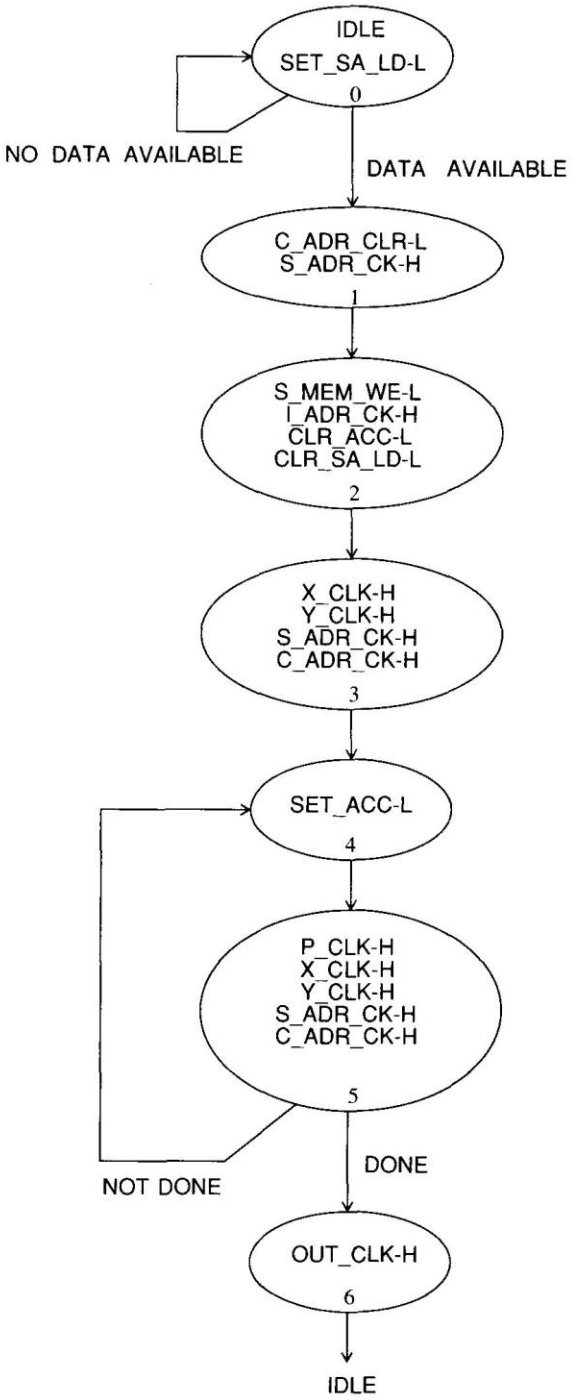


1.) FIR szűrő részletes FSM állapot diagrammja

SET-RESET FF-kal:

- Set S_ADR_LD (State0)
- Reset / Clear S_ADR_LD (State2)
- Reset ACC_L (State2)
- Set ACC_L (State4)

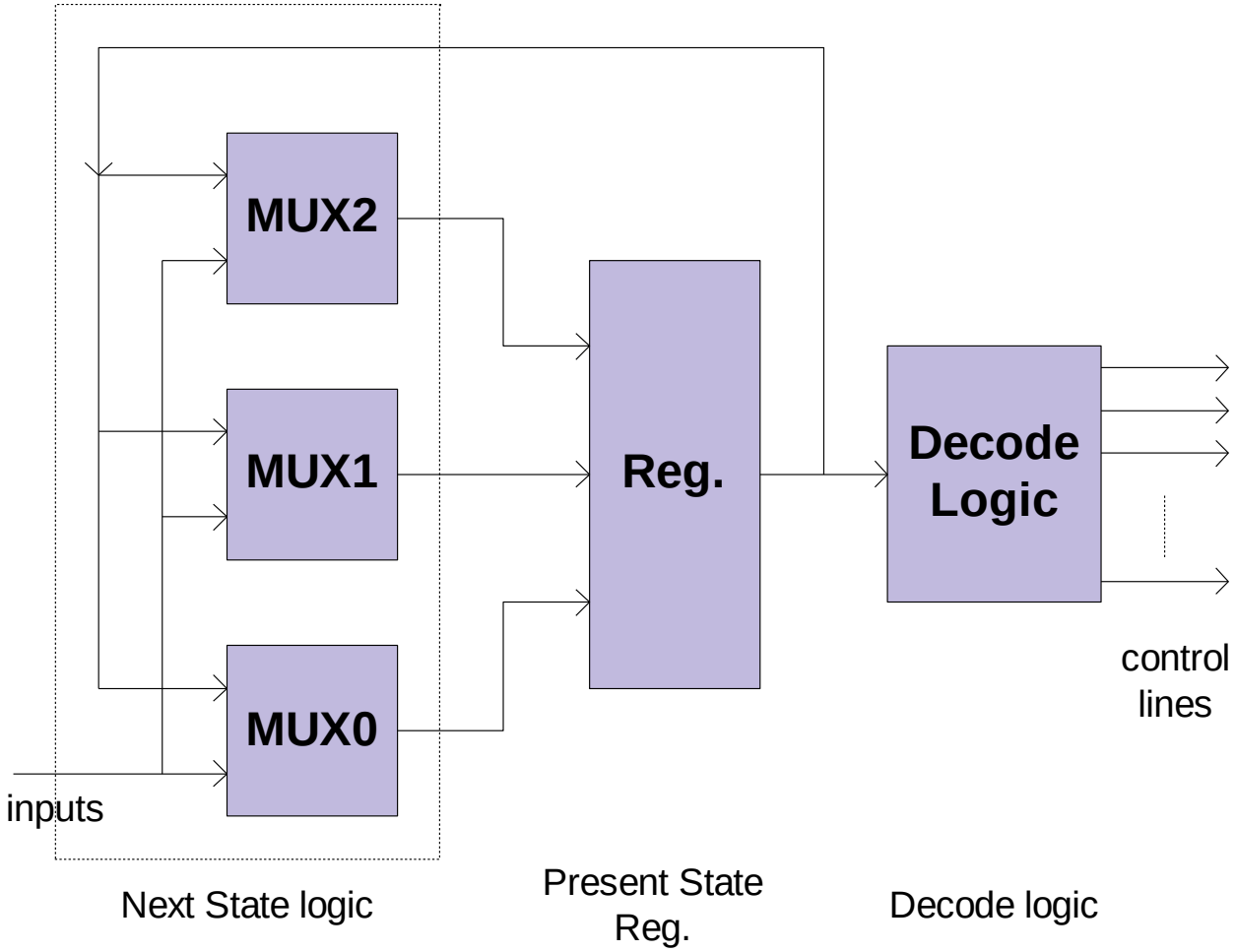
- Vezérlés folyam: $State_0$, … ,$State_6$



2.) „Mapping”: FIR szűrő vezérlőjének multiplexeres megvalósítása

- **Present State Reg**: aktuális állapotot tárolja
- **Next State Logic**: következő állapot kiválasztása a bemenetek, és a visszacsatolt állapotoktól függően (multiplexerek!)
- **Output Logic**: az aktuális állapot dekódolt formája kerül a kimenetekre (vezérlőjel generálás)

FIR szűrő vezérlőjének multiplexeres megvalósítása



FIR szűrő vezérlőjének multiplexeres megvalósítása

- **További két input signal**
  - DATA-H: A/D felől új feldolgozandó adat érkezett (flag)
  - DONE-H: szükséges számú iteráció végrehajtódott

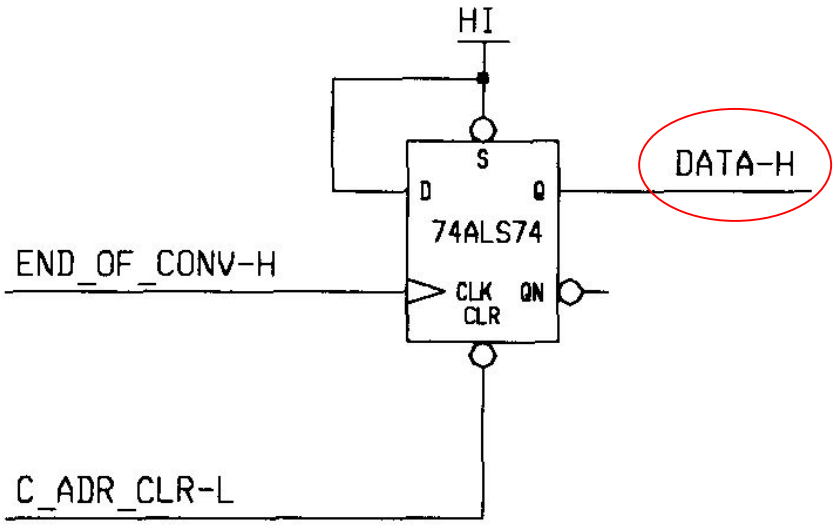
Table 5.1. Next State Multiplexer Specifications.

	MUX 2	*MUX 1*	*MUX 0*
State 0	0	0	DATA-H
State 1	0	1	0
State 2	0	1	1
State 3	1	0	0
State 4	1	0	1
State 5	1	DONE-H	0
State 6	0	0	0

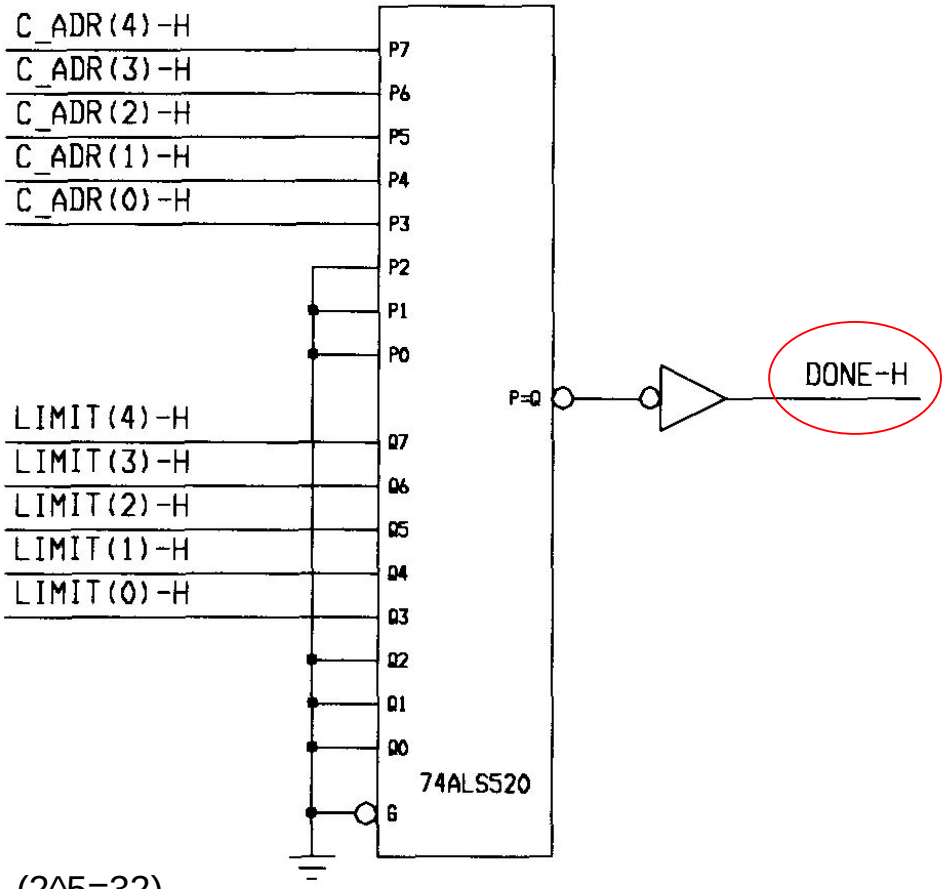
FIR szűrő részletes FSM állapot-diagram alapján kapjuk a táblázatot!

FSM vezérlőjelei

DATA-H



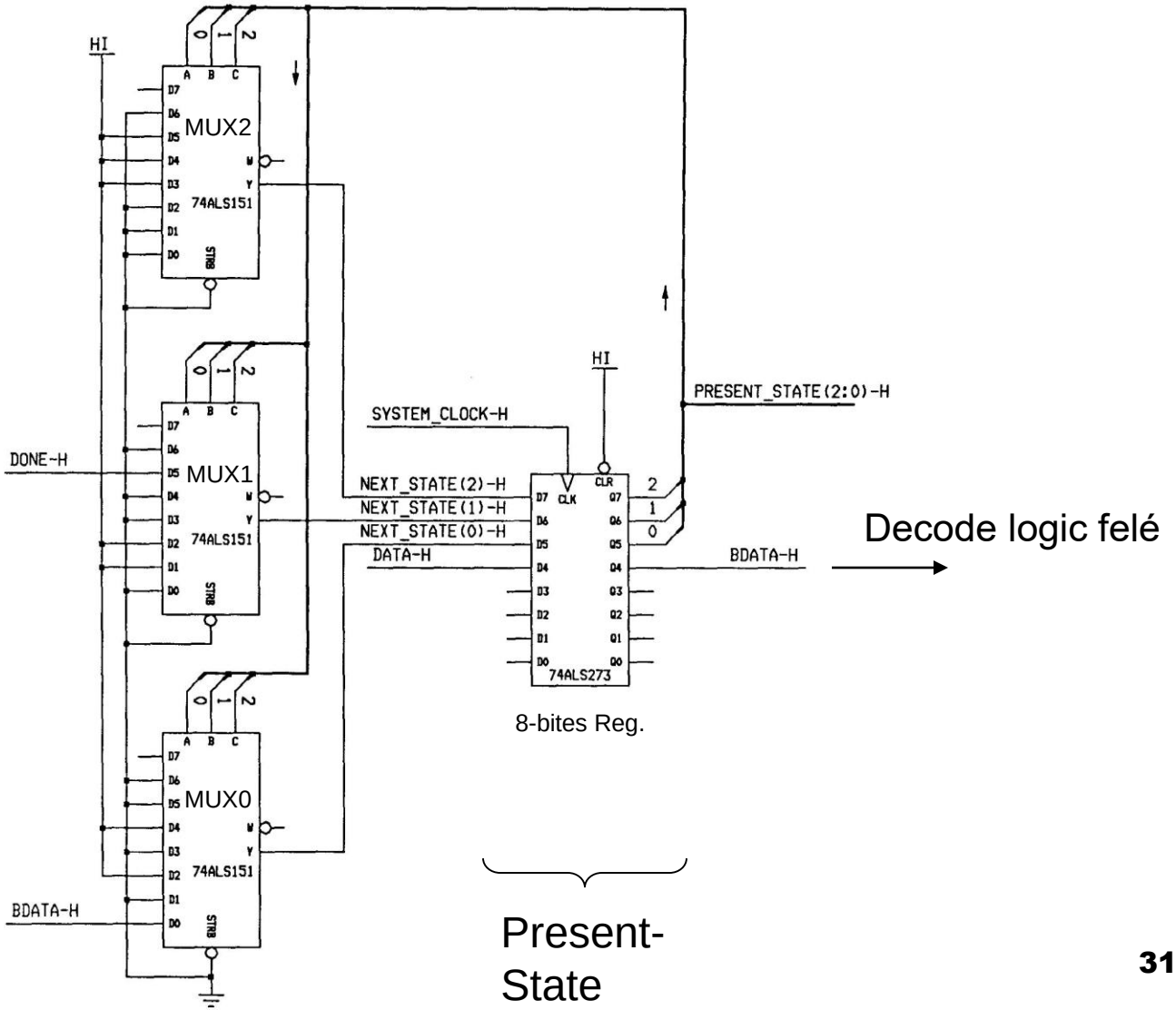
DONE-H



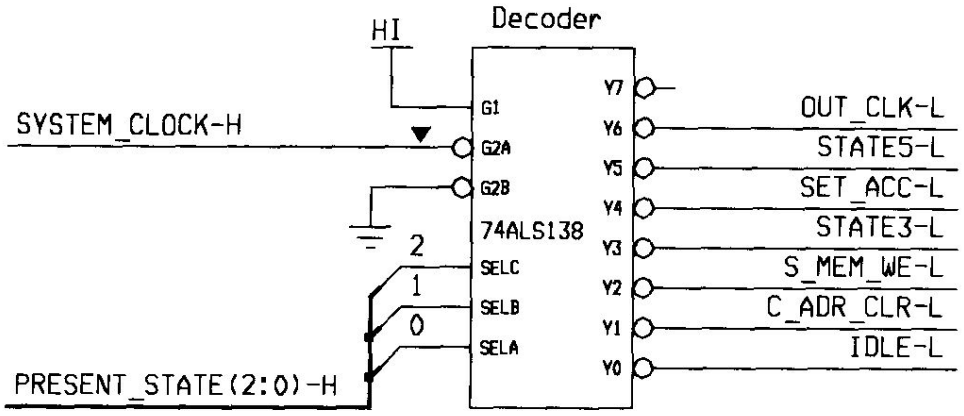
(2^5=32)

