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

Digitális technika
VIMIAA02
6. EA

Fehér Béla, Benesóczky Zoltán, Raikovich Tamás
BME MIT

COVID-19 előadások

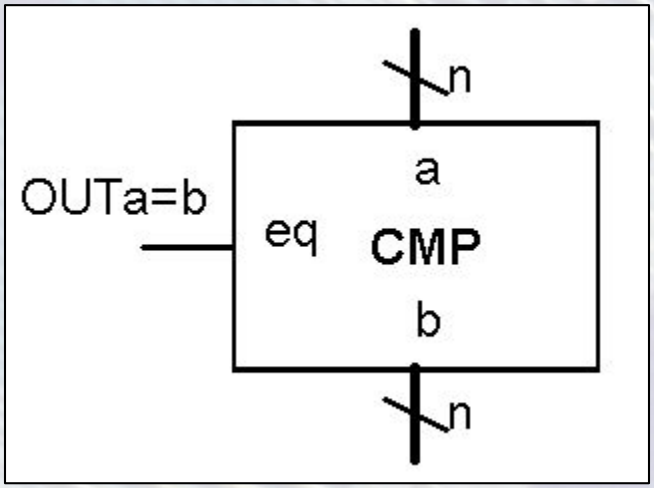
- **Ebben az évben az előadások további két teremben is, kivetítőről követhetők, a kapcsolat így nem elég közvetlen**
- **Ennek javítására beiktatunk visszajelzési lehetőséget**
- **Bármelyik teremből kérdések, megjegyzések emailben küldhetők a benes@mit.bme.hu címre, de csak ... @edu.bme.hu címről, tárgy: DIGIT Résztémák végén, ha marad idő, a kérdéseket megnyitjuk és próbálunk válaszolni**

Funkcionális egységek

- **Kiegészítés az eddigi előadások anyagához**
- **Kombinációs funkcionális egységek eddig:**
  - Dekóder (DEK), multiplexer (MUX), összeadó (ADD)
- **További adatfeldolgozási funkciók, melyeket a számlálóknál alkalmaztunk:**
  - Értékfelismerés, adat összehasonlítás (CMP)
  - Összeadó/kivonó (ADD/SUB)
  - Inkrementáló/dekrementáló (INC/DEC)
- **Speciális tároló funkcionális egységek**
  - Regisztertömbök
  - Memóriák (RAM, ROM), STACK, FIFO

Funkcionális egységek komparátor

- **Komparátor (CMP)**
  - Feladata értékek, adatok összehasonlítása
  - Két azonos bitszámú számot hasonlít össze
- **Egyenlőség komparátor**
  - Akkor ad 1-et a kimenete, ha a két szám egyenlő, vagyis a két szám azonos sorszámú bitjei azonosak



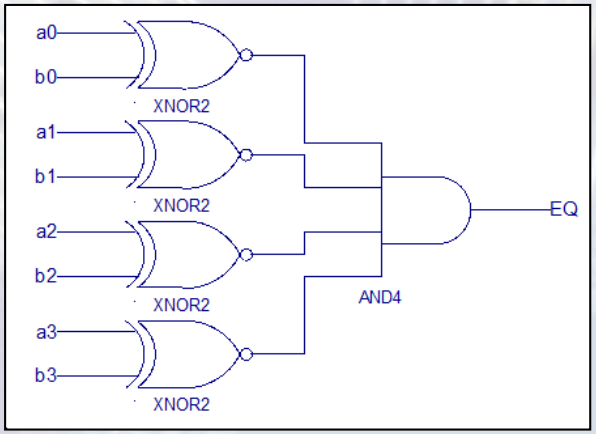
Funkcionális egységek
komparátor

- Egybites számok esetén az XNOR (ekvivalencia kapu) művelet azonos értékű bitek esetén jelez
  f = ab + /a/b = a $\odot$ b

a	b	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

- n bites számok esetén akkor kell jelezni, ha az egyenlőség minden bitre teljesül:
  equ = (a0 $\odot$ b0)(a1 $\odot$ b1)...(a_{n-1} $\odot$ b_{n-1})
- Verilog leírása 4 bitre

```
module CMPeq(in [3:0] a,b,
                      out equ)
assign equ = (a == b);
endmodule
```

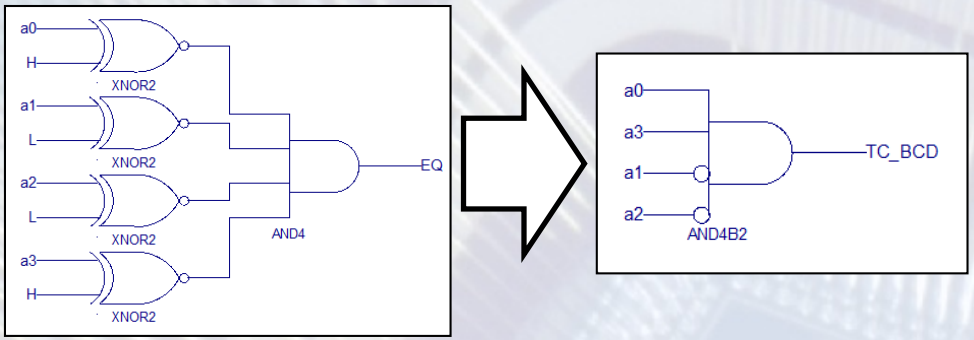


- Tetszőleges kódolásra működik

Funkcionális egységek
komparátor

Egyenlőség komparátor *konstanssal* hasonlításra

- Fix érték vizsgálatára XNOR kapuk egyik bemenete konstans 0 vagy 1
- Az f = ab+/a/b függvény egyik változója helyére a konstansokat helyettesítve belátható, hogy ***az XNOR kapu az egyik benentén levő konstans 0 értéke esetén meginvertálja a másik bemenetén levő változót, a konstans 1 értéke esetén pedig nem***:
  Ha b=0, f = a*0 + /a*/0 =/a
  Ha b=1, f = a*1 + /a*/1 = a



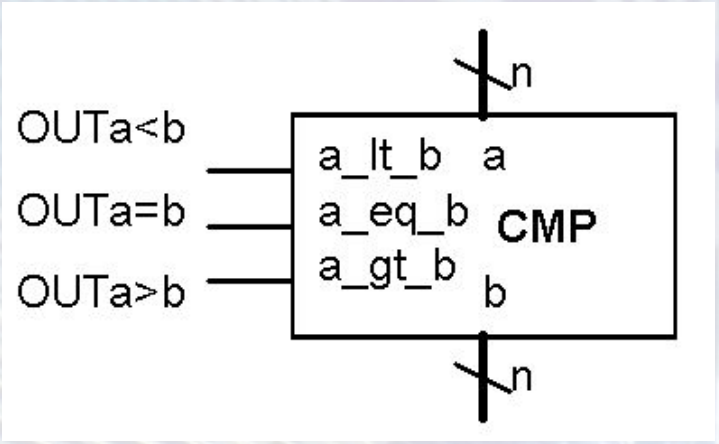
- Így ***az egyenlőség komparátor konstans összehasonlítás esetén*** egy olyan ***AND kapuvá alakul***, melynek bemenetire a változó 0-val hasonlított bitjei invertálva jutnak rá, a többi pedig ponáltan.

Funkcionális egységek
komparátor

Teljes funkciójú (magnitudo vagy nagyság) komparátor

- A teljes funkciójú komparátor a két adat ***egyenlősége*** (***a_eq_b***) mellett egy-egy kimenetén azt is jelzi, hogy melyik a ***nagyobb (a_gt_b)*** ill. ***kisebb (a_lt_b).***
- ***Verilog leírás, 4 bites példa:***

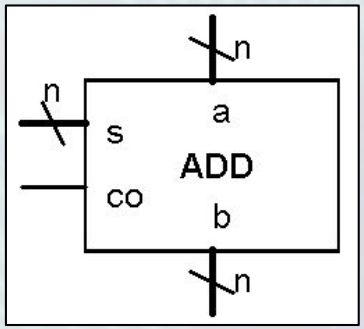
```
wire [3:0] a, b;
wire a_lt_b, a_equ_b, a_gt_b
        assign a_lt_b = (a < b);
        assign a_eq_b = (a ==b);
        assign a_gt_b = (a >b);
```



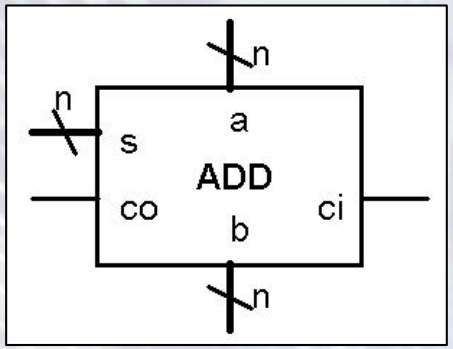
- Egy sorban is leírható:
  assign {a_lt_b, a_equ_b, a_gt_b} = {(a < b), (a ==b), (a > b)};
- Kivonó is használható a nagysági viszony eldöntésére (lásd később)

Funkcionális egységek
összeadó

- Szerepelt korábban az 1 bites teljes **összeadó**
- Ebből kaszkádosítással készíthetünk több biteset, de egyéb megvalósítások is léteznek
- Input: összeadandók (a,b), áthozat (ci) Output: összeg (s), átvitel (co)
- n-bites összeadó:



n bites összeadó kaszkádosító bemenettel:



Verilog megvalósítások:

```
wire [3:0] a, b, s
wire co
assign {co, s} = a + b;
```

```
wire [3:0] a,b,s
wire ci, co;
assign {co, s} = a + b + ci;
```

Funkcionális egységek
összeadó

- Kettes komplemens kódú számok összeadására is alkalmas az összeadó.
- Itt is figyelni kell a ***számtartomány***ra! 2-es komplemens ***túlcsordulás*** lehet! Ha az összeadandó számok előjele különbözik, akkor biztosan nem lesz túlcsordulás, viszont azonos előjelűek esetén előfordul.

4 bites 2-es komplemens számok tartománya: +7...-8		
(+7) + (-1) = +6	0111 + 1111 = 0110	
	(-7) + (-2) = -9	1001 + 1110 = 0111 (+7 hibás!)
		(+2) + (+7) = +9	0010 + 0111 = 1001 (-7 hibás!)

- Akkor keletkezett túlcsordulás, ha az összeadandók előjele azonos, de az eredmény előjele ettől eltérő. 4 bites számok esetén:

  OVF = a[3]*b[3]*/s[3] + /a[3]*/b[3] *s[3]

- ***Probléma elkerülése***: használjunk annyi bites számábrázolást, amiben az eredmények elférnek.

Funkcionális egységek
kivonó

Kivonás (előjel nélküli számok esetén)

- **Szabályok:** 0-0=0, 1-1=0, 1-0=1, 0-1=1 átvitel 1
- **Magyarázat az átvitelhez**:(0-hoz hogy 1 legyen kell 1, átvitel 1)
- Az eredmény ***csak akkor helyes***, ha a kisebbítendő nagyobb vagy egyenlő mint a kivonandó.