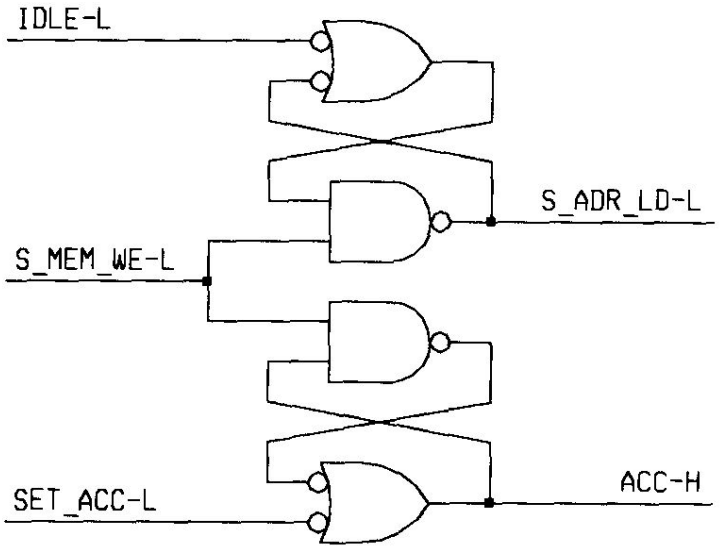
komparátor

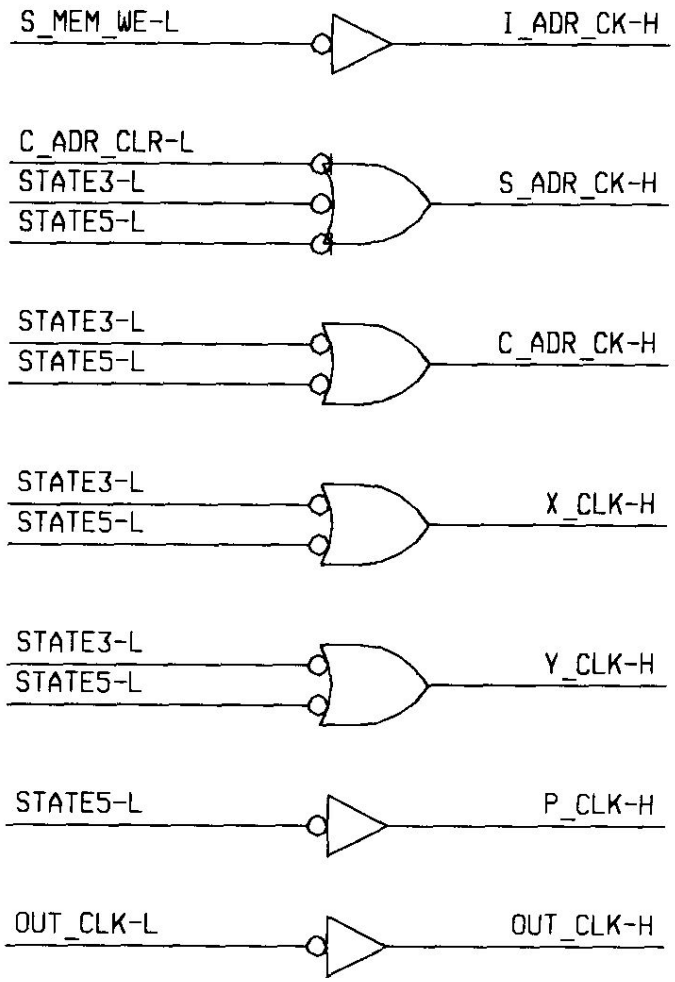
FSM vezérlő egységének Next-State és Present-State logikája (blokk diagramm)



Present-State logikai áramkörei





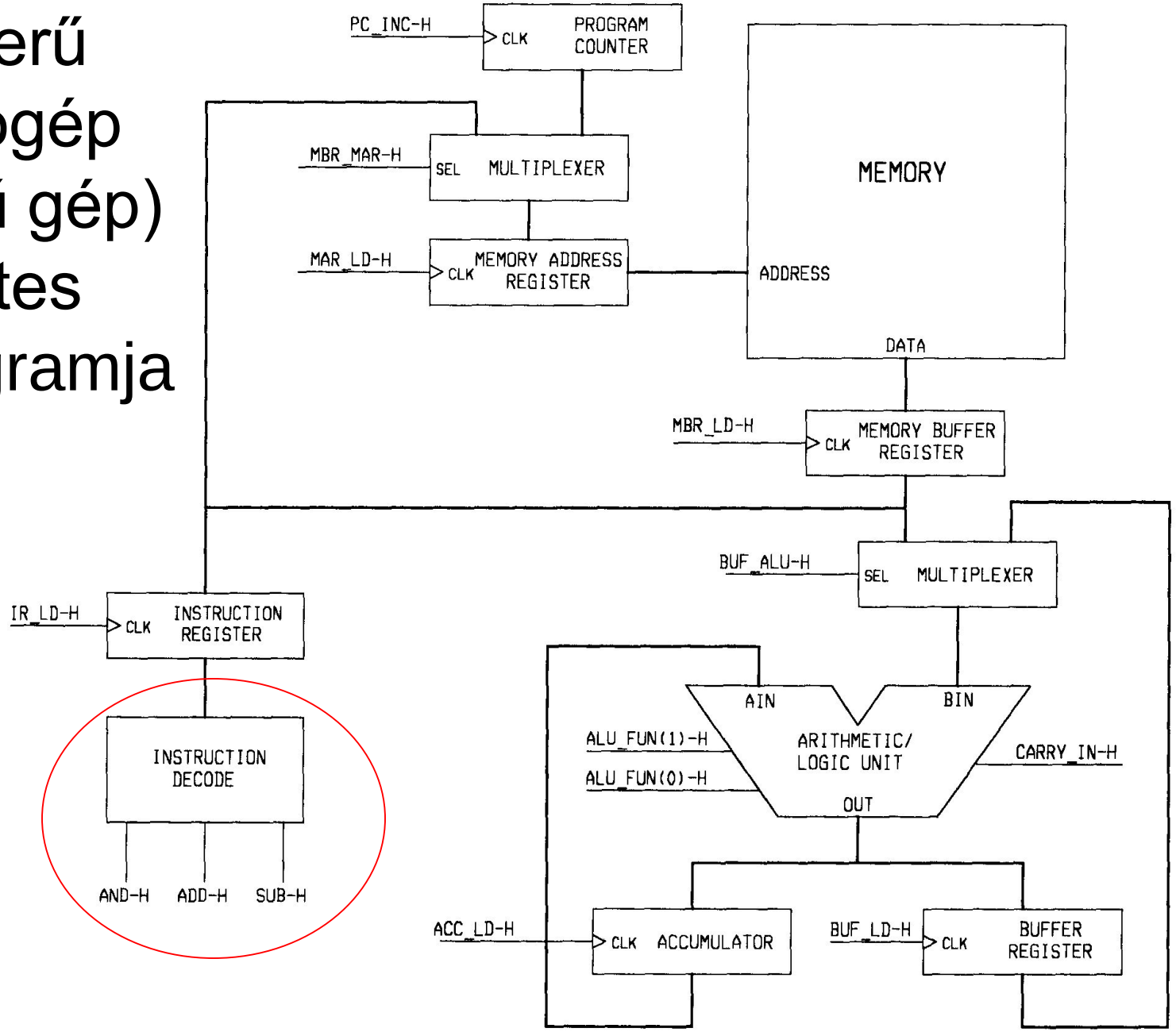


Szekvenciális vezérlő rendszerek – egyedi késleltetéses módszer

Szekvenciális vezérlő rendszerek

- Ahelyett – mint azt az állapotgépnél (FSM) láthattuk – hogy egyetlen regiszter tömbben tárolnánk a rendszer állapotait, ebben az esetben *külön regisztereket* definiálunk egy egyszerű számítógép (egycímű gép) blokkdiagramját felhasználva.
- Regiszter-transzfer műveletek sora mutatja be ennek a *késleltetéses* vezérlő rendszernek a működését.

Egyszerű számítógép (egycímű gép) részletes blokkdiagramja



Példa: 3 egyszerű utasítás

- ADD
- SUB
- AND

Példa: Egycímű gép vezérlési funkciója

- Mivel példaként három egyszerű két-operandusú utasítást (AND, ADD, SUB) akarunk végrehajtani egy egycímű gépen, ezért a második operandus értékét az ACC-ből kell betölteni!
- Ehhez az ALU néhány alapvető funkciója:

Késleltetések!

T_{REG}= 40ns

T_{MEM}= 200ns

ALU_FUN		OUT function	
0	0	bitenkénti AND (A_In, B_In)	[40 ns]
0	1	bitenkénti OR (A_In, B_In)	[40 ns]
1	0	inverz NOT (B_In)	[40 ns]
1	1	bináris ADD (A_In, B_In)	[80 ns]

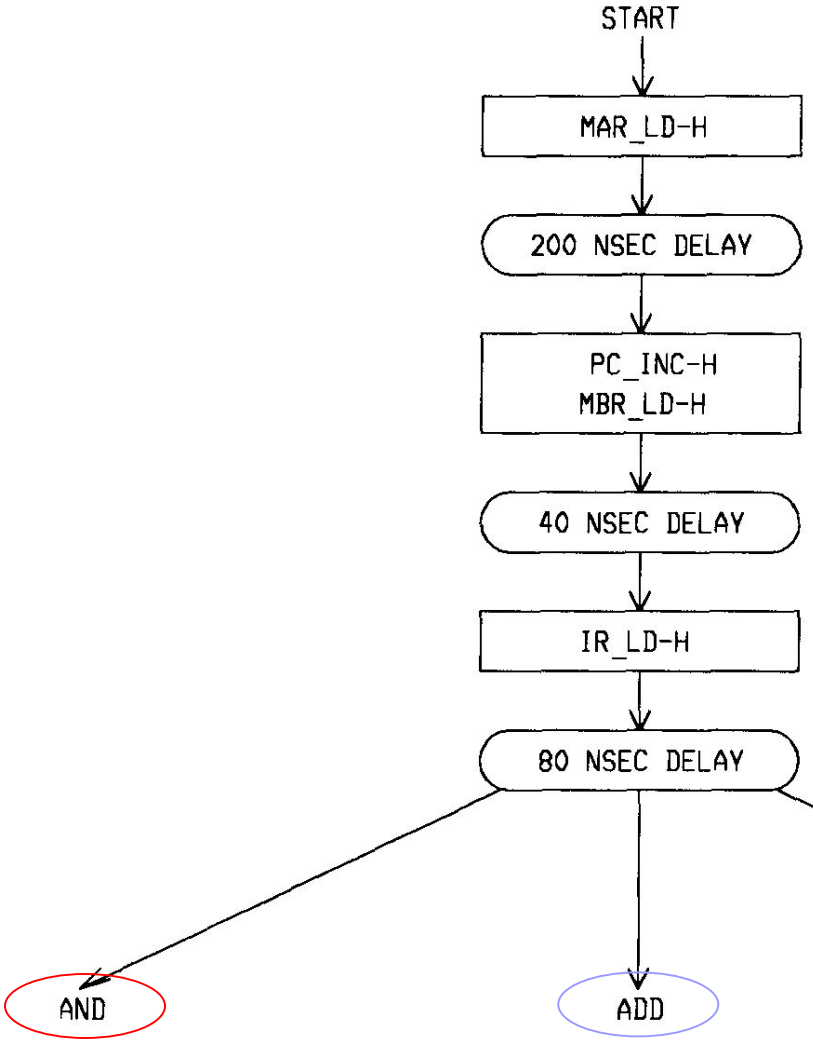
Három utasítás RTL leírása

Table 5.2. Register Transfers for Three Instructions.

Register Transfers for Example

AND *Instruction*	ADD *Instruction*	SUBTRACT *Instruction*
PC → MAR	PC → MAR	PC → MAR
M [MAR] → MBR	M [MAR] → MBR	M [MAR] → MBR
PC + 1 → PC MBR → IR	PC + 1 → PC MBR → IR	PC + 1 → PC MBR → IR
MBR → MAR	MBR → MAR	MBR → MAR
M [MAR] → MBR	M [MAR] → MBR	M [MAR] → MBR
MBR • ACC → ACC	MBR + ACC → ACC	¬ MBR → BUF (2's Comp)
		BUF + ACC + 1 → ACC (2's Comp)

Három utasítás késleltetéses folyamat ábrája (delay flowchart)



Utasítás Fetch:

$$PC \rightarrow MAR$$

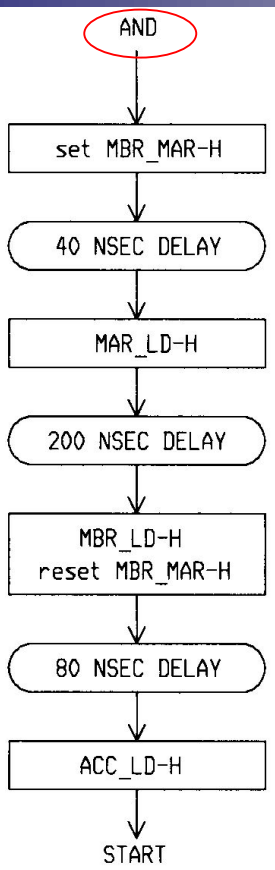
$$M[MAR] \rightarrow MBR$$

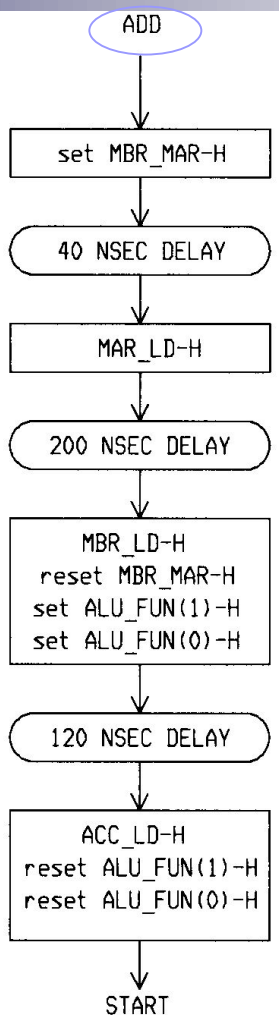
$$PC + 1 \rightarrow PC$$

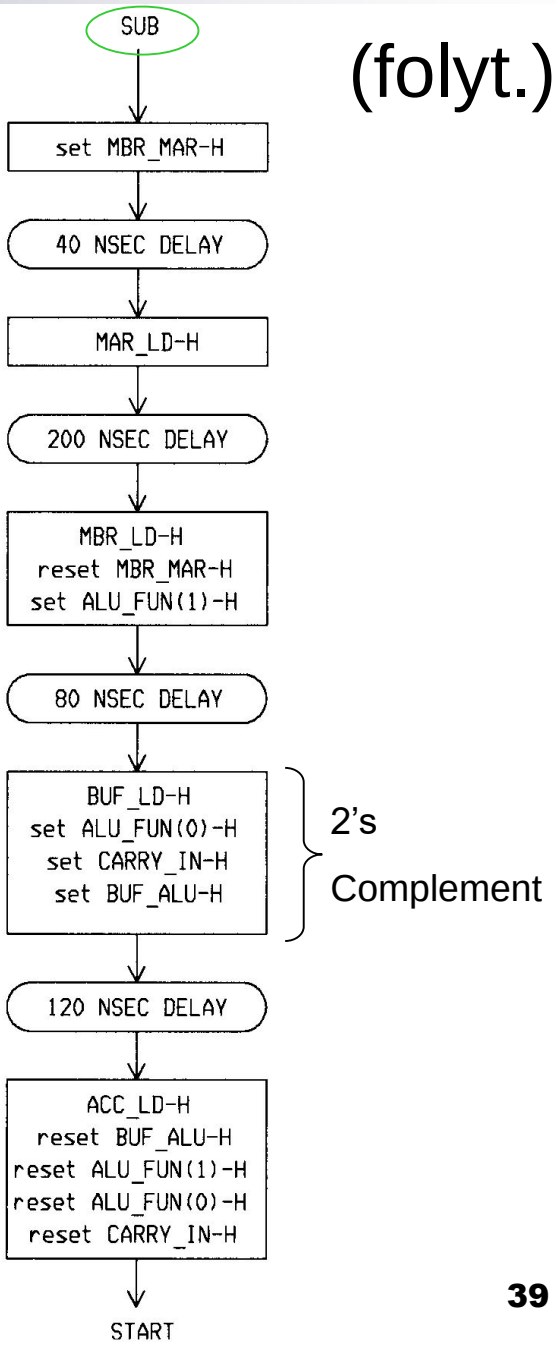
$$MBR \rightarrow IR$$

Mindhárom esetben ugyanaz.

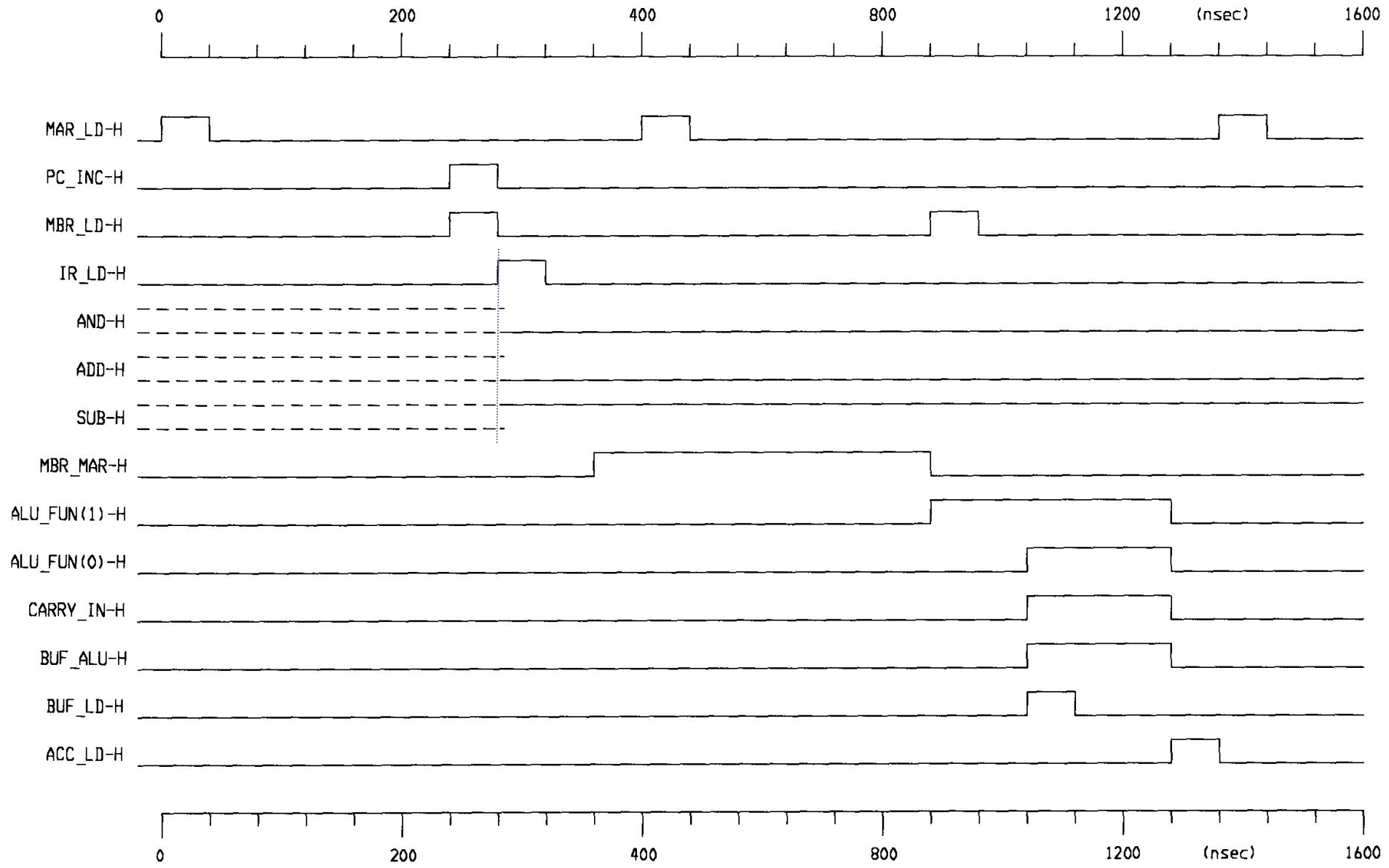
(folyt.)


AND
set MBR_MAR-H
40 NSEC DELAY
MAR_LD-H
200 NSEC DELAY
MBR_LD-H
reset MBR_MAR-H
80 NSEC DELAY
ACC_LD-H
START

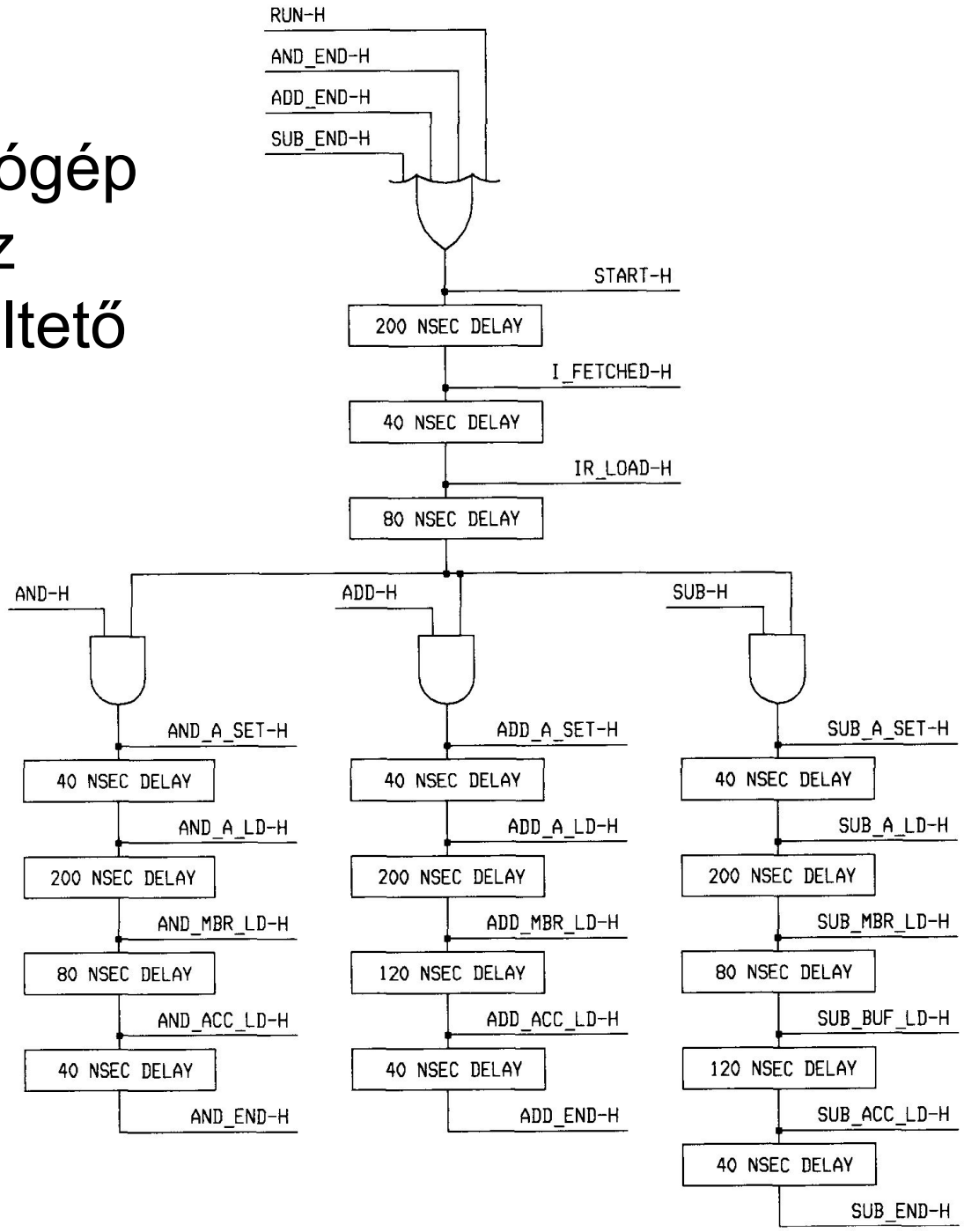

ADD
set MBR_MAR-H
40 NSEC DELAY
MAR_LD-H
200 NSEC DELAY
MBR_LD-H
reset MBR_MAR-H
set ALU_FUN(1)-H
set ALU_FUN(0)-H
120 NSEC DELAY
ACC_LD-H
reset ALU_FUN(1)-H
reset ALU_FUN(0)-H
START


SUB
set MBR_MAR-H
40 NSEC DELAY
MAR_LD-H
200 NSEC DELAY
MBR_LD-H
reset MBR_MAR-H
set ALU_FUN(1)-H
80 NSEC DELAY
BUF_LD-H
set ALU_FUN(0)-H
set CARRY_IN-H
set BUF_ALU-H
2's
Complement
120 NSEC DELAY
ACC_LD-H
reset BUF_ALU-H
reset ALU_FUN(1)-H
reset ALU_FUN(0)-H
reset CARRY_IN-H
START

PI: SUB művelethez tartozó időzítési diagram



Egyszerű számítógép vezérléséhez szükséges késleltető elemek

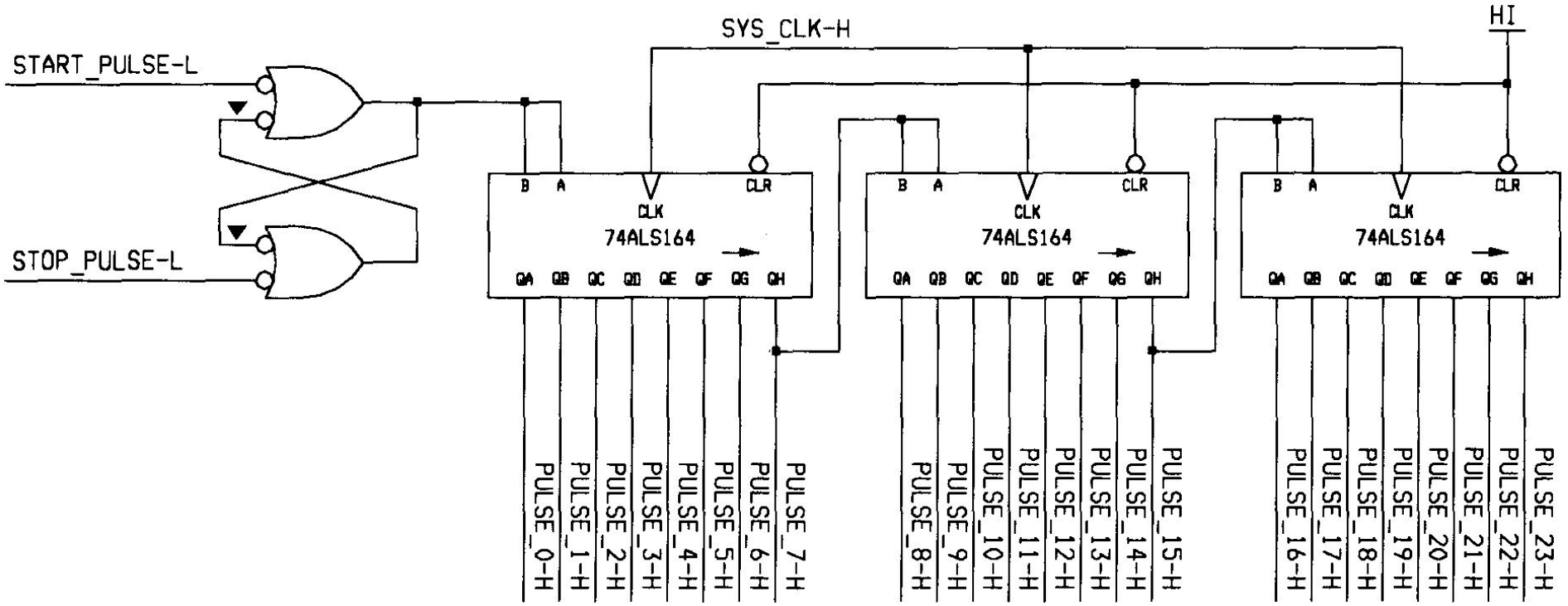


Szekvenciális vezérlő rendszerek tervezése Shift-regiszteres időzítővel

Vezérlő Shift-regiszteres időzítővel

- Ez a megvalósítás az egyedi késleltetéses módszerhez nagyban hasonlít, ugyanis:
  - *Adatút-diagrammot* használunk a vezérlőjelek azonosítására,
  - *Folyamat-diagrammot* a regiszter-transzferek (RTL) ábrázolására, míg
  - *Idő-diagrammot* a vezérlőjelek kölcsönhatásának leírására.
- Három 8-bites Shift-regiszter segítségével generálja az impulzusokat (közvetve a vezérlő jeleket is).