Példa:

	11
14	1110
- 11	- 1011
3	0011

Magyarázat:
1-hez, hogy 0-legyen kell 1, marad 1
1+1=0, marad 1, 0-hoz, hogy 1 legyen, kell 1
1+0=1, 1-hez, hogy 1 legyen kell 0, maradék 0
1-hez, hogy 1 legyen kell 0, maradék 0

Funkcionális egységek
kivonó

- **Kivonó – írásbeli kivonásra alapozva**
  - Az 1 bites teljes összeadóhoz hasonlóan elkészíthető az ***1 bites teljes kivonó***, amely elvégzi a kivonást egy helyiérték esetén → kaszkádosítással: N bites kivonó
  - Bemenetek: ***a*** (kisebbítendő), ***b*** (kivonandó), ***cin*** (bejövő átv.)
  - ***Igazság-tábla:***

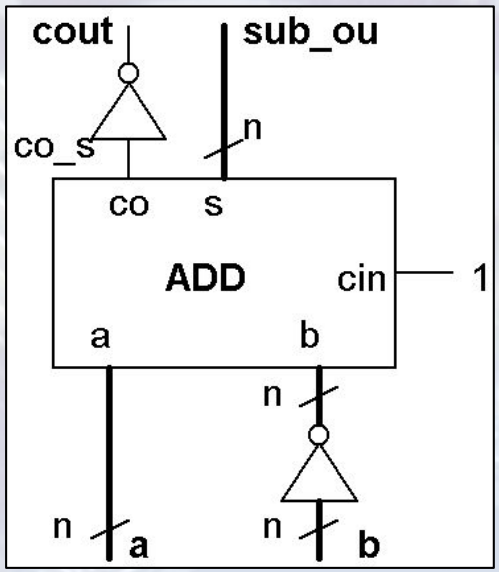
a	b	cin	cout	s
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0

a	b	cin	cout	s
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

  - A kimenetek legegyszerűbb alakban:
    - Különbség: $s = a \text{ xor } b \text{ xor } cin$
    - Kimenő átvitel: $cout = /a \cdot b + /a \cdot cin + b \cdot cin$

Funkcionális egységek
kivonó

- **Kivonó – kettes komplemens hozzáadásával**
  - ***Az összeadó jól működik pozitív és kettes komplemens negatív számokra is!*** (Ezért terjedt el a 2-es komplemens számábrázolás.)
  - Az **a-b** művelethez képezzük **-b**-t a 2-es komplemens képzéssel
  - **a-b** = a+(-b) = **a + /b + 1**
    A (-B) előállításához 2-es komplemenst képzünk (B minden bitjét invertáljuk és hozzáadunk 1-et)
  - Az 1- hozzáadásához az összeadó cin bemenetére 1-et kapcsolunk (S = a+b+cin)
  - ***Az összeadó co kimenete itt fordítva működik***, mint a hagyományos kivonónál, ha zavaró, ***meginvertáljuk***.



Funkcionális egységek komparátor kivonóval

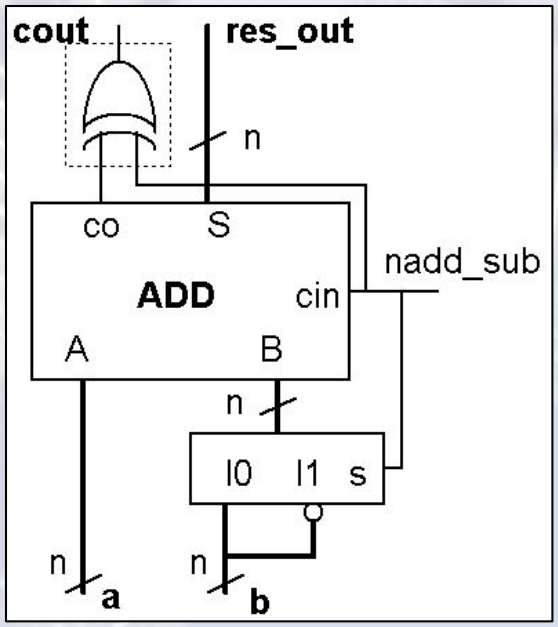
Nagyság komparátor megvalósítása kivonóval

- **Előjel nélküli számábrázolás esetén: átvitel (Co) keletkezik, ha a kisebbítendő (a) kisebb, mint a kivonandó (b)**
  - **Kivonó – írásbeli kivonásra alapozva:**
    - **cout = 0: a ≥ b** (azaz a – b ≥ 0) vagyis, ha a különbség pozitív
    - **cout = 1: a < b** (azaz a – b < 0) vagyis, ha a különbség negatív volna azonban ilyenkor az eredmény hibás!
  - **Kivonó – kettes komplemens hozzáadásával:**
    - b kettes komplemense: $2^N - b = /b + 1$ (N: bitszám)
    - $a - b = a + (-b) = a + (2^N - b) = \mathbf{2^N} + (a - b)$
    - **A 2^N hozzáadása invertálja a kimenő átvitel bitet!**
    - Tehát: az összeadó átvitel kimenete: **co_s = 1, ha a ≥ b** és **co_s = 0, ha a < b** egy **invertálással megkaphatjuk a megszokottat**: **cout = /co_s**
- **(Kettes komplemens számábrázolás: kivonásnál is 2-es komplemens túlcsordulás lehet, mint az összeadónál! Nem részletezzük.)**
- **Szoftverben kivonásra képződik le a komparálás!**
- **Státusz bitek jelzik az eredménnyel kapcsolatos információkat:**
  Z (zero, eredmény==0) NOR kapu az eredmény bitekre
  C (carry, átvitel),
  N (negative, előjel bit 1), Az eredmény legnagyobb bitje, csak 2-es komplemens számok esetén használható
  V (overflow, 2-es komplemens túlcsordulás, az eredmény nem fér bele a számtartományba)

Funkcionális egységek összeadó/kivonó

Összeadó/kivonó egyetlen egységben

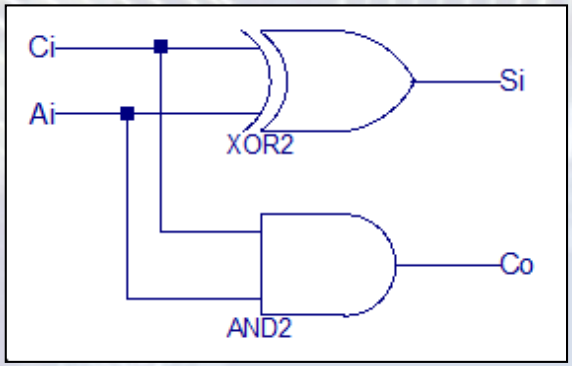
- ***Egy vezérlő bittől függően összead (nadd_sub=0) vagy kivon (nadd_sub = 1)***
- ***Összeadás*** esetén **b *normál értéke***, **kivonás** esetén **invertált értéke** *jut* az ADDER *b* bemenetére**. *Összeadásnál cin = 0, kivonásnál cin = 1.***
- Az összeadó B bemenetére a 2/1-es MUX ***b-t*** vagy ***/b-t*** választja ki ***nadd_sub*** értékétől függően.
  Ugyanez vezérelhető INVERTEReként használt EXOR kapuval is megoldható.
- A Cin-re közvetlenül az ***nadd_sub*** vezérlőjelet kapcsoljuk.
- A ***cout kimenet*** csak akkor kell, ha ***tartomány túllépés***t vagy ***nagyság hasonlítás***t is szeretnénk. Kivonásnál ekkor a co-t meginvertálhatjuk.



Funkcionális egységek

Inkrementer/dekrementer

- Az összeadó és kivonó alapján, csökkentett funkció
- Ha B = 0 és Cin = 1, akkor S = A + 1
- A bitenkénti ***FADD teljes összeadóból egyszerűsíthető***
  → **Half ADDER** (fél összeadó, csak 2 bemeneti bitje van: 1 operandus, 1 átvitel), ez másképp az ***inkrementer***
  S = A XOR **B** XOR C Co = A**B** + ACi +**B**Ci, **B = 0**
  S = A XOR **0** XOR C Co = A***0** + ACi + **0***Ci
  S = A XOR C Co = ACi
- **Lényegesen egyszerűbb, mint a teljes összeadó**
- **A dekrementer ugyanígy származtatható, az INC/DEC pedig az ADD/SUB egységből (nem részletezzük)**



Funkcionális egységek – Verilog leírás

- **Kivonó:**
  - Használjuk a – (kivonás) operátort
  - Nincs átvitel kimenet (co) : hogy ne kelljen, használjunk megfelelő bitszámú kivonót

```
wire [15:0] a, b, diff0, diff1;
wire        cin;
assign diff0 = a - b;         // Nincs cin, nem kell cout
assign diff1 = a - b - cin;   // Van cin, nem kell cout
```

- **Összeadó/kivonó:**
  - Nincs sem átvitel bemenet, sem átvitel kimenet (co)

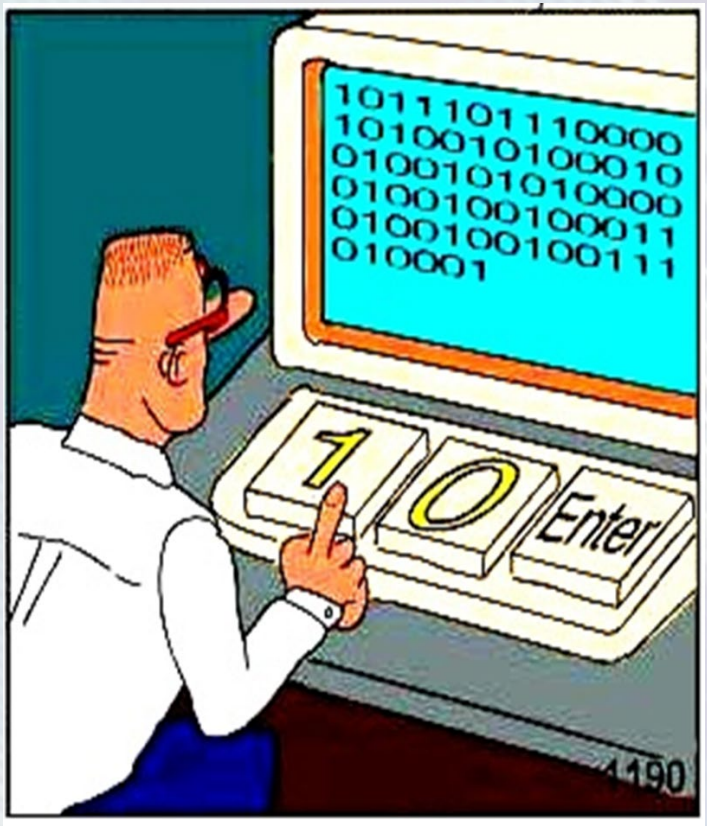
```
wire [15:0] a, b;
wire [15:0] result
wire        sel;
assign result = (sel) ? (a - b) : (a + b);
```

- **Komparátor:**
  - Használjuk a relációs operátorokat

```
wire [15:0] a, b;
wire        a_lt_b = (a < b);   //Kisebb komparátor
wire        a_eq_b = (a == b);  //Egyenlőség komp.
wire        a_gt_b = (a > b);   //Nagyobb komparátor
```

Kérdések, észrevételek

- **A kérdéseket a benes@mit.bme.hu címre várjuk de csak ... @edu.bme.hu címről, tárgy: DIGIT**