Sorrendi vezérlő egységek megvalósítása Shift-regiszterekkel



Vezérlő impulzusok

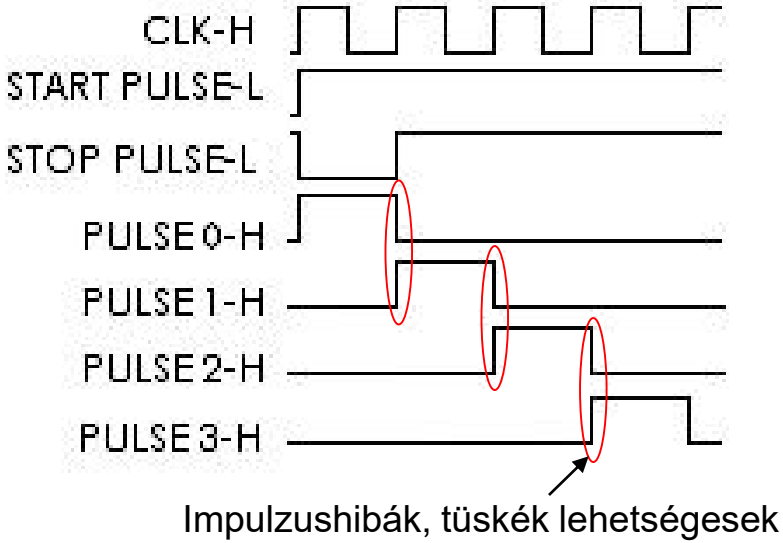
1. módszer: Nem-átlapoló impulzusok

- Inicializáláskor a kívánt impulzus előállítását a START_PULSE_L beállításával érhetjük el, amelyet a teljes folyamat végéig „L” alacsony-aktív szinten tartunk. A következő órajelciklusban a PULSE_0-H jel állítódik be magas jelszintre *rövid 40 ns-os impulzus* ideig.
- A STOP_PULSE-L vezérlőjelet a PULSE_0-H jel negálásával kapjuk meg (abban az esetben, ha egy ciklus, azaz 40ns ideig tart). Az órajel minden egyes felfutó élére shiftelődik az impulzus. Ekkor nem lapolódnak át, mivel egymás utáni 40ns –os részekből állnak össze, és a megfelelő PULSE_XX kimeneti vezérlőjelek OR kapcsolatából képződnek.

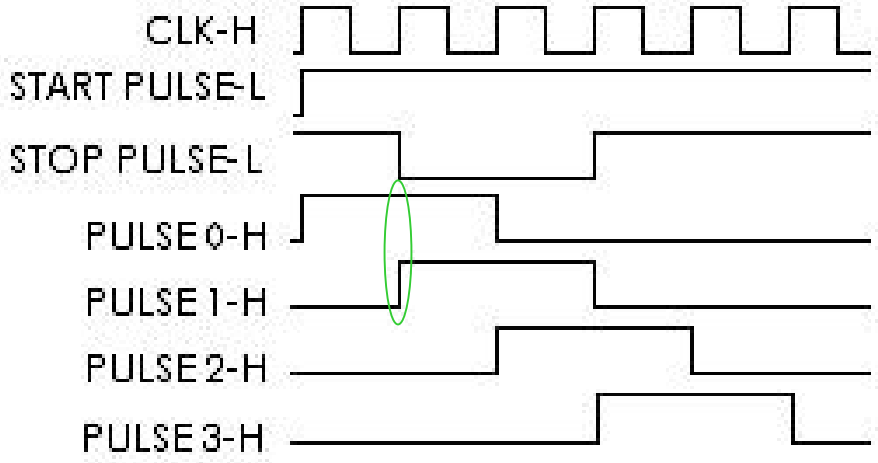
2. módszer: Átlapoló impulzusok

- Bizonyos esetekben azonban nem kívánt impulzushibák ún. tüskék (glitch) keletkezhetnek (pl. ha az egyik jel alacsony szinten marad, a másik viszont magas jelszintre vált). Ezeket a nem megfelelő jelváltásokat vagy SET-RESET flip-flopok használatával küszöbölhetjük ki, vagy pedig alkalmazni kell az **átlapoló impulzusok** technikáját.
- Ezt az idődiagramot a jobboldali ábra mutatja. Két egység *hosszú impulzusokat* (80ns) egyszerűen létrehozhatunk a PULSE_1-H jel és az invertált STOP_PULSE-L jel *OR* kapcsolatával (a bal oldali ábra jeleiből!). Az így kapott impulzus mentes lesz a hibáktól, és kiküszöbölhetők a hazárdjelenségek.

„Nem-átlapoló” és „átlapoló” impulzusok bemutatása időd iagramon



Nemátlapoló rövid impulzusok (40ns)



Átlapoló hosszú impulzusok (80ns)

Mikrokódos vezérlők – reguláris vezérlési struktúrák

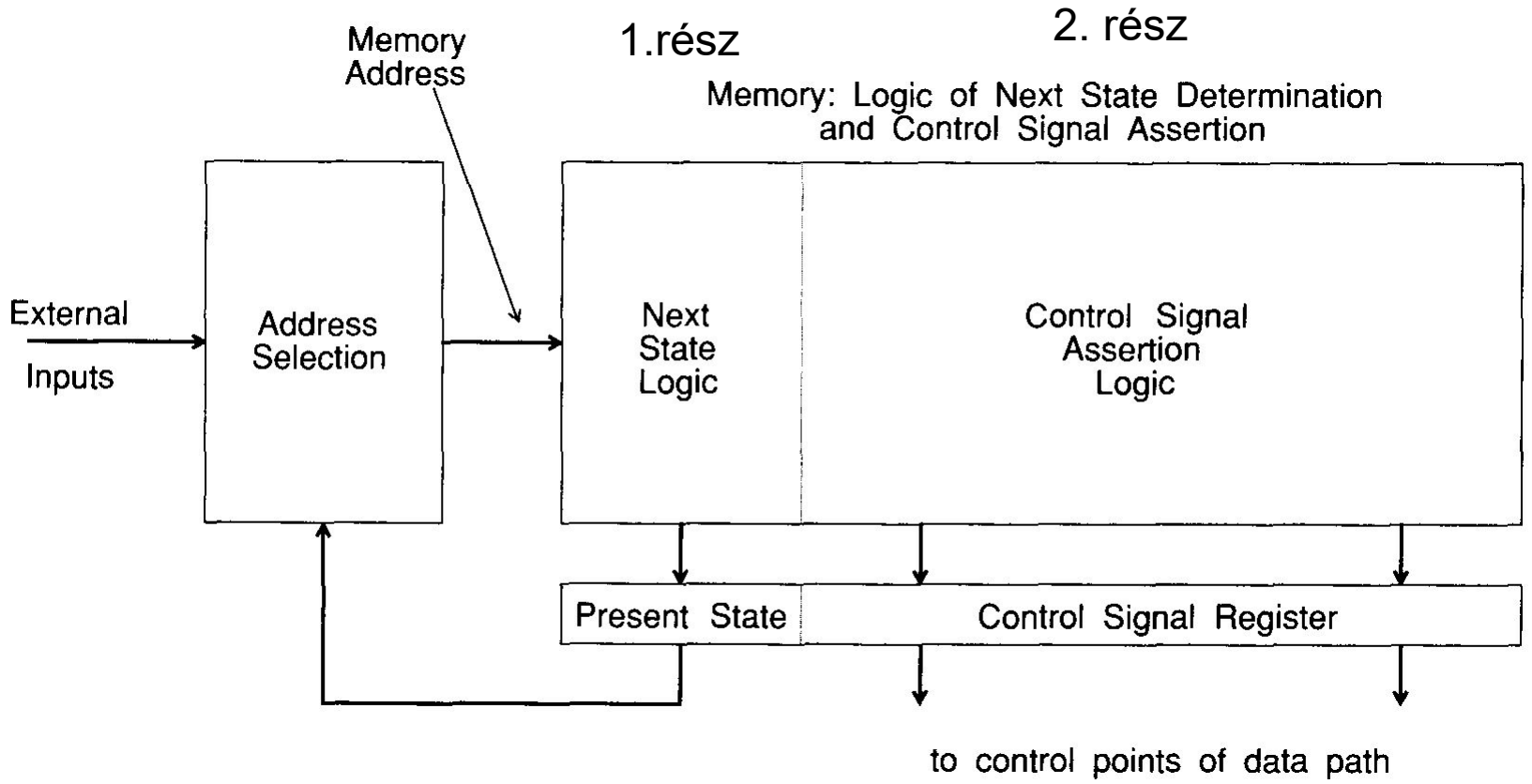
Ismétlés: Vezérlő egységek

- Általánosságban: a vezérlő egység feladata a memóriában lévő gépi kódú program utasításainak
  - értelmezése (decode),
  - részműveletekre bontása,
  - és ezek alapján az egyes funkcionális egységek vezérlése (**a vezérlőjelek megfelelő sorrendben történő előállítása**).

Klasszikus vagy reguláris módszer

- Eddig a „klasszikus”, késleltetéses módszereket tárgyaltuk (multiplexeres és shift-regiszteres példákkal). A rendszer tervezésekor, miután a feladat elvégzéséhez szükséges vezérlőjeleket definiáltuk, meg kell határozni a kiválasztásuk sorrendjét, és egyéb specifikus információkat (rendszer ismeret, tervezési technikák, viselkedési leírások - VHDL)
- Wilkes (1951): A komplex vezérlési folyamatokat **„reguláris módszerrel”** lehet egyszerűsíteni: nevezetesen **gyors memória elemeket** kell használni az utasítássorozatok tárolásánál. Állapotgépekkel (FSM) modellezik a vezérlő egység működését, és ezt a modellt transzformálják át mikrokódot használva. Az adatútvonal vezérlési pontjait memóriából (ROM) kiolvasott *vertikális-* vagy *horizontális-mikrokódú utasításokkal állítják be*!

FSM megvalósítása Memóriával

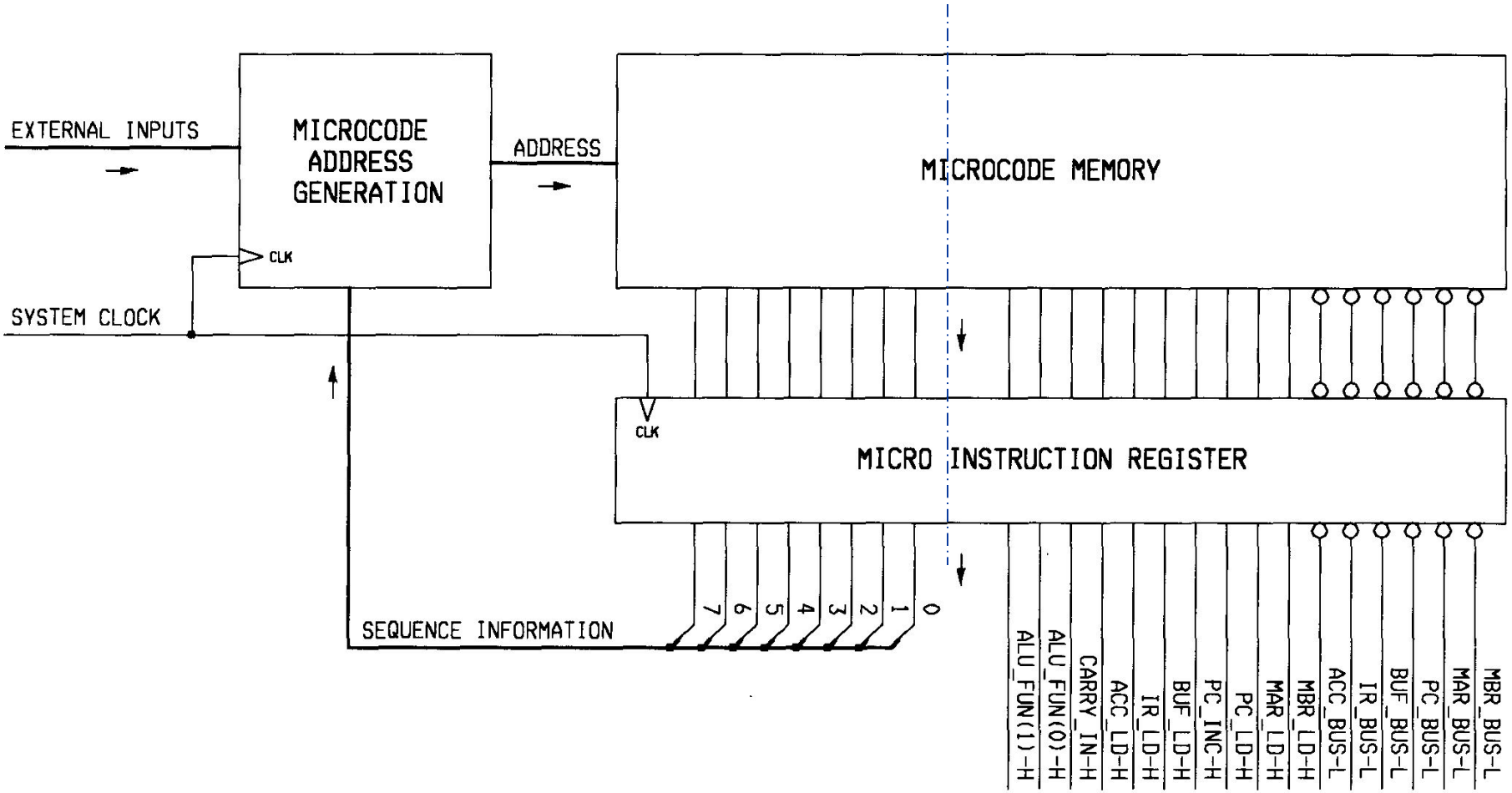


- 1.rész: szabályozza az eszköz működését a megfelelő állapotok *sorrendjében*
- 2.rész: szabályozza az adatfolyamot a megfelelő *vezérlőjelek beállításával* (assertion) az adatúton (vezérlési pontokon)

FSM megvalósítása Memóriával (folyt)

- ***Address Selection***: (mint új elem) a következő utasítás (*Next State*), és beállítani kívánt vezérlőjel (*control signal assertion*) címére mutat a memóriában.
- A memória címet (***memory address***-t) külső bemenő jelek és a present state határozzák meg együttesen. E cím segítségével megkapjuk az adott vezérlő információ pontos helyét a memóriában, ill. ez az információ, mint új állapot betöltődik a vezérlőjel regiszterbe (*Control Signal Register*).
- ***Next-State*** kiválasztásához szükséges logikai memória méretét az aktuális állapotok száma, az állapotdiagram komplexitása, és a bemenetek száma határozza meg.
- ***Control Signal*** generálásához szükséges logikai memória méretét a bemenetek száma, a függvény (vezérlő jel) komplexitása, és a vezérlőjelek száma határozza meg.

Általános Mikrokódos vezérlő



Általános Mikrokódos vezérlő felépítése

- ***Micro Instruction Register**:* a „Present State” (aktuális állapot) regisztert + a Control Signal regisztert egybeolvasztja (az adatút vezérlővonalainak beállítása / kiválasztása). Mikroutasítások sorrendjében generálódik a vezérlőjel!
- ***Microcode Memory**:* a Control Signal Assertion Logic vezérlőjel generálás/beállítás + „Next-State” kiválasztása (mikroprogram eltárolása) összevonása
- ***Microcode Address Generator**:* a vezérlő jelet az aktuális mikroutasítások lépéseiként sorban generálja, de címkiválasztási folyamat komplex. **Sebesség a komplexitás rovására változhat!** (komplexebb vezérlési funkciót alacsonyabb sebességgel képes csak generálni). A következő cím kiválasztása még az aktuálisan futó mikroutasítás végrehajtása alatt végbemegy! Számlálóként működik: egyik címről a másik címre inkrementálódik (mivel a mikroutasításokat tekintve szekvenciális rendszerről van szó). Kezdetben resetelni kell.