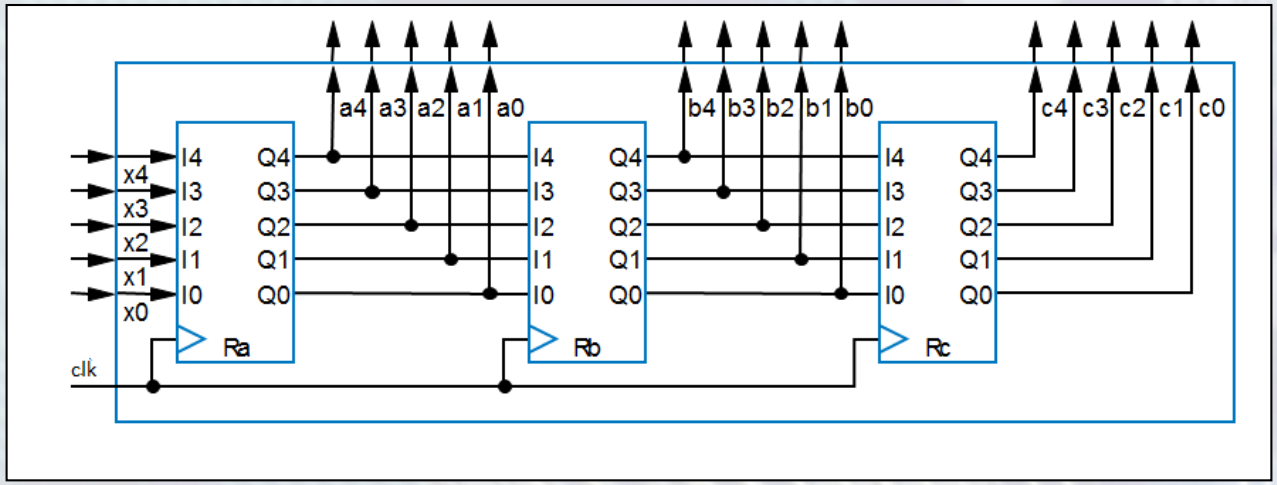


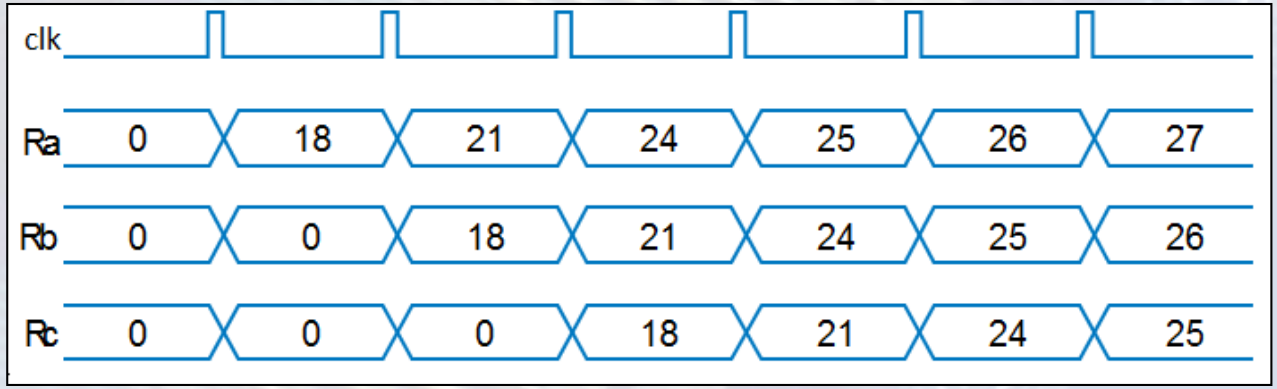
Funkcionális egységek

- **Regiszterek, adattárolók használata**
- **Eddig olyan regiszter alkalmazásokat néztünk, melyek egyetlen adat kezelésével foglalkoztak**
  - Egyszerű párhuzamos regiszter, S/P shiftregiszter
- **Vannak több regisztert tartalmazó gyakorlatban fontos adattároló struktúrák (léteznek SW-ben is)**
  - Regiszteres késleltető sor
  - Regiszter tömb
- **Nem regiszter alapú adattárolók: memória**
  - ROM, (EEPROM, Flash),
  - RAM (aszinkron) nem részletezzük
  - RAM (szinkron)

Funkcionális egységek

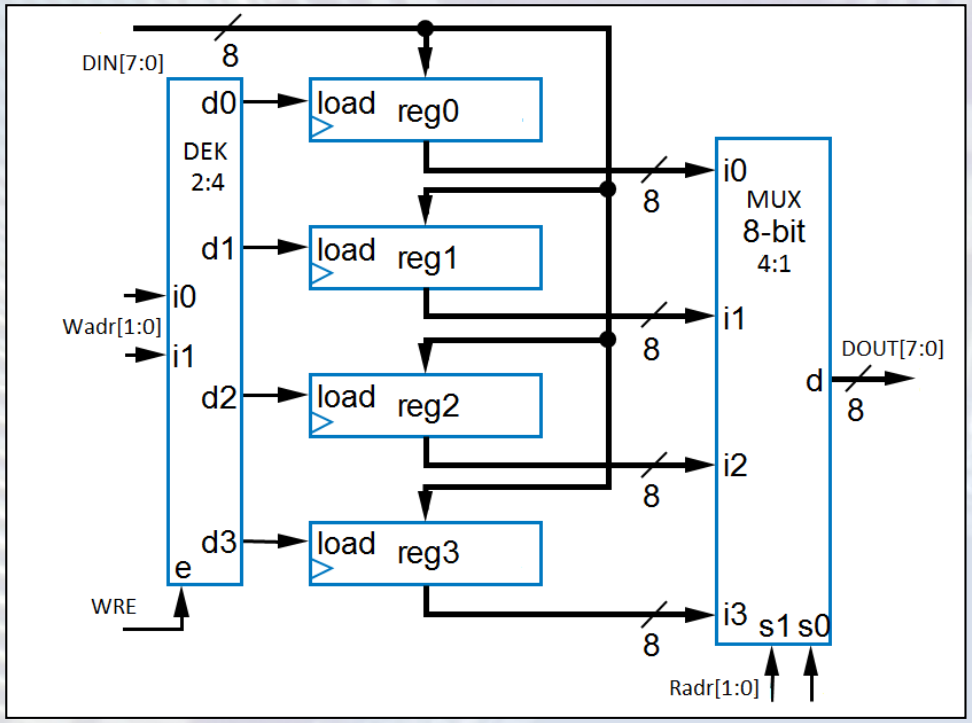
- **Regiszteres késleltető sor**
- A bemeneti adatokból minden órajelben mintát vesz, a legutolsó n db mintát tárolja





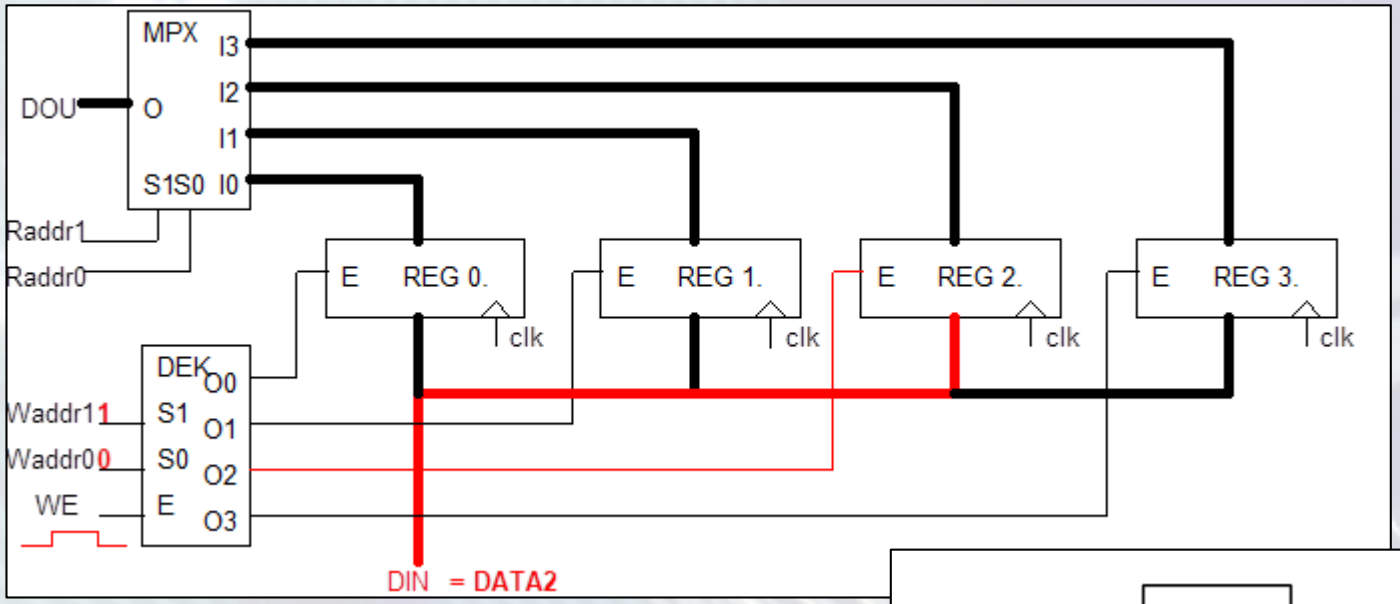
Funkcionális egységek

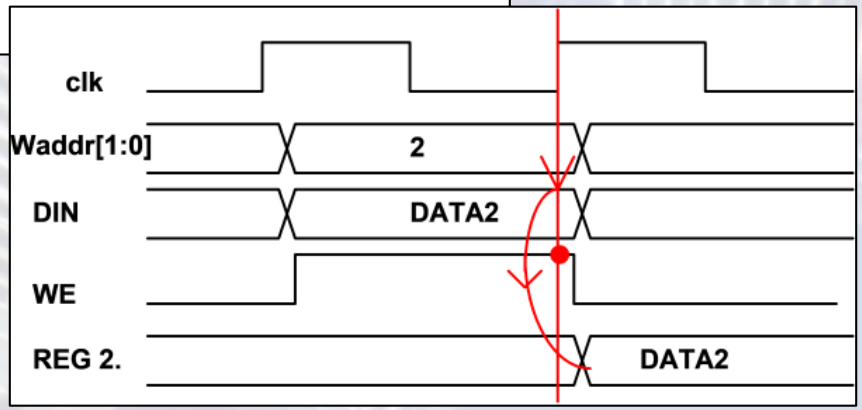
- **Regisztertömb (az adatszélesség 8-16-32 bit)**
- **Több regiszterből álló egység (ált. 2^n, 4-8-16-32-64 reg)**
- **Tárolóegység kiválasztása az írási és olvasási címmel**
  - Írási cím: az írás engedélyező jel kiadása (DEK)
  - Olvasási cím: kimeneti adat kiválasztás (BUS_MUX)
  - Címbitek: 2,3,4,5,6
- A regiszterek egyedileg írhatók, olvashatók, egyidőben csak egy
- Írás szinkron, CLK felfutó élre, ha WRE = 1
- Olvasás aszinkron, azonnal megjelenik



Funkcionális egységek

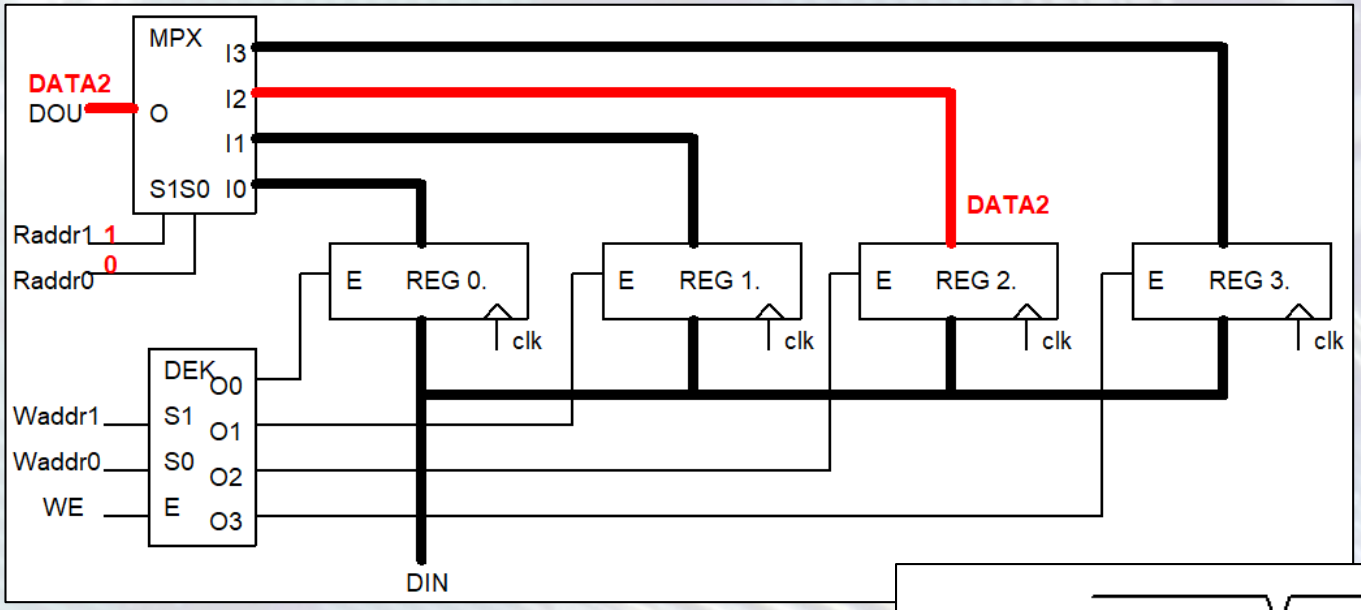
- **Regisztertömb írása (pl. a Waddr = 2 címre)**
- **Az írás szinkron, CLK felfutó élre, ha WRE = 1**

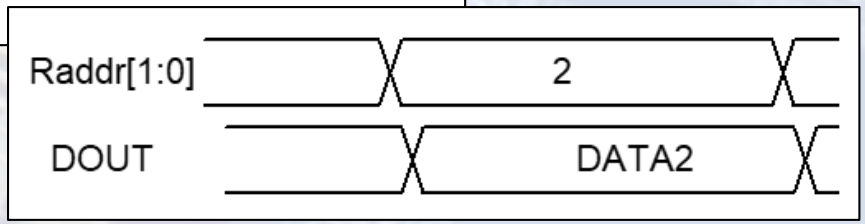




Funkcionális egységek

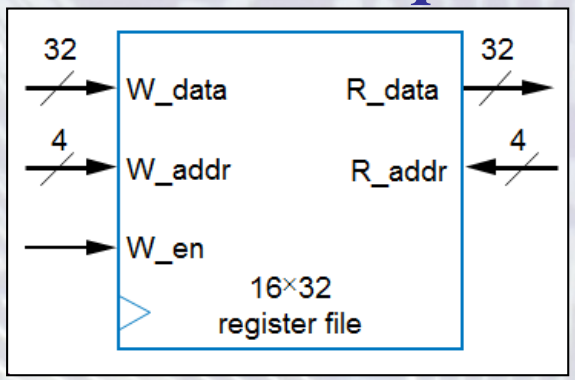
- **Regisztertömb olvasása (pl. az Raddr = 2 címről)**
- **Az olvasás aszinkron: a cím megváltozása után kis késleltetéssel megjelenik a kiválasztott regiszterben tárolt adat a kimeneten.**

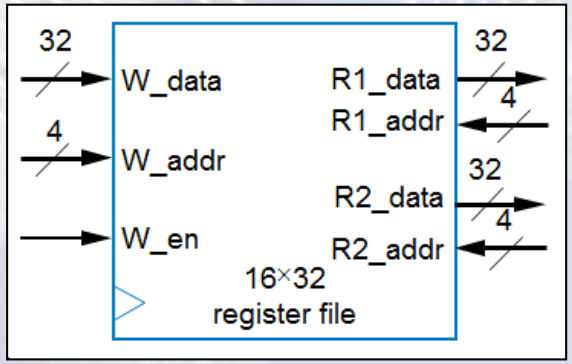




Funkcionális egységek

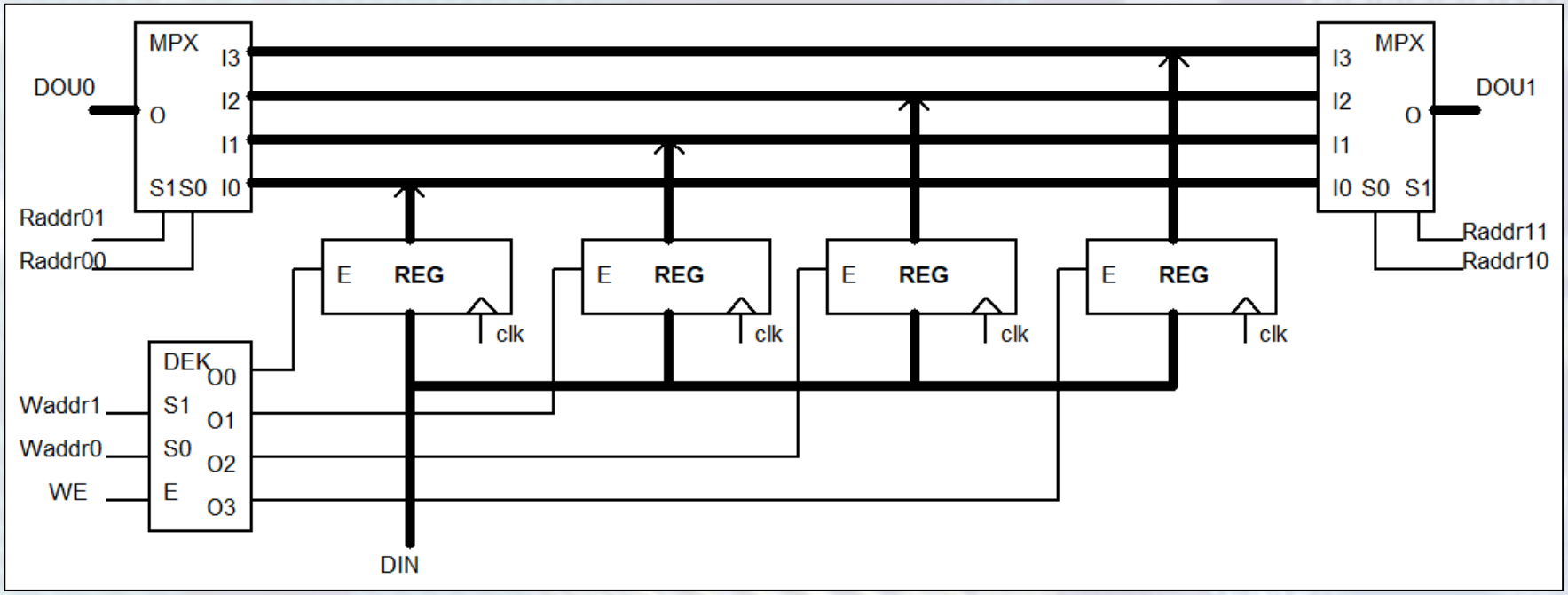
- **Regisztertömb**
  - A CPU egységekben az elsődleges adattárolók
  - Az ALU operandusa(i) és az eredmény tárolóhelye, ennek megfelelően ***független írási és olvasási portok***
- **Típusai:**
- **1W1R: Egy írási cím (W_addr) és egy olvasási cím (R_addr)**
- **1W2R: Egy írási és két olvasási cím**
  - Csak az olvasási multiplexert kell duplikálni és egy sokkal rugalmasabb elemet kaptunk
    ERED = OP1 műv_kód OP2
    (WR) (RD1) (RD2)





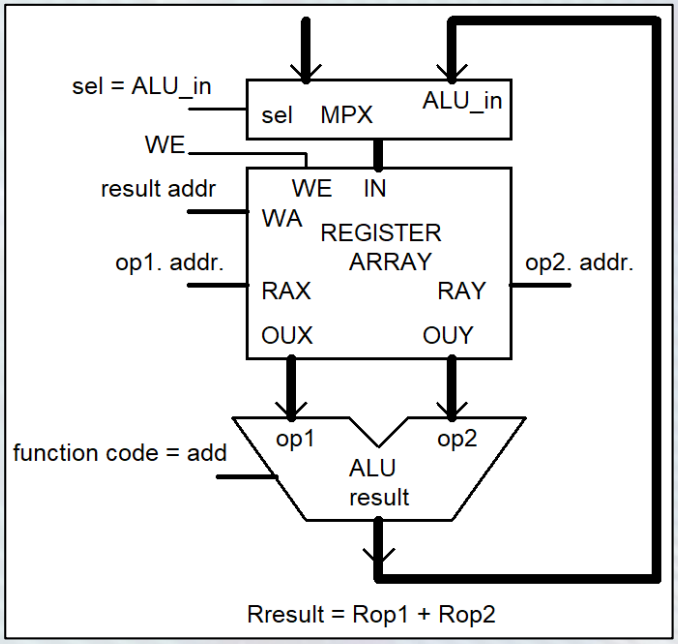
Funkcionális egységek

- **Regisztertömb egy írási és két olvasási porttal**
- **Csak az olvasási multiplexert kell megduplázni**

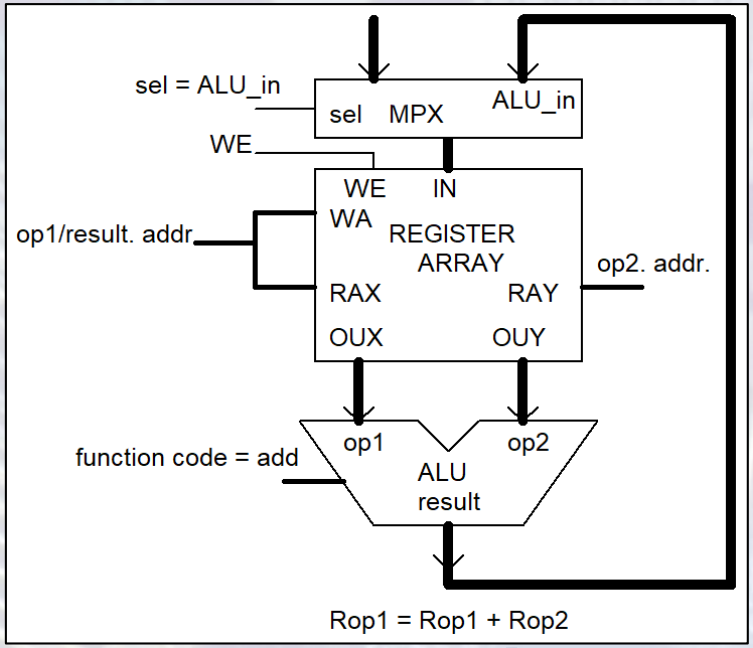


Funkcionális egységek

- **Regisztertömb alkalmazási példa: processzor műveletvégző egysége**
- **A műveletet az ALU (Arithmetic Logic Unit) végzi el**