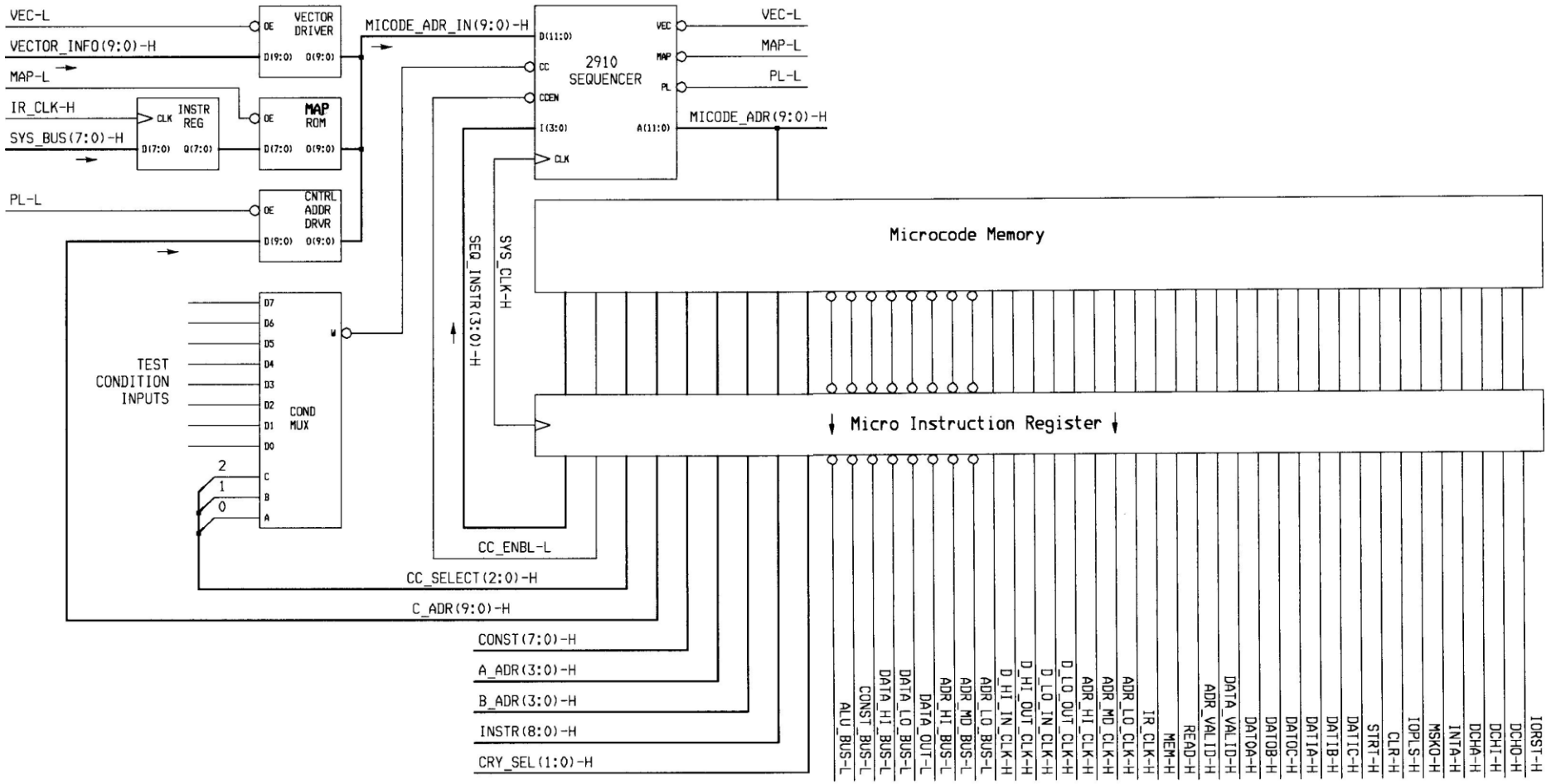
Általános Mikrokódos vezérlők tulajdonságai

- Egy gépi ciklus alatt egy **mikroprogram** fut le (amely mikroutasítások sorozatából áll). A műveleti kód a végrehajtandó mikroprogramot jelöli ki. A mikrokódú memória általában csak olvasható ROM, ha írható is, akkor dinamikus mikroprogramozásról beszélünk.
- Ha a mikroprogram utasításai szigorúan *szekvenciálisan* futnak le, akkor a címüket egy egyszerű számláló inkrementálásával megkaphatjuk. Memóriából érkező bitek egyik része a következő *cím kiválasztását* (Sequence Information), míg a fennmaradó bitek az *adatáramlást* biztosítják.
- Lassabb, mint a huzalozott vezérlő egységek, hiszen itt a memória elérési idejével is számolni kell (nem csak a visszacsatolt aktuális állapot késleltetésével.)

1.) Horizontális mikrokódos vezérlő

- Mindenegyes 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, egyidőben 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, **fogyasztása**.

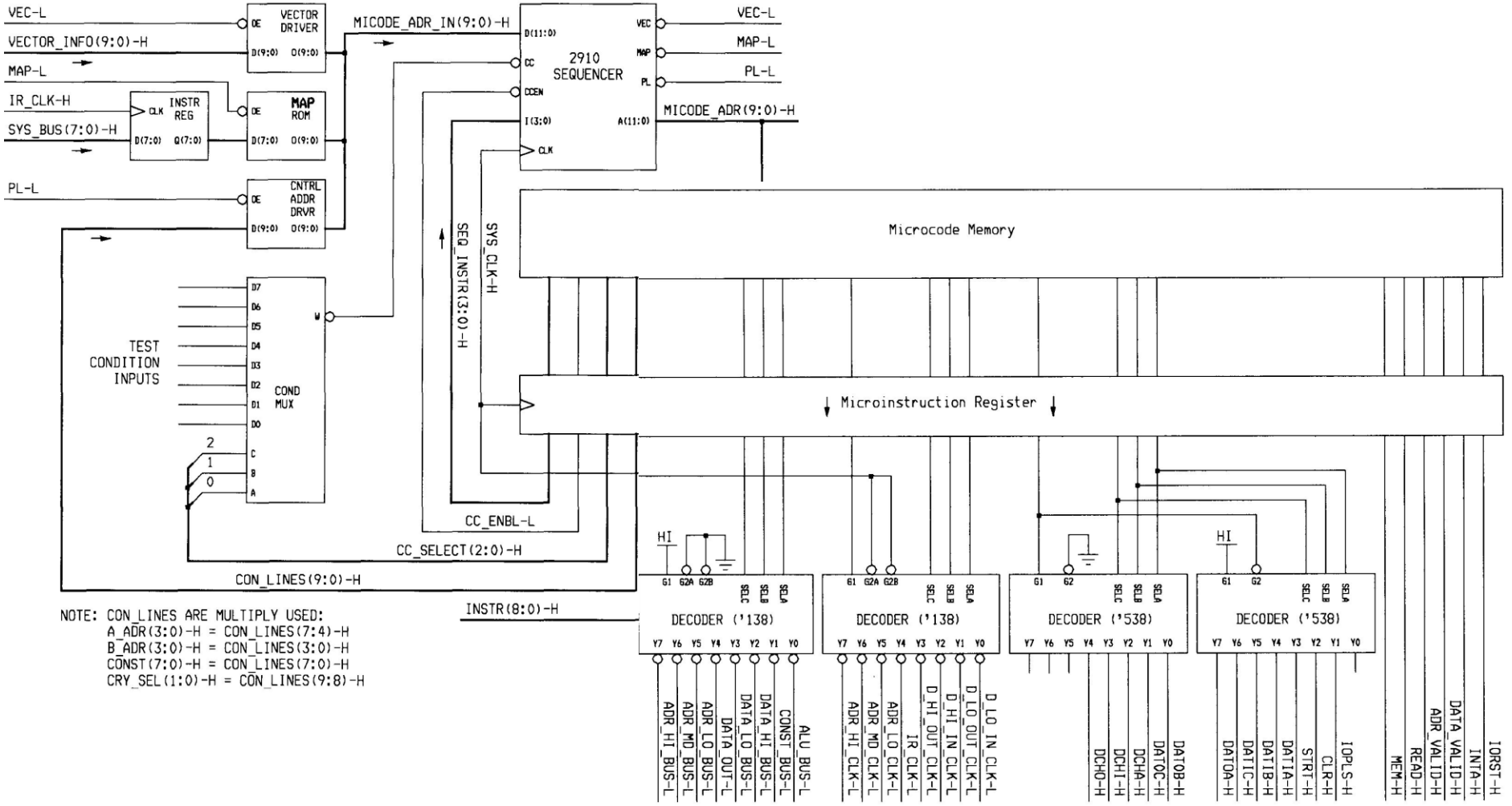
Horizontális mikrokódos vezérlő



2.) Vertikális mikrokódos vezérlő

- Nem a sebességen van a hangsúly, hanem hogy **takarékoskodjon** az erőforrásokkal (fogyasztás, mikrokódban a bitek számával), ezért is **lassabb**.
- Egyszerre csak a szükséges (korlátozott számú) biteket kezeljük, egymástól nem teljesen függetlenül, mivel közülük egyszerre csak az egyiket állítjuk be. A jeleket ezután **dekódolni** kell (a dekódolás több időt vesz igénybe). A kiválasztott biteket megpróbáljuk minimális számú vonalon keresztül továbbítani.
- A műveletek párhuzamos (konkurens) végrehajtása korlátozott. Dekódolás: N bites buszt, $\log_2(N)$ számú bittel próbálunk dekódolni. Több mikroutasítás szükségeltetik $\Rightarrow$ így a mikrokódú memóriát „**vertikálisan**” meg kell növeli.

Vertikális mikrokódos vezérlő

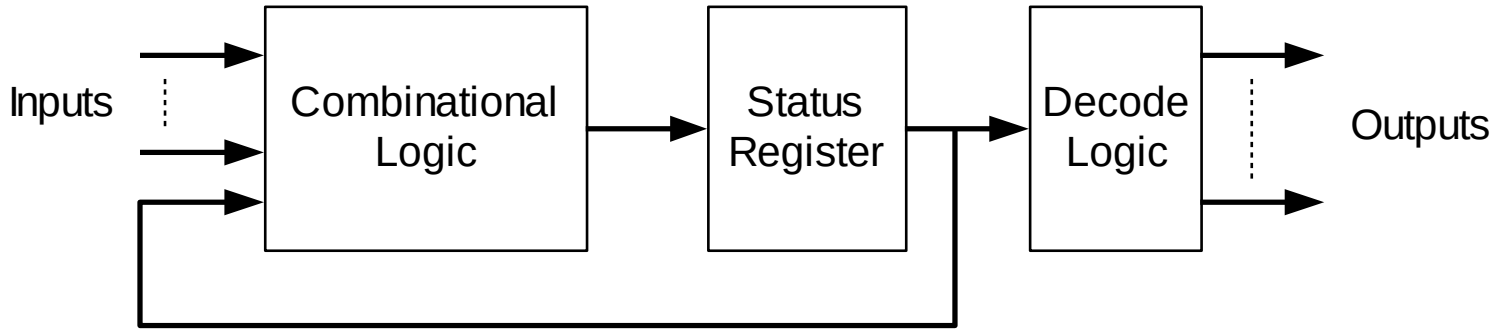


NOTE: CON_LINES ARE MULTIPLY USED:
A_ADR(3:0)-H = CON_LINES(7:4)-H
B_ADR(3:0)-H = CON_LINES(3:0)-H
CONST(7:0)-H = CON_LINES(7:0)-H
CRY_SEL(1:0)-H = CON_LINES(9:8)-H

Programozható logikai eszközök (PLD)

Állapotgép FSM tervezés tulajdonságai

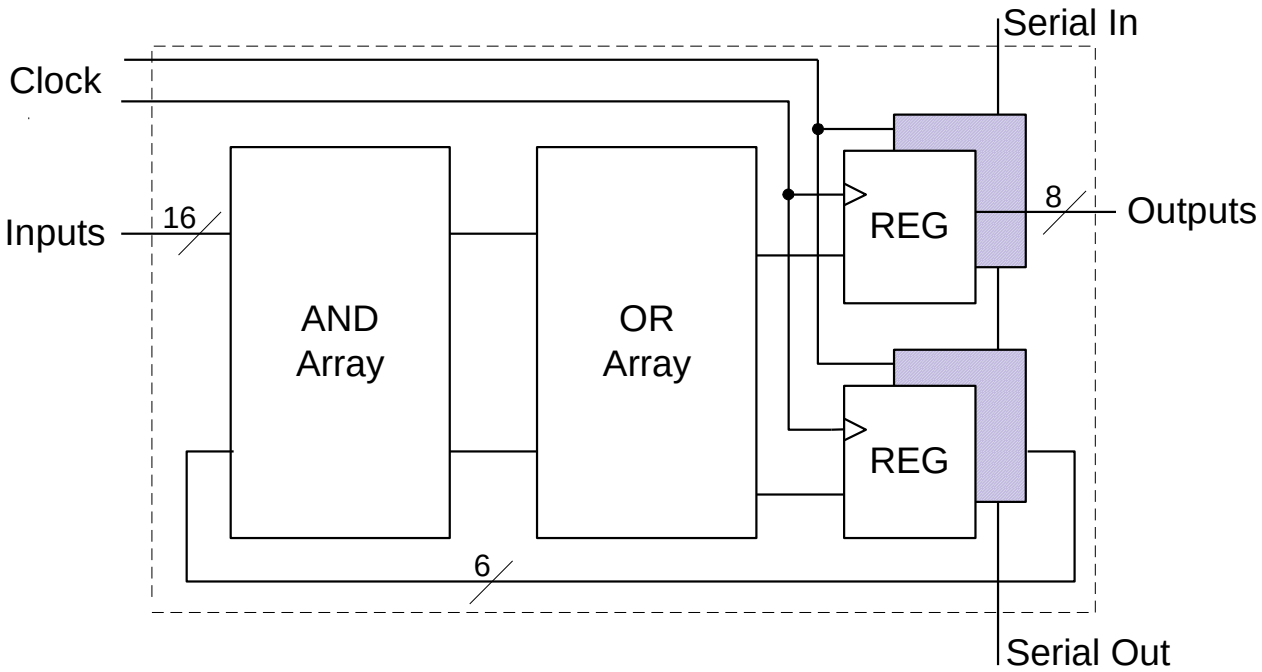
- Két kombinációs logikai hálózatból és egy regiszterből áll
- Tervezés során az állapot-átmeneteket vesszük figyelembe, **DE**
- Hibavalószínűség nagy,
- Szimulációs eszközök (Tools) hiánya,
- Hibák lehetségesek a prototípus fejlesztése során is,
- Könnyen konfigurálható / flexibilis eszközök szükségesek
  → mindezek használunk programozható alkatrészeket



Field Programmable Logic Sequencer (FPLS) – Logikai sorrendvezérlő

- Egy vezérlő egység Next-State logikai blokkjának megvalósítása a tervező, gyártó feladata. Általában változó logikát használnak a függvények megvalósításánál.
- A felhasználó által programozható logikai sorrendvezérlő (**Field Programmable Logic Sequencer**) programozható alkatrészekből építhető fel, amelyek a következőkben részletesen ismertetésre kerülnek.