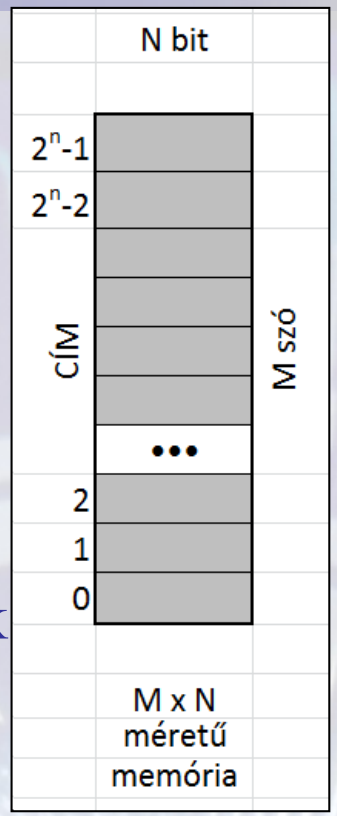
3 regiszter címes
R3 = R1 + R2



2 regiszter címes
R2 = R1 + R2

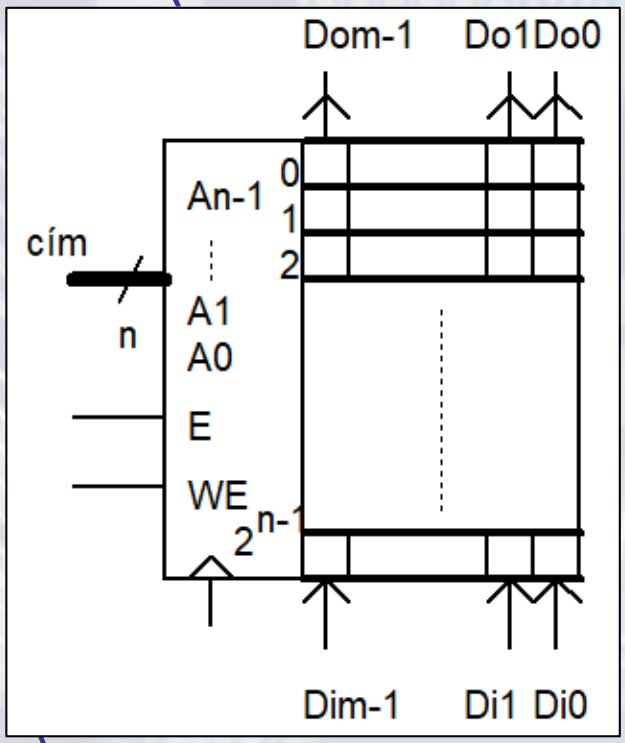
Memória

- **Logikailag a regisztertömbre hasonlít**
- **Dimenzióban jelentősen eltér**
- **Jellemző méretek:**
  - Adatszélesség N: (4), 8-16-32 bit
  - Adatszavak száma M: $2^{10} – 2^{20}$ …
    (1Ki – 1Mi adatszó), technológia függő
    - Hogyan lehetséges? Nem DFF bittároló,
      hanem kifejezetten a legegyszerűbb megoldások
      a nagy adatsűrűség érdekében (6T, 1T)
- **Fő típusok (használat szerint):**
  - **ROM:** ***csak olvasható memória*** (Read-Only Memory), az adatok programozással kerülnek bele (mit jelent?)
  - **RAM:** ***írható-olvasható memória*** (Random Access Memory), tetszőlegesen elérhető (címezhető) memória



Memória – Szinkron RAM

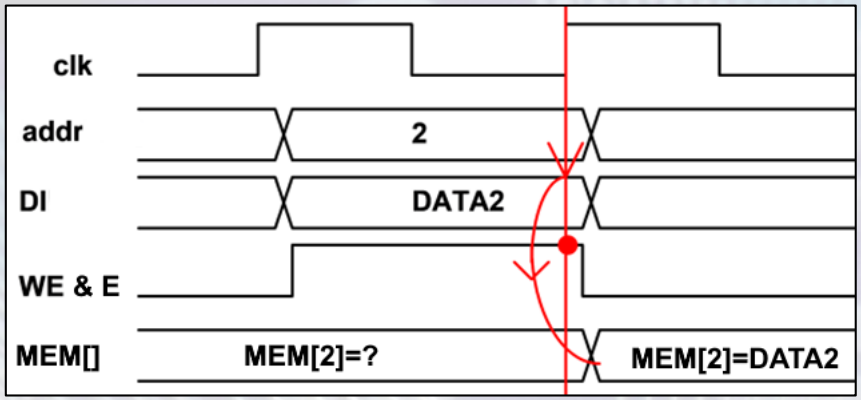
- **A műveleteket az órajel ütemezi**
  - Az írás mindig szinkron
  - Az olvasás lehet aszinkron vagy szinkron (a szinkronnal nem foglalkozunk)
- **Bemenetek és kimenetek**
  - ***CLK***: órajel
  - ***ADDR***: írási/olvasási cím
  - ***E***: a működést engedélyező jel
    - Ha 0, akkor nincs állapotváltozás
    - Szinkron olvasású típusnál szokásos
  - ***WE***: az írást engedélyező jel
  - ***Di***: írási adatbusz (adat bemenet)
  - ***Do***: olvasási adatbusz (adat kimenet)



Memória – Szinkron RAM

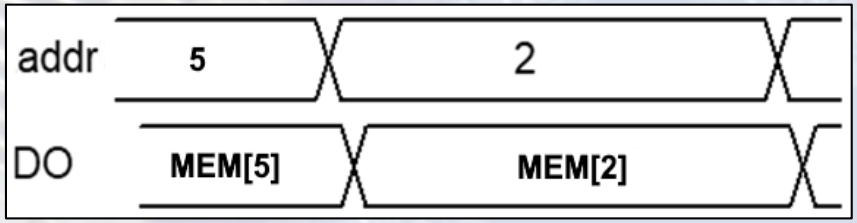
Szinkron RAM írása

- Írás: ha E=1 és WE=1, akkor az órajel felfutó élére Di beíródik az ADDR címre



Aszinkron olvasás

Az ADDR megváltozása után a Do kimenet kis késleltetés-sel felveszi az adott címen tárolt adat értékét



Memória – Szinkron RAM

- **Memóriák az FPGA-ban**
  - Elosztott RAM (Xilinx Spartan-3E FPGA család)
    - Kisméretű adatok tárolására (1 blokk: 16 x 1 bit, pl. reg. tömb)
    - Szinkron írás, aszinkron olvasás
    - 1 írási port és 2 olvasási port külön cím bemenetekkel
  - Blokk RAM (Xilinx Spartan-3E FPGA család)
    - Nagyobb méretű adatok tárolására (1 blokk: 16 kbit + 2 kbit) jobban nem részletezzük

Memória – Verilog leírás

- **Memóriák leírása Verilog nyelven:**
  - A memória tekinthető egy egydimenziós tömbnek
    - **WIDTH:** ***egy szóban lévő bitek száma***
    - **WORDS:** ***a memóriában lévő szavak száma***, melynek **kettő hatványnak** kell lennie
  - A memória adatot tárol (állapottal rendelkezik), ezért regiszter típusúnak kell deklarálni

```
reg [WIDTH-1:0] mem [WORDS-1:0];
```

- **A Xilinx FPGA-k kétféle típusú memóriát tartalmaznak**
  - Elosztott RAM (distributed RAM)
  - Blokk RAM (block RAM)
- **Mindkét memória külön adat bemenettel és kimenettel rendelkezik**

Memória – Verilog leírás

- **Az elosztott RAM Verilog leírása:**
  - Példa: 32 x 4 bites RAM 1 írási és 2 olvasási porttal
  - Írás: szinkron → always blokkban megvalósítva
  - Olvasás: aszinkron → wire típusú kimenet

```
reg  [3:0] mem [31:0];  //4 bit szószélesség, 32 db memória rekesz
wire [4:0] addr_a;      //Cím az írási és az 1. olvasási porthoz
wire [4:0] addr_b;      //Cím a 2. olvasási porthoz
wire [3:0] din;         //A beírandó adat
wire       write_en;    //Írás engedélyező jel

//Írási port (szinkron)
always @(posedge clk)
  if (write_en)
    mem[addr_a] <= din;

//Olvasási portok (aszinkron)
wire [3:0] dout_a = mem[addr_a];
wire [3:0] dout_b = mem[addr_b];
```

Memória – Verilog leírás

- **A memória tartalmának inicializálása külső adatfájlból**
  - Az adatfájl soronként 1 bináris vagy hexadecimális stringet tartalmaz
  - A fájlban lévő sorok számának azonosnak kell lennie a memóriában lévő szavak számával
  - Az inicializáláshoz használjuk a ***$readmemb*** (bin. adatfájl) vagy a ***$readmemh*** (hex. adatfájl) Verilog függvényeket ***initial*** blokkon belül

```
$readmemb("adatfájl", ram_név, kezdőcím, végcím);
$readmemh("adatfájl", ram_név, kezdőcím, végcím);
```

- ***Initial*** **blokk:**

```
initial
    hozzárendelések
```

  - Nem használható hardver komponensek megvalósításához
  - Az ***initial*** blokkon belüli hozzárendelések a szintézis vagy a szimuláció kezdetén értékelődnek ki
  - Az ***if…else***, a ***case*** és a ***for*** utasítások használhatók az ***initial*** blokkban
  - Több utasítás esetén a ***begin*** és az ***end*** kulcsszavakat kell használni

Memória – Verilog leírás

- **Példa: a 2k x 16 bites RAM tartalmának inicializálása**

```
reg  [15:0] mem [2047:0];
wire [10:0] wr_addr;      //Írási cím
wire [10:0] rd_addr;      //Olvasási cím
wire [15:0] din;          //A beírandó adat
reg  [15:0] dout;         //Adatkimenet (szinkron olvasás -> reg)
wire        write_en;     //Írás engedélyező jel

//A RAM tartalmának inicializálása
initial
   $readmemh("mem_data_hex.txt", mem, 0, 2047);

//Írási és olvasási portok (szinkron)
always @(posedge clk)
begin
  if (write_en)
    mem[wr_addr] <= din;

  dout <= mem[rd_addr];
end
```

A fájl tartalma:

0x000:	01a5
0x001:	d2f8
0x002:	1342
0x003:	6a18
0x004:	4209
0x005:	ffff
0x006:	89ab
0x007:	5566
⋮	⋮
0x7fe:	99aa
0x7ff:	abcd

Memóriák – Verilog leírás

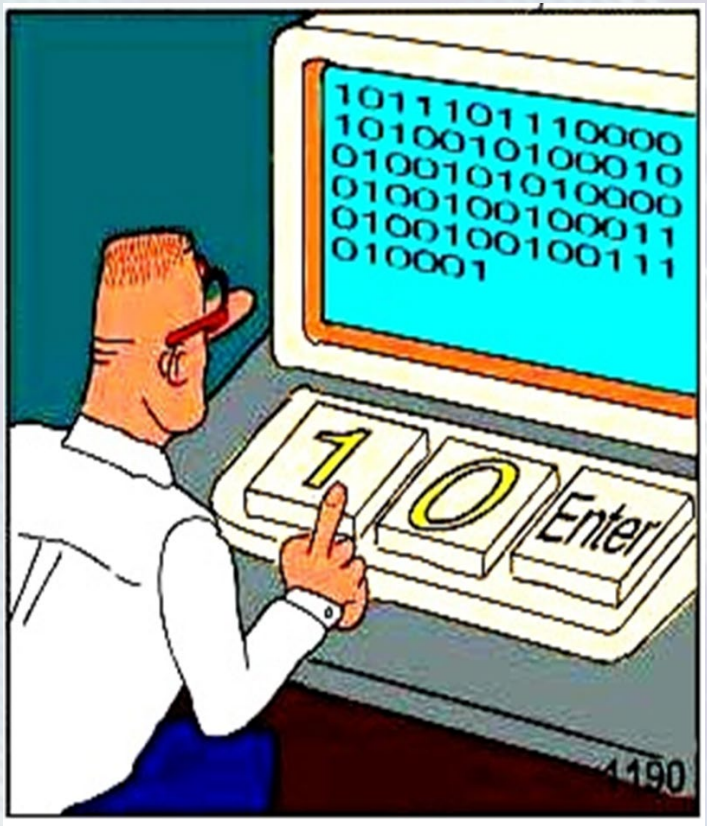
- Kisméretű ROM-ok leírhatók a *case* utasítás segítségével (mint a 7 szegmenses dekódernél, az is tekinthető ROM-nak):

```
wire [2:0] rom_addr;  //8 x 8 bites ROM
reg  [7:0] rom_dout;

always @(*)
   case (rom_addr)
      3'd0:    rom_dout <= 8'b1010_1010;
      3'd1:    rom_dout <= 8'b1111_1000;
      3'd2:    rom_dout <= 8'b0010_0000;
      3'd3:    rom_dout <= 8'b1110_0011;
      3'd4:    rom_dout <= 8'b0000_0000;
      3'd5:    rom_dout <= 8'b0010_1110;
      3'd6:    rom_dout <= 8'b1011_1011;
      3'd7:    rom_dout <= 8'b1111_1011;
      default: rom_dout <= 8'b0000_0000;
   endcase
```

- Írási port nélküli elosztott RAM és a blokk RAM használható ROM-ként
  - Azonban a memória tartalmát inicializálni kell!

Kérdések, észrevételek

- **A kérdéseket a benes@mit.bme.hu címre várjuk de csak ... @edu.bme.hu címről, tárgy: DIGIT**

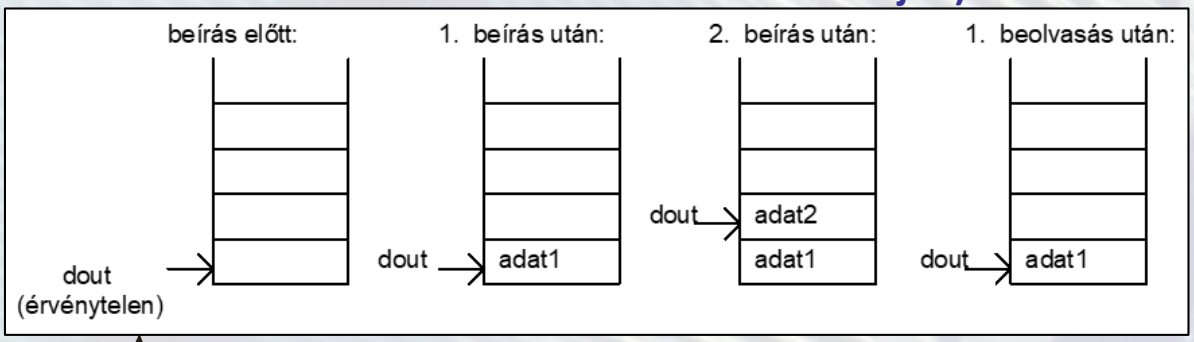


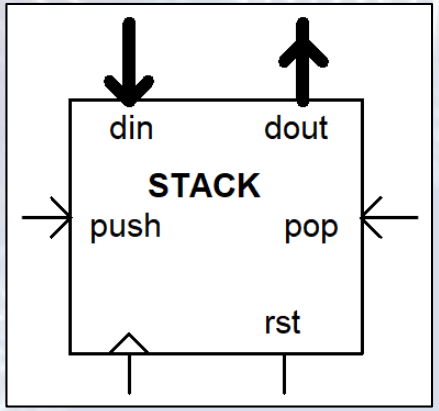
Speciális adatszerkezetek

- **Két fontos általános célú eszköz:**
  - Veremtár (stack) LIFO (Last-In-First-Out)
  - Sor (queue) FIFO (First-In-First-Out)
- Ezek az adatszerkezetek **kis méretben realizálhatók közvetlenül regiszterekben**
- Nagyobb méretben **memóriában, megfelelő vezérlő logikával kiegészítve**
- Ugyanakkor nagyobb rendszerekben gyakori a **szoftveres realizáció is a számítógép rendszermemóriáját használva**
- **Mi most a kisméretű, autonóm, hardver alapú megoldásokat mutatjuk be**

Speciális adatszerkezetek
Verem (stack, LIFO, zsákmemória)

- **A legutoljára beírt adat olvasható belőle ki először.**
- Nincs cím bemenet, csak **2 vezérlő jel: Push** (beírás), **Pop** (kiolvasás).
- A **dout kimeneten olvasáskor (Pop) a legutoljára beírt adat jelenik meg.**
- **Push: adat beírása a verem tetejére, az eddig tárolt adatok egy szinttel lejjebb lesznek találhatók**
- **Szinkron stack-nél a beírás a Push művelet alatti órajel aktív élére történik meg.**
  - Tele állapot esetén Push tiltott művelet
- **Pop: adat elvétele a verem tetejéről, Pop alatt a dout-on a stack tetején levő adat jelenik meg. A Pop alatti órajel aktív élére az adatok eggyel felfele lépnek.**
  - A Pop művelet destruktív, a kiolvasott adat törlődik a tárból
  - Üres állapot esetén a Pop tiltott művelet
- **A STACK az állapotáról nem ad státusz jelzéseket (a használójának kell figyelni, hogy üres STACK-ből ne olvasson és tele STACK-be ne írjon)**





Speciális adatszerkezetek - Verem

- **Megvalósítási lehetőségek:**
  - Multifunkciós regiszterekkel: minden szinten egy 4:1 MUX a regiszterek bemenetén (kétirányú SHR)

STACK VEZ		MUX VEZ		STACK REGISZTER MŰVELET
PUSH	POP	S1	S0	
0	0	0	0	TART
0	1	0	1	TÖLT FELÜLRŐL
1	0	1	0	TÖLT ALULRÓL
1	1	0	0	TART

  - Memóriával és speciális címaritmetikával
- **Minden regiszter művelet természetesen közös felfutó órajel élre történik**

Speciális adatszerkezetek - Verem

Megvalósítás multifunkciós regiszterekkel (4x4 bit)

```
////////////////////////////////////////////////
// 1. verzió
// A tárolókat regiszterek valósítják meg,
// a szintek közötti adatmozgatások
// explicit előírásával
// Egyszerű leírás, a működés könnyen érthető.
// Nagyobb komplexitás esetén nehézkes
// a leírás (sokat kell gépelni, téveszthetünk)
////////////////////////////////////////////////
module stack(
    input clk,
    input rst,
    input push,
    input pop,
    input  [3:0] din,
    output [3:0] dout
   );

// Belső regiszterek definiálása
reg [3:0] level0, level1, level2, level3;

// A stack kimeneti adata az első szint tartalma
assign dout = level0;
```

```
always @ (posedge clk)
   if (rst)
       begin level0 <= 4'b0;
             level1 <= 4'b0;
             level2 <= 4'b0;
             level3 <= 4'b0;    end
   else
    begin
      if (push & ~pop)
       begin level0 <= din;
             level1 <= level0;
             level2 <= level1;
             level3 <= level2;   end
      else
      if (~push & pop)
       begin level0 <= level1;
             level1 <= level2;
             level2 <= level3;
             level3 <= 4'b0;     end
   end
```

Speciális adatszerkezetek - Verem

Megvalósítás memóriával és címmutatóval (16x4 bit)

- 16x4 bites elosztott RAM, 4 bites kétirányú CNT
- Írás az aktuális címre (a következő üres helyre), majd a cím növelése (poszt inkremens címzés)
- Olvasásnál visszalépés az utolsó beírt adatra, majd a csökkentett cím rögzítése (pre dekremens címzés)

```
reg [3:0] mem [15:0];          // 16x4 bit tároló memória a stack adatok tárolására
assign address = (~push & pop) ? cnt - 1 : cnt; // POP-nál auto pre dekremens címzés

always @ (posedge clk)     // A memória is szinkron felfutóél vezérelt az írásra
      if (push & ~pop)  mem[address] <= din;

assign dout = mem[address]; // Az olvasás folyamatos, azaz a kimeneten mindig megjelenik
                            // a megfelelő adat, a POP művelet csak érvényesíti az adat
                            // elvételét, azaz a számlálót lépteti.
```

Speciális adatszerkezetek - Verem

Megvalósítás memóriával és címmutatóval (16x4 bit)

- A számláló bitszámát és a memória méretét átírva tetszőleges méretű verem realizálható

```
reg [3:0] cnt;
wire [3:0] address;

always @ (posedge clk)     // A címszámláló felfutó él vezérelt
   begin
   if (rst)  cnt <= 4'b0; // Aktív resetre törlődik
   else                   // Ha a reset nem aktív, akkor
      begin
      if (push & ~pop)    // érvényes push parancsnál (ha nincs szimultán pop)
            cnt <= cnt+1; // annak végrehajtása után a címszámláló inkrementálódik
                          // de előtte az adatot beírtuk az aktuális címre
                          // (auto post-inkremens beírás!)
      else                // vagy
      if (~push & pop)    // érvényes pop parancsnál (tehát nincs szimultán push)
            cnt <= cnt-1; // a címszámláló dekrementálódik,
      end                 // (az aktuális olvasást eleve erről a címről kellene
   end                    // végrehajtanunk (pre-dekremens olvasás), de ehelyett
                          // az aktuális cnt-1-et használjuk a címzésre és a
                          // órajelciklus végén korrigáljuk a számlálót
```

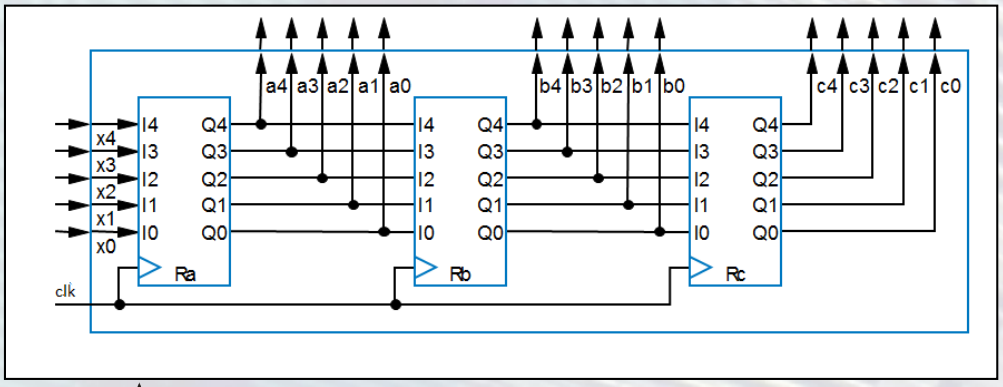
Speciális adatszerkezetek – Sor, FIFO (First-In-First-Out)