Field Programmable Logic Sequencer (FPLS) – Logikai sorrendvezérlő

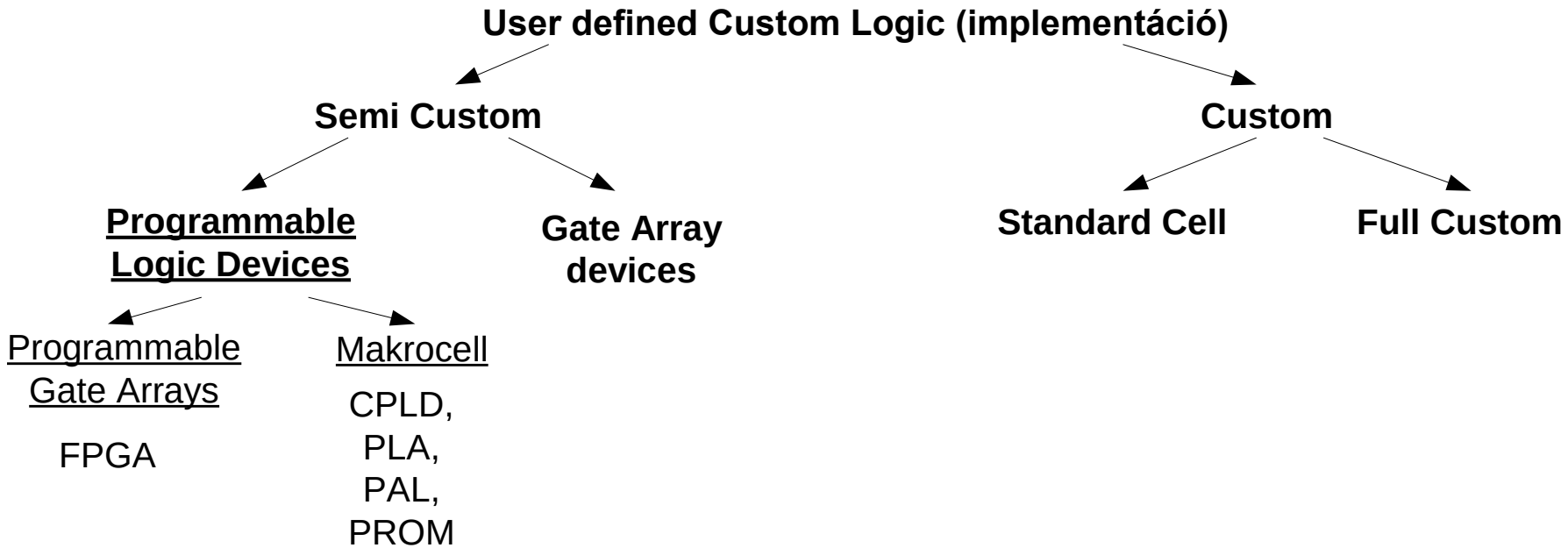


Működése:

- Normál (D-FF),
- Debug (Shift Reg.)

A logikai sorrendvezérlőnek 16 külső bemeneti vonala van, 1 RESET és 1 kimenet-engedélyező vonala, ill. 8 kimeneti vonala. A regiszterek egy-egy állapotot tárolnak, amelyek az órajel hatására a kimenetre íródnak, vagy a 6 belső, *visszacsatoló* vonalon keresztül visszacsatolódnak. A Next-State ill. a kimeneti szintek meghatározásánál programozható AND/OR tömböket használnak.

Felhasználó által definiált logikai implementációk:



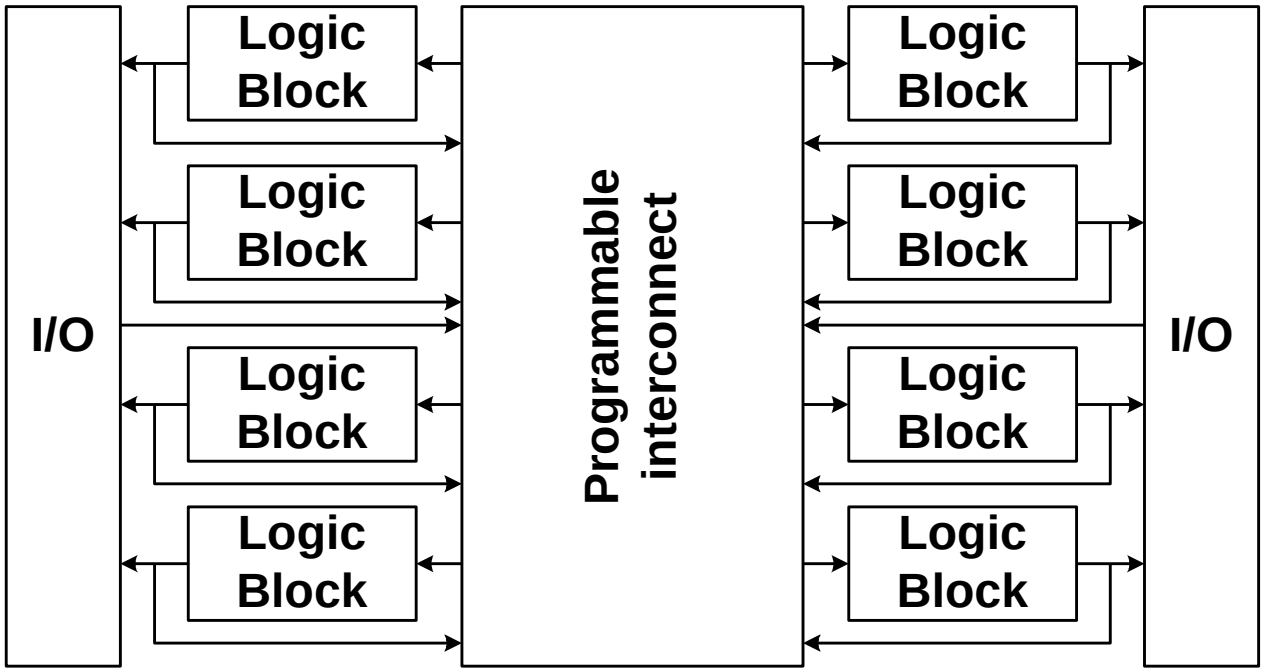
* Fontos: Ezek a részek nem találhatóak meg a könyvben!

Programozható logikai eszközök (PLD-k) két fő típusa:

- 1.) Makrocellás PLD-k: (Programmable Logic Devices):
  - PLA
  - PAL
  - PROM
  - EPLD, CPLD
- 2.) FPGA (Field Programmable Gate Array): Programozható Gate Array áramkörök
  - **XILINX** (Spartan, Virtex)
  - Altera,
  - Actel (főleg űrkutatásban alkalmazott) sorozatok

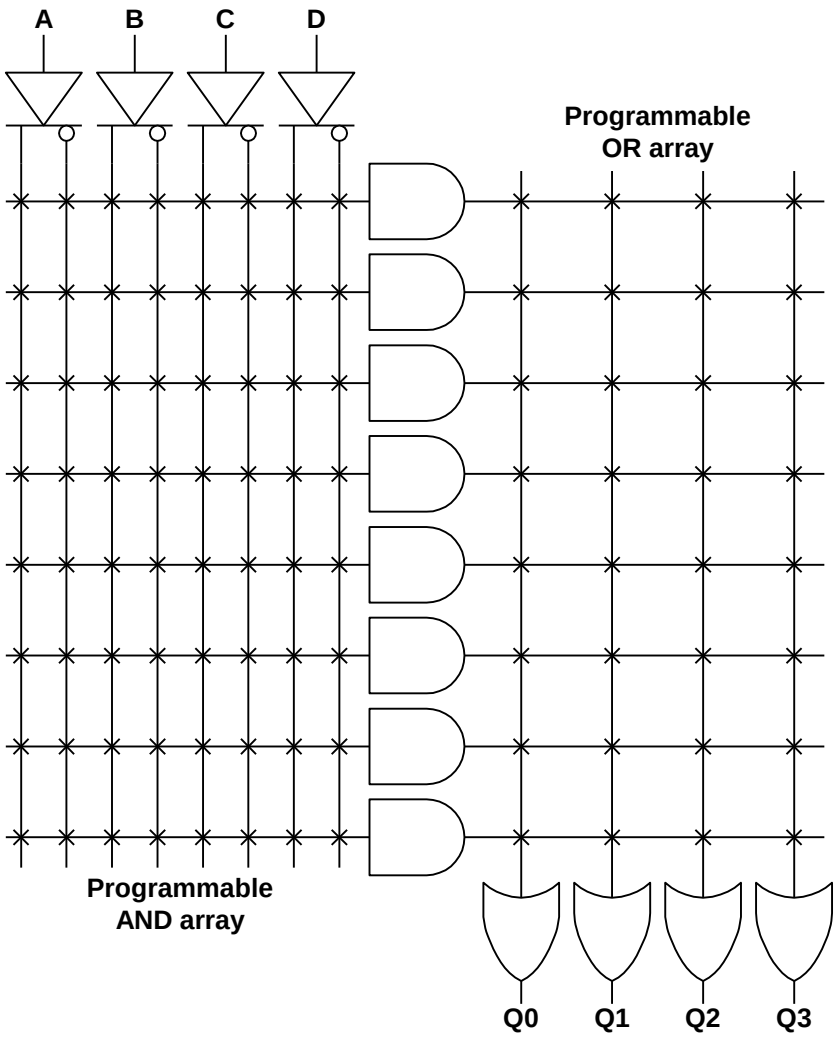
Complex Programmable Logic Device (CPLD)

- Logikai Blokk-ban
  - PAL / PLA
  - Regiszterek
- I/O Blokkok
- Programozható összeköttetések (Interconnect)
  - Teljes (Full-crossbar), vagy
  - Részleges összeköttetés hálózat



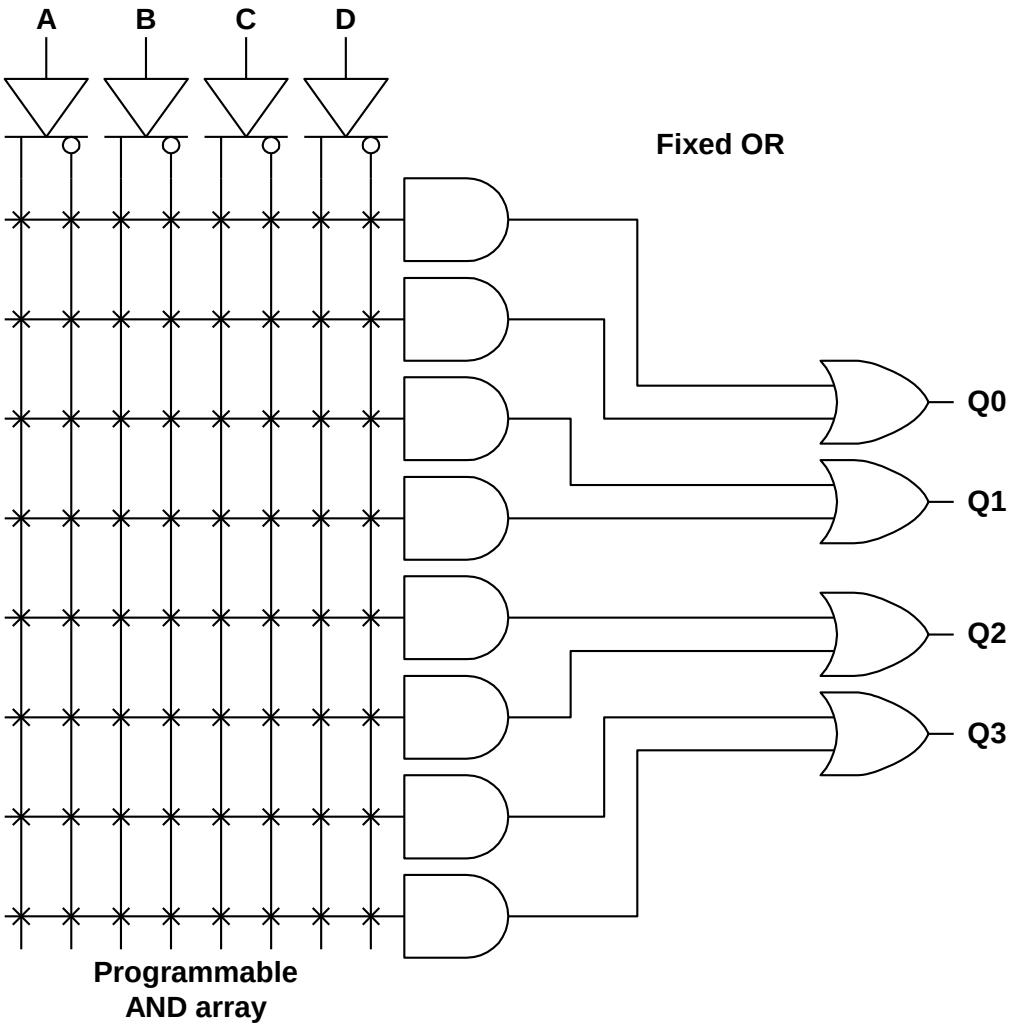
Programmable Logic Array (PLA)

- Mindkét része (AND, OR) programozható
- Bármely kombinációja az AND / OR-nak előállítható
- Mintermek OR kapcsolata (DNF)
- Programozható kapcsolók a horizontális/ vertikális vonalak metszésében
- Q_n Kimeneteken D tárolók! (v.csat. a bemenetekre)



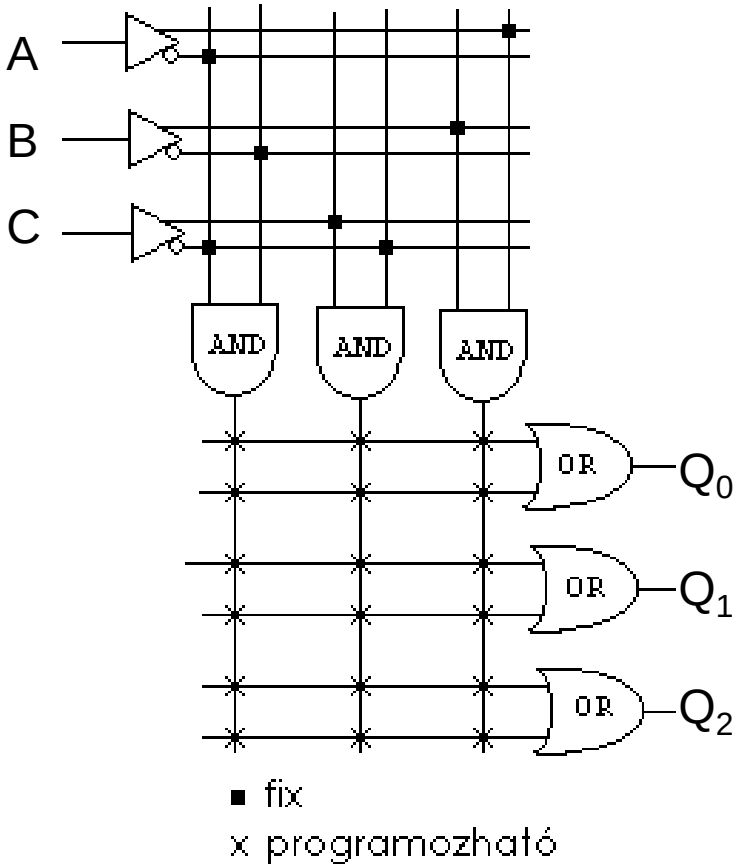
Programmable AND Logic (PAL)

- Egy programozható rész - AND / míg az OR fix
- Véges kombinációja áll elő az AND / OR kapcsolatoknak
- Metszéspontokban kevesebb kapcsoló szükséges
- Gyorsabb, mint a PLA
- Q_n kimeneteken D tárolók (visszacsatolódhatnak a bemenetekre)



Programmable Read Only Memory (PROM)

- Egy programozható rész - OR / míg az AND fix
- Véges kombinációja áll elő az AND / OR kapcsolatoknak
- Metszéspontokban kevesebb kapcsoló szükséges
- Gyorsabb, mint a PLA
- Q_n kimeneteken D tárolók! (visszacsatolódhatnak a bemenetekre)



Hogyan programozhatók a VSLI alkatrészek?