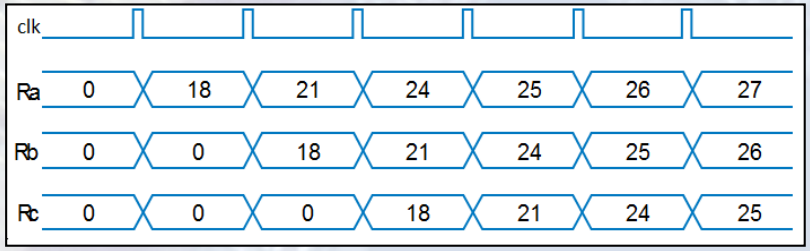
- **Egy olyan csőhöz hasonlítható, amelybe az egyik oldalon (cső eleje, din) betoljuk az adatokat (írás), a másik oldalon (a cső vége, dout) pedig kiszedjük (olvasás)**
- **A legelőször beírt adat olvasható belőle ki először.**
- Nincs cím bemenet, csak **2 vezérlő jrel: WR** (beírás), **RD** (kiolvasás).
- **A dout kimeneten olvasáskor (RD) a legelőször beírt adat jelenik meg.**
- **WR: adat beírása a FIFO elejére, (az eddig tárolt adatok elé).**
- **Szinkron stack-nél a beírás a WR alatti órajel aktív élére történik meg.**
  - Tele állapot esetén WR tiltott művelet
- **RD: adat elvétele a FIFO végéről, RD alatt a dout-on a FIFO végén levő adat jelenik meg. A RD alatti órajel aktív élére a FIFO végéről eltűnik a kiolvasott adat és az 1-el előtte levő lesz a legközelebb kiolvasható.**
  - A RD művelet destruktív, a kiolvasott adat törlődik a tárból
  - Üres állapot esetén a RD tiltott művelet

Speciális adatszerkezetek – Sor, FIFO (First-In-First-Out)

A ***legegyszerűbb verziója a késleltető regiszter sor***

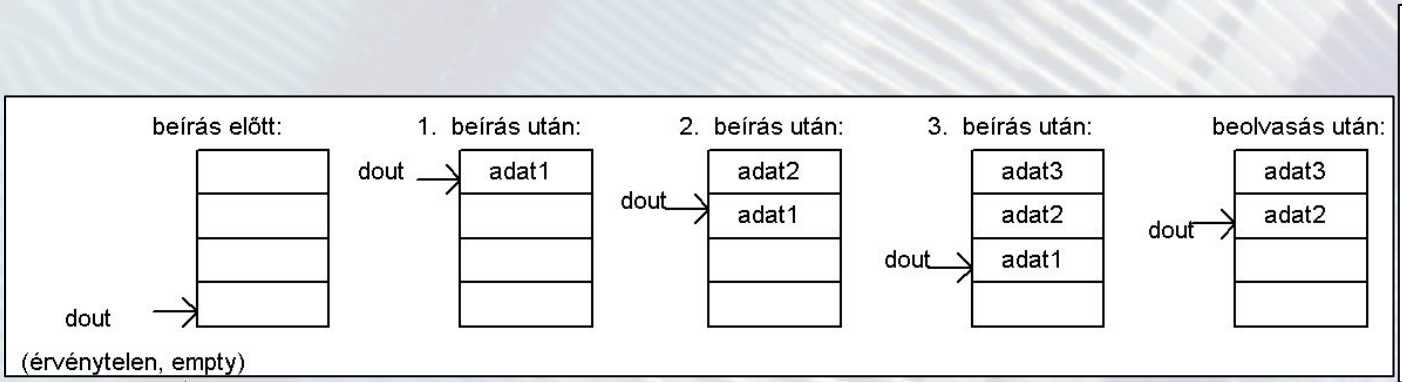
- A bemeneti adatokból minden órajelre mintát vesz és az utolsó N mintát tárolja: x[t-1], x[t-2], …, x[t-N]
- A ***beírás és a kiolvasás folyamatos***, periodikus, egyidejű, ***minden órajel élre megtörténik***. (Itt még nincs beírás és kiolvasás engedélyező jel.)
- Hasznos, de nem tud adatsebességet kiegyenlíteni

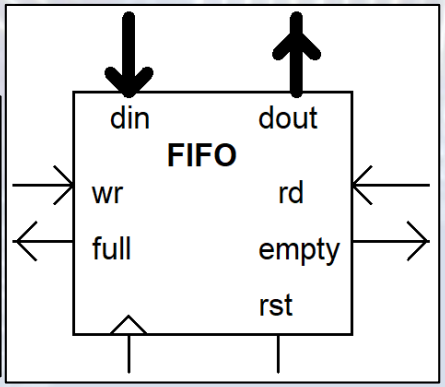




Speciális adatszerkezetek – Sor, FIFO

- **A FIFO egyik fontos alkalmazása: rugalmas puffer**
  - A bemeneti és a kimeneti oldal közötti adatsebesség ingadozás kiegyenlítése (pl. A kiolvasás időnként lassabb, mint a beírás.)
- **Műveletek: írás (Write, Wr, Push) és olvasás (Read, Rd, Pop, Get)** különféle elnevezések
  - Az olvasás a veremhez hasonlóan destruktív
- **Státusz jelzések: üres (empty) és tele (full)**
  - Üres FIFO nem olvasható, tele FIFO nem írható





Speciális adatszerkezetek – Sor, FIFO

FIFO megvalósítása késleltető regiszter sorral:

- **Az alap késleltető regiszter sor egy kis kiegészítéssel**
  - Engedélyezhető beírás: csak ha van új adat
  - Címezhető kiolvasás: a legrégebben beírt adatra mutasson
- **Ez a megvalósítás kis méret esetén használatos**
  - Pl. egy soros kommunikációs interfész adási és vételi pufferéhez egy 8-16 bájtos FIFO elég lehet
    - Vételnél egy-két bájt beérkezése után kiolvassuk
    - Adásnál, ha nincs tele, tehát írható, 4-5 karaktert írunk bele
    - Az állapotjelző bitek (üres, tele) vezérlik a használatot

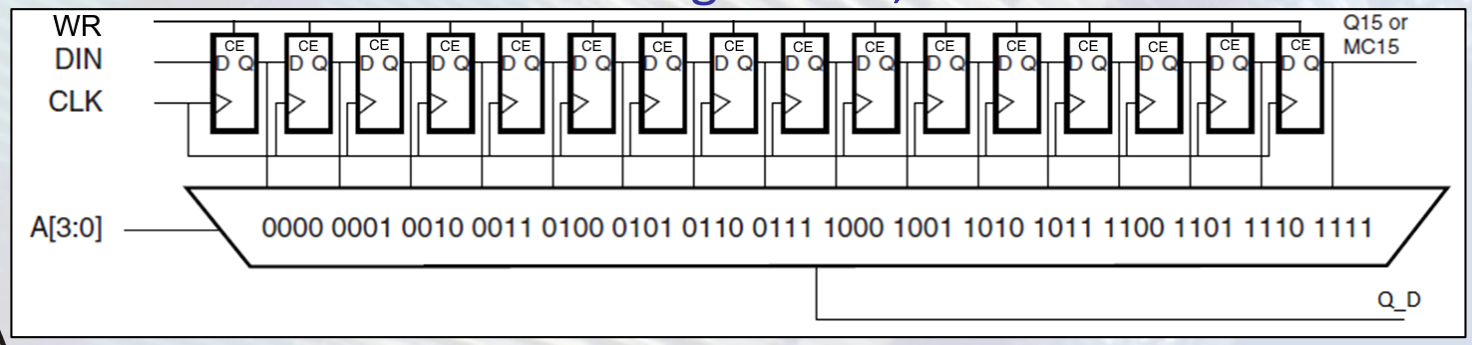
Speciális adatszerkezetek – Sor, FIFO

FIFO megvalósítása késleltető regiszter sorral és multiplexerrel:

- **Írási cím mutató nincs, mindig a sor elejére írunk és íráskor minden adat lép (shiftelődik)**
- **Olvasási cím mutató (A): kétirányú bináris számláló címzi a MPX-ert** (az ábrán a számlálót és vezérlő logikáját nem résztleteztük)
  - Írás esetén növekszik, ha EMPTY=0 és FULL=0
  - Olvasás esetén csökken, ha értéke nem 0
  - Egyidejű írás és olvasás esetén nem változik

Nincs átfordulás!

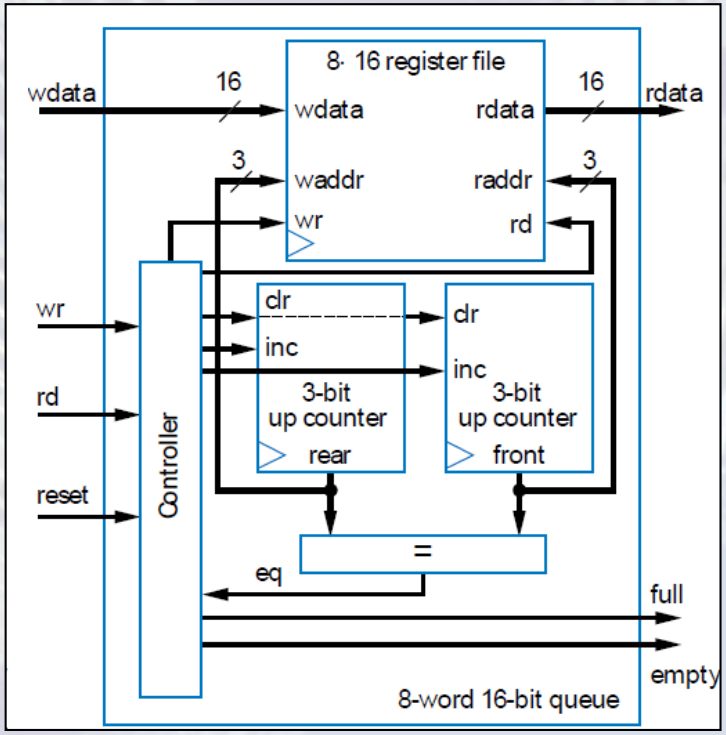
- **Kezdetben EMPTY=1 és FULL=0**
  - Az EMPTY jelzést az első írási művelet törli és a 0. címről történő olvasás állítja be
  - Ha az olvasási cím eléri a végértéket, akkor FULL=1