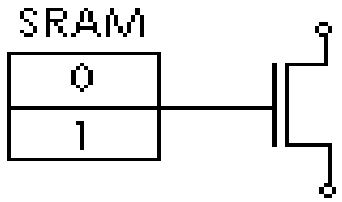
Programozási technikák

Programozási technikák

- a.) SRAM
- b.) MUX
- c.) Antifuse
- d.) Floating Gate
- e.) EPROM/EEPROM/Flash (Lattice)

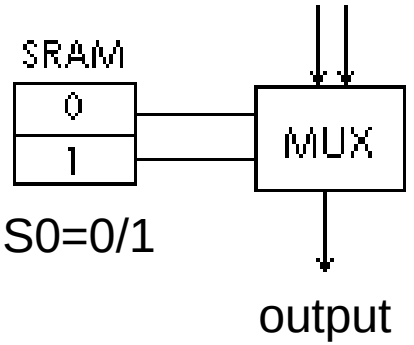
a.) SRAM cellás

- 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.) MUX - multiplexeres

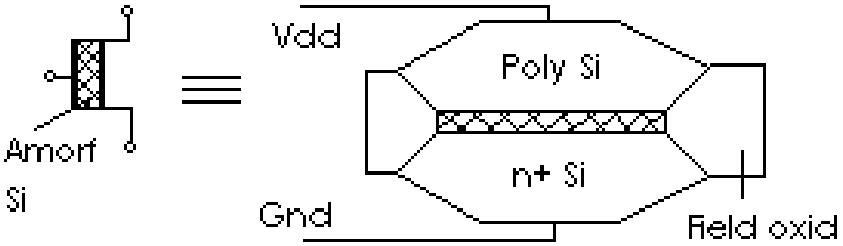
- Tulajdonságai:
  - az SRAM-ban tárolt ’0’ vagy ’1’ értéket használunk a Multiplexer bemeneti vonalának kiválasztásához. (Működése hasonló az SRAM celláéhoz.) /Bemenetek közül választ a selektáló SRAM-beli érték segítségével és a kimenettel köti össze./

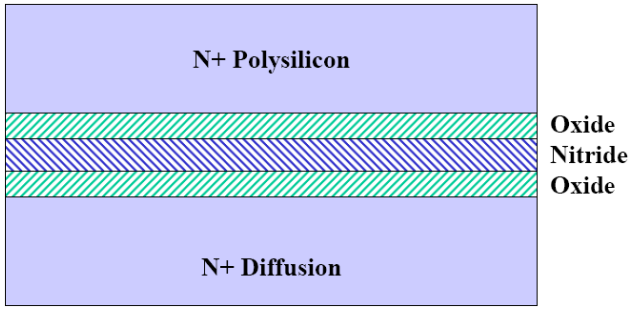


c.) Antifuse

A tranzisztor Gate-jét amorf kristályos Si alkotja, amelyet relatíve nagy feszültség (kb 20-30V) hatására átkristályosítunk, így vezetővé válik véglegesen. Pl. Texas Instruments alkalmazza ezt a technológiát. Tulajdonságai:

- 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
  - ONO (Oxid-Nitrid-Oxid)
  - Amorf Si

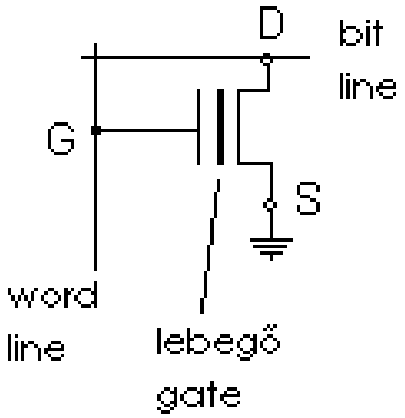


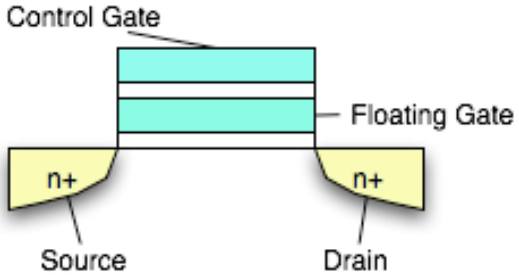


d.) Floating gate

Két-gates tranzisztort használunk, amelynek középső gate-je a lebegő gate. A másik gate fix. INTEL alkalmazza. Programozható összeköttetéseknél, csomópontokban használatos.

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





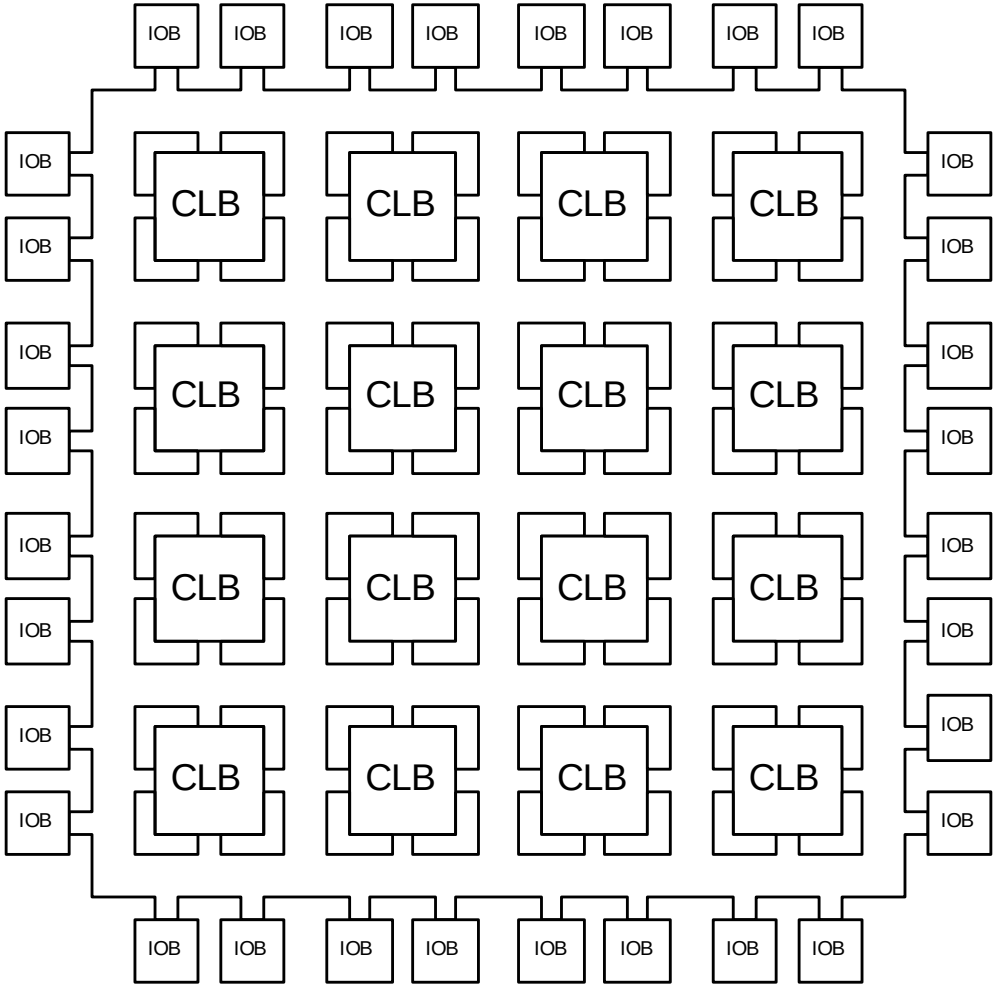
e.) EPROM / EPROM / Flash

- Tulajdonságai
  - **„Floating-gate” technológiát alkalmazza**!
  - 10000x szer programozható
  - Megőrzi tartalmát (Non-volatile)
  - UV-fénnyel törölhető (EPROM)
  - Elektromosan törölhető (EEPROM / Flash)
  - Nagy felület
  - Nagy parazita ellenállás
  - Nagy parazita kapacitás
  - További CMOS gyártási lépések (sok maszk réteg) szükségesek - drága

FPGA (Field Programmable Gate Array) architektúrák

Field Programmable Gate Array (FPGA)

- Konfigurálható Logikai Blokk (CLB)
  - Look-up table (LUT)
  - Regiszter
  - Logikai áramkörök
    - Összeadók (Adder)
    - Szorzók (Multiplier)
    - Memóriák (Memory)
    - Microprocessor
- Input/Output Block (IOB)
- Programozható összeköttetés hálózat

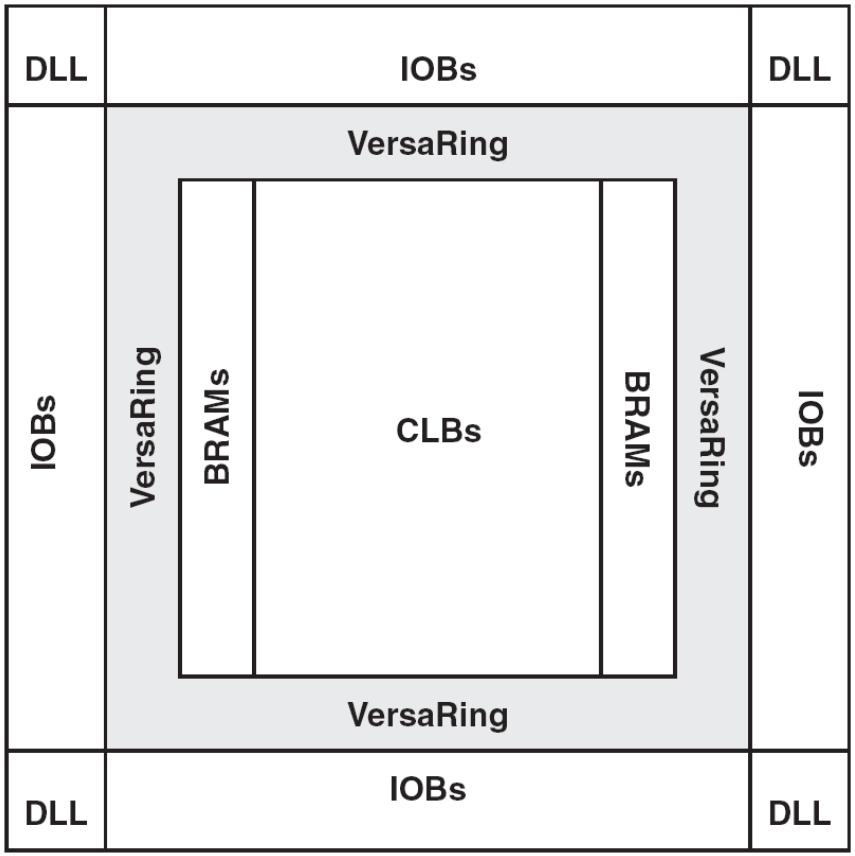


Xilinx FPGA családok

- Nagy teljesítmény
  - Virtex (1998)
    - 50K-1M gates, 0.22µm
  - Virtex-E/EM (1999)
    - 50K-4M gates, 0.18µm
  - Virtex-II (1999)
    - 40K-8M gates, 0.15µm
  - Virtex-II Pro/X (2002)
    - 50K-10M gates, 0.13µm
  - Virtex-4 (2004)
    - 50K-10M gates, 90nm
  - Virtex-5 (2006)
    - 65nm
- Alacsony költség
  - Spartan-II (2000)
    - 15K-200K gates, 0.22µm
  - Spartan-IIE (2001)
    - 50K-600K gates, 0.18µm
  - Spartan-3 (2003)
    - 50K-5M gates, 90nm

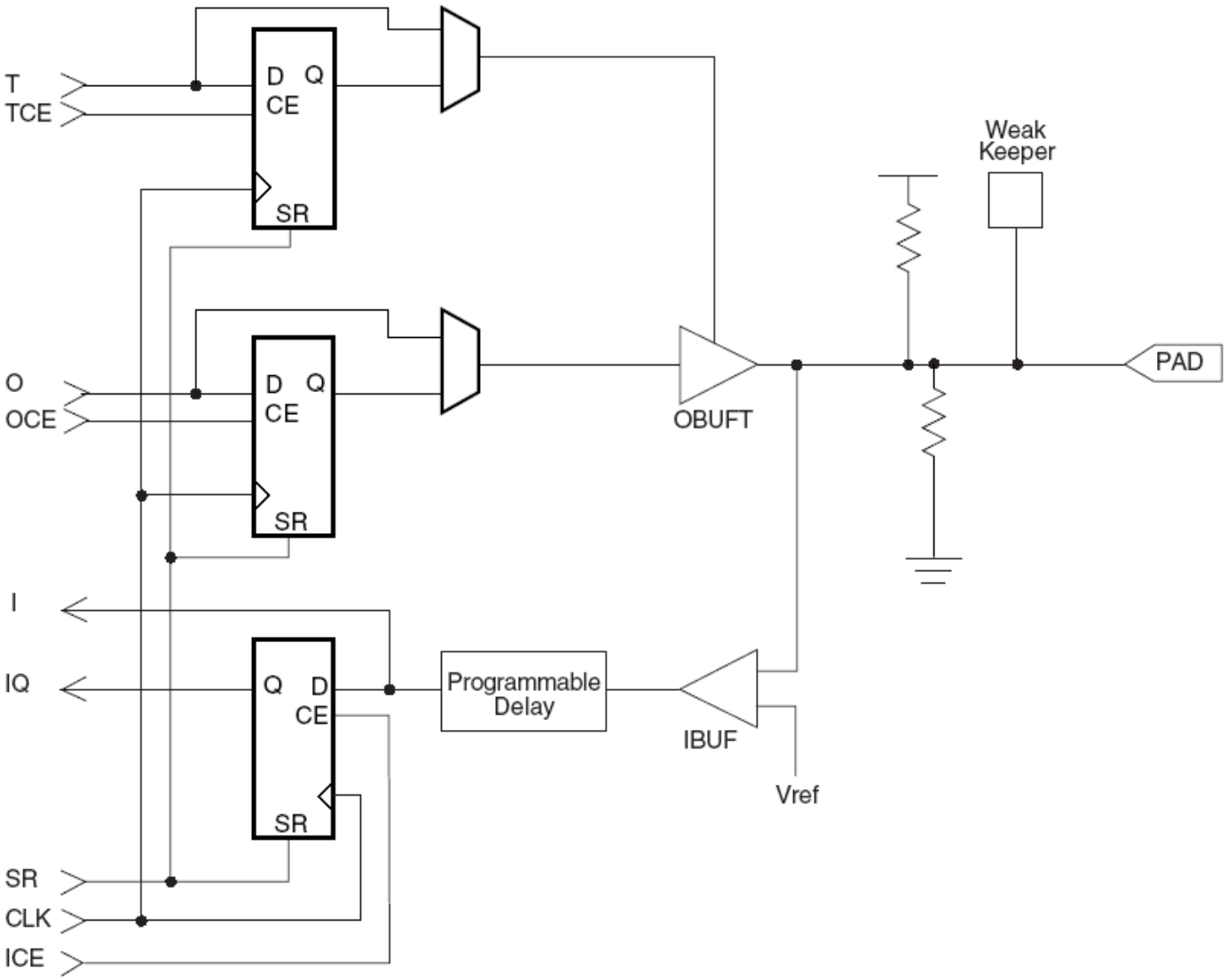
http://www.xilinx.com

Virtex architektúra

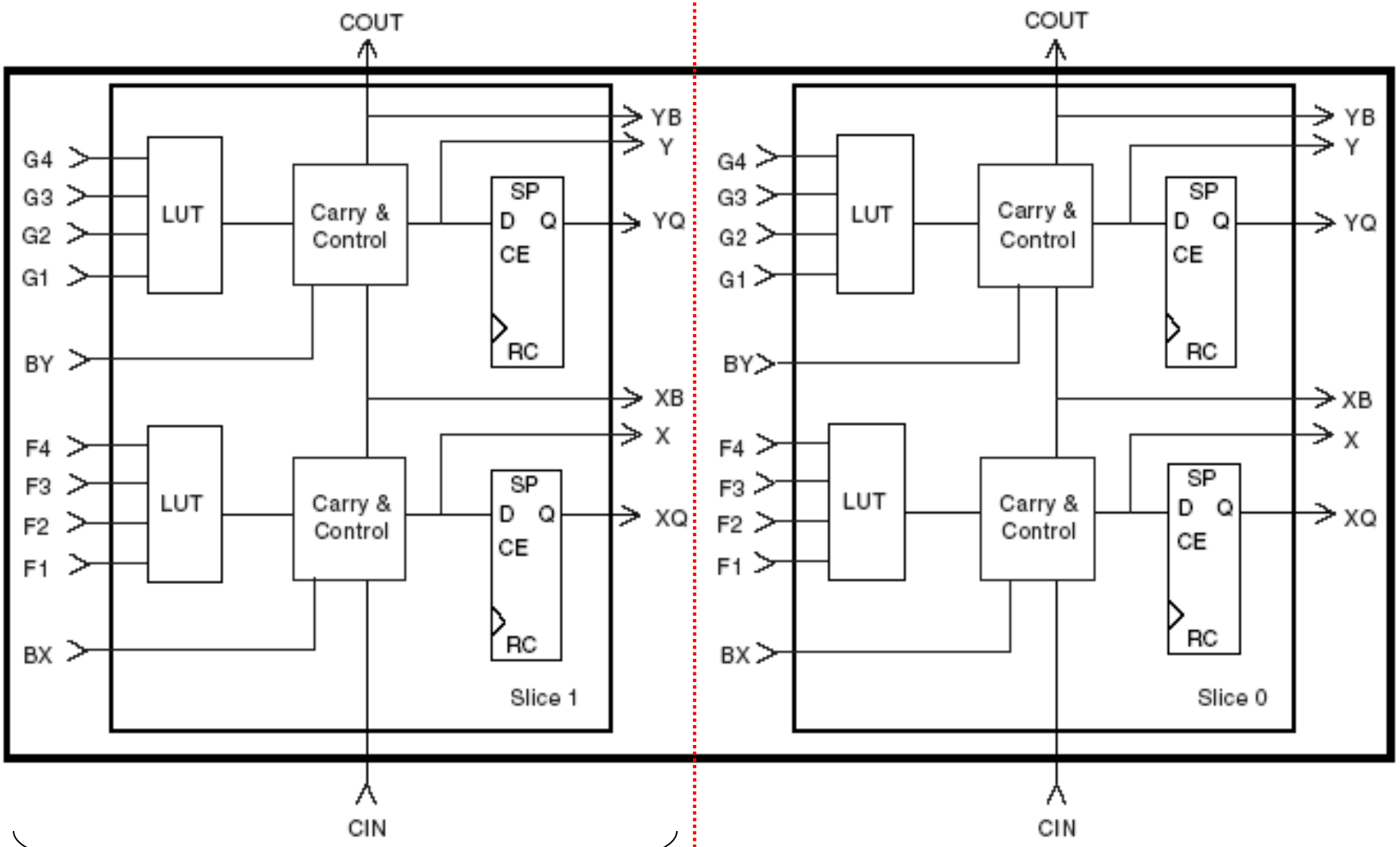


- CLB: Konfigurálható Logikai Blokk
- BRAM: Blokk SelectRAM
  - szinkron dual-portos 4096-bit RAM
- DLL: Delay-Locked Loop
  - Órajel menedzselési funkciók
  - Órajel osztás/ szorzás 1.5, 2, 2.5, 3, 4, 5, 8, vagy 16
  - Fázis tolás: 0º, 90º, 180º, 270º
- IOB: Input/Output Blokk
  - 13 különböző I/O szabványt támogat (pl: LVDS)

Virtex - (IOB) Input/Output Blokk



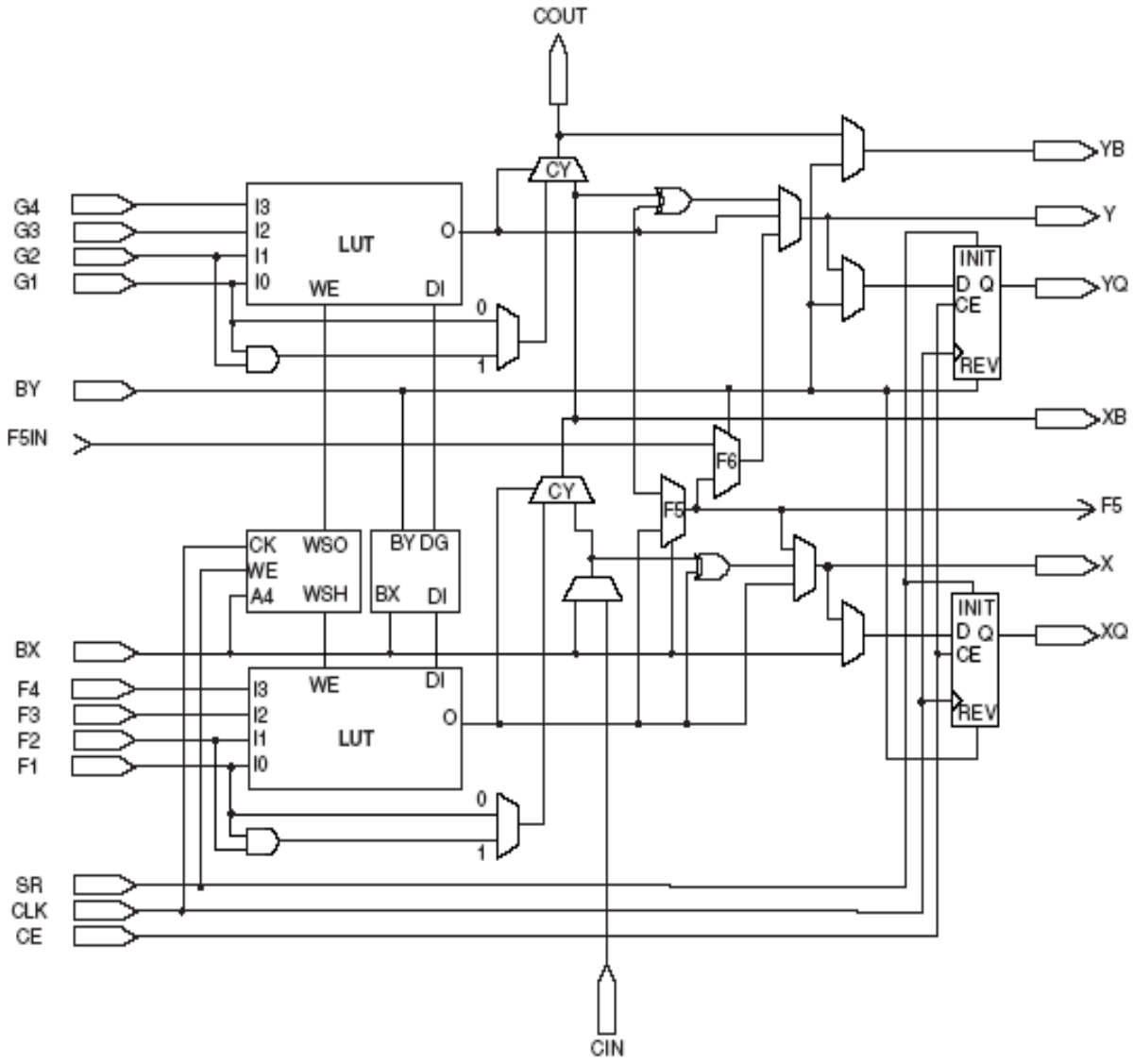
Virtex (CLB) Konfigurálható Logikai Blokk



„slice” - szelet

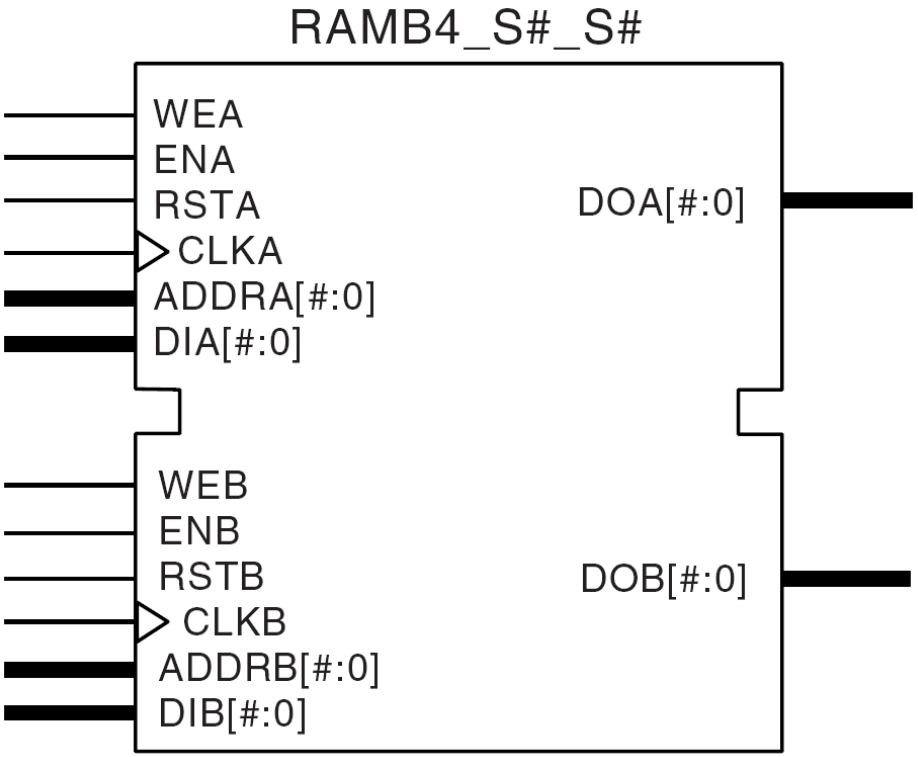
Virtex: Slice - szelet

FPGA-k esetében egy alapvető mérőszám a „slice” – szelet.



- Look-Up Table
  - 4-bemenetű LUT
  - 16 x 1-bit szinkron RAM
  - 16-bit shift regiszter
- Tároló elemek
  - ÉI-vezérelt D-típusú flip-flopok
- További Logikai áramkörök
  - F5 multiplexer
    - bármely 5-bemenetű függvény
    - 4:1 multiplexer
  - F6 multiplexer
    - bármely 6-bemenetű függvény
    - 8:1 multiplexer
- Aritmetikai Logikai Egység
  - Dedikált carry logika
  - Dedikált AND kapuk

Virtex Duál-portos Blokk SelectRAM

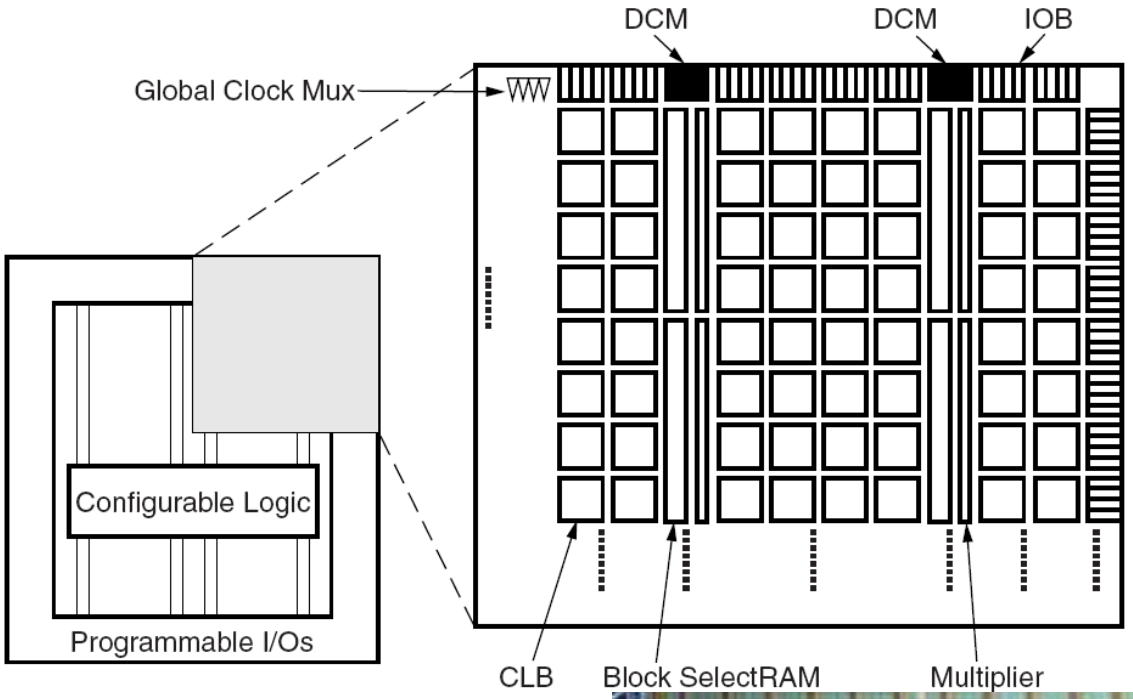


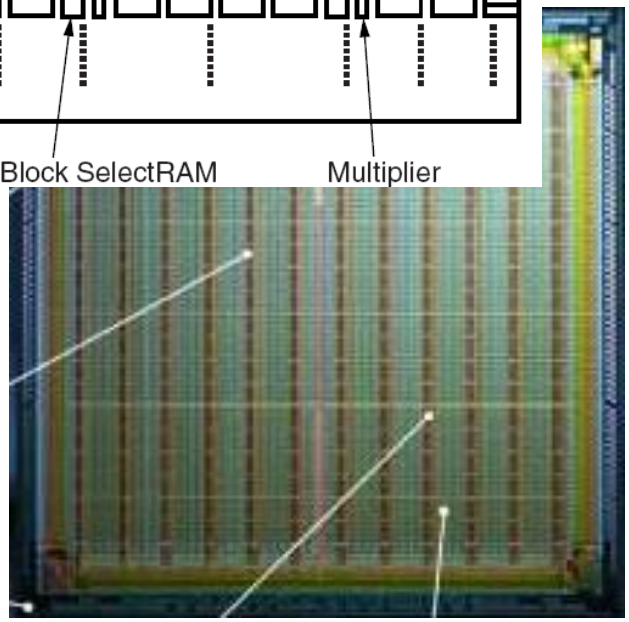
- Minkét portját (A, B) külön címezhetjük (RW)
- Független adatbusz szélesség definiálható:
  - Bus-szélesség konverzió

Table 4: **Block SelectRAM Port Aspect Ratios**

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

Virtex-II architektúra





Dedikált blokkok:

- Blokk SelectRAM
  - 18Kb dual-port RAM
- Multiplier (szorzó)
  - 18x18-bit szorzó
- DCM (Digital Clock Manager)
  - Órajel ütemezés
  - Frekvencia szintézis
  - Fázis tolás stb.