Speciális adatszerkezetek – Sor, FIFO

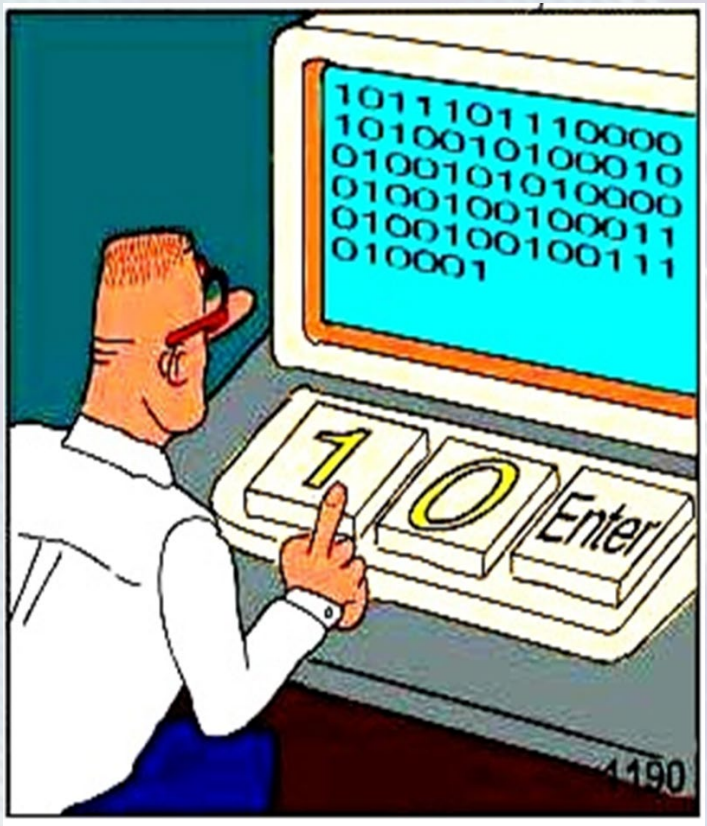
FIFO megvalósítása memóriával vagy regisztertömbbel:

- **A memória alapú FIFO nagyobb méret esetén használatos**
- **Írási port → FIFO bemenet, olvasási port → FIFO kimenet**
- **Írási és olvasási mutatók: bináris számlálók**
  - A példában moduló 8 számlálók 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, …
  - Átfordulnak → körkörös puffer
- **Vezérlés és státusz jelzés a számlálók értéke alapján:**
  - Ha írás után egyenlők, akkor a FIFO megtelt (FULL=1)
  - Ha olvasás után egyenlők, akkor a FIFO kiürült (EMPTY=1)



Kérdések, észrevételek

- **A kérdéseket a benes@mit.bme.hu címre várjuk de csak ... @edu.bme.hu címről, tárgy: DIGIT**



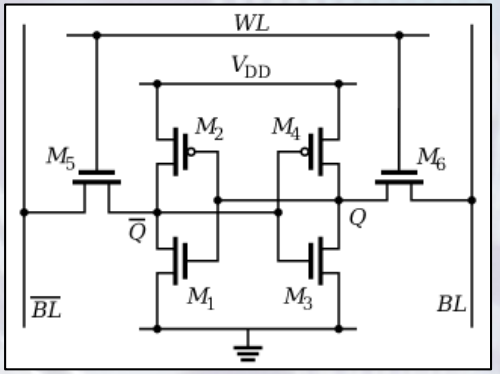
Digitális technika
6. EA vége

A további diák az FPGA-k belső felépítéséről csak tájékoztató anyag, nem része a vizsgának

Memória – RAM típusok

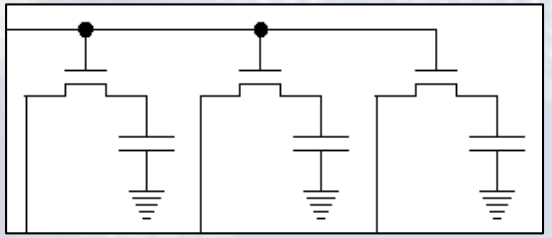
Statikus RAM: az információt latch tárolja

- Bitenként 4 vagy 6 tranzisztor
- Kisebb adatsűrűség, drágább
- Nincs szükség frissítésre



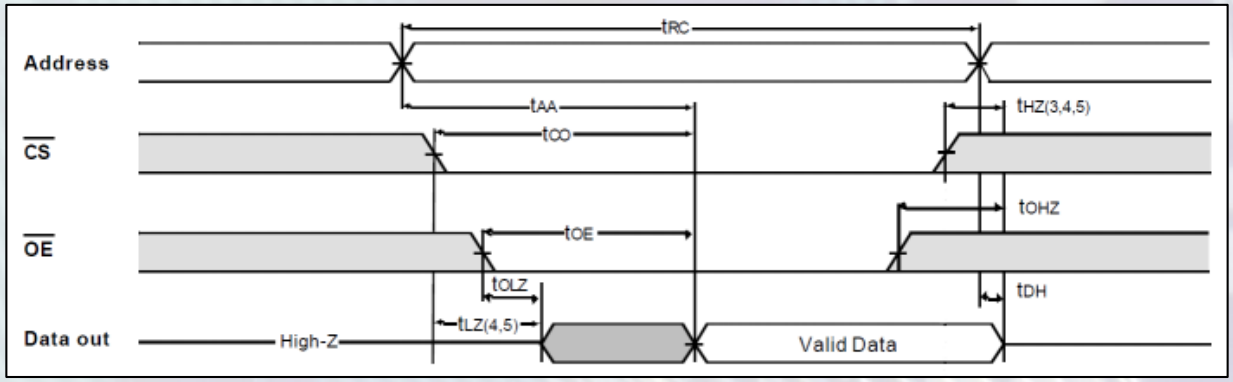
Dinamikus RAM: az információt kapacitás tárolja

- Bitenként egy kondenzátor és egy tranzisztor
- Nagyobb adatsűrűség, olcsóbb
- Frissíteni kell a tárolt tartalmat

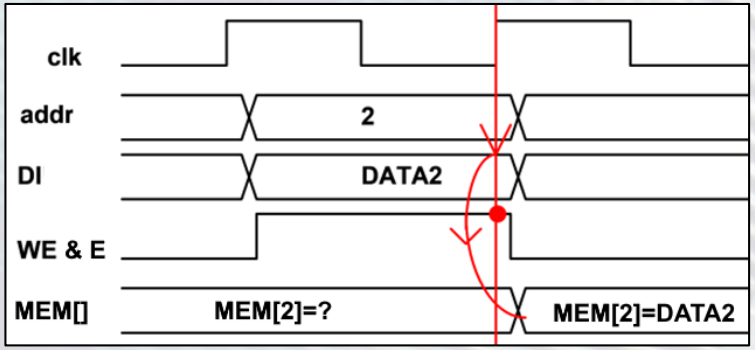


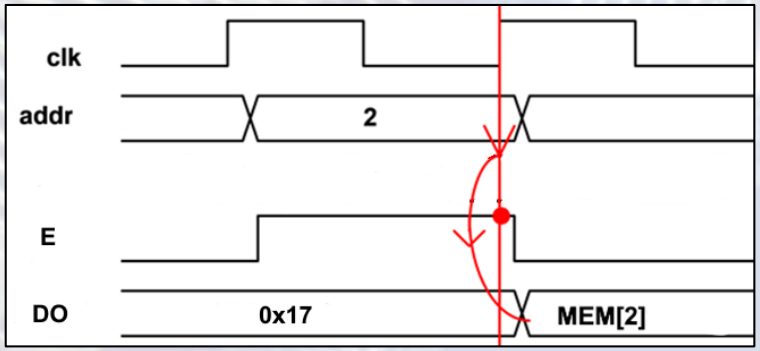
Memória – RAM típusok

Aszinkron RAM: a műveleteket a vezérlőjelek változása ütemezi, nincs órajel



Szinkron RAM: a műveleteket órajel ütemezi





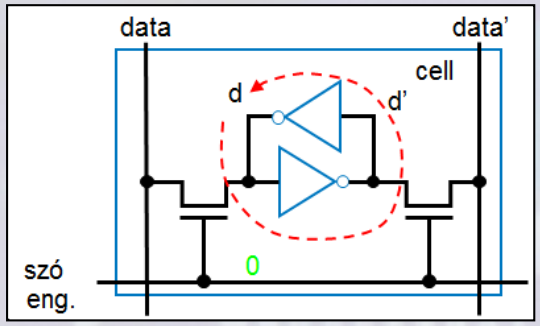
Memória – RAM típusok

	Statikus	Dinamikus
Aszinkron	*FPGA elosztott RAM (olvasás)* (aszinkron SRAM)	(aszinkron DRAM)
Szinkron	*FPGA elosztott RAM (írás)* ***FPGA blokk-RAM*** (QDR SRAM)	(SDRAM) (DDR SDRAM) (DDR2/3/4 SDRAM)

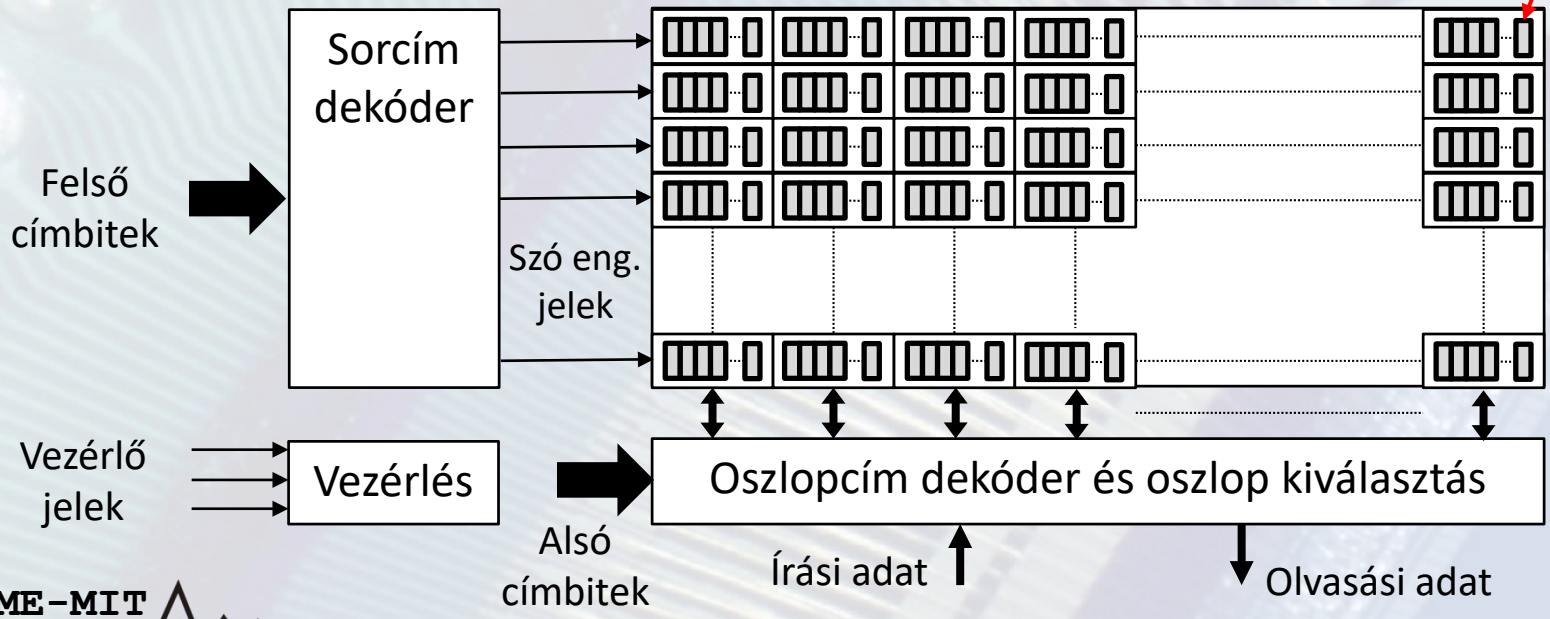
Memória – Belső felépítés

Hasonló a regisztertömb működéséhez, csak…
2D adattároló tömb
2D címdekódolás: sor- és oszlopcím dekóderek
- Hatékonyabb, mint a normál (1D) dekódolás

Memória típusonként eltérő vezérlés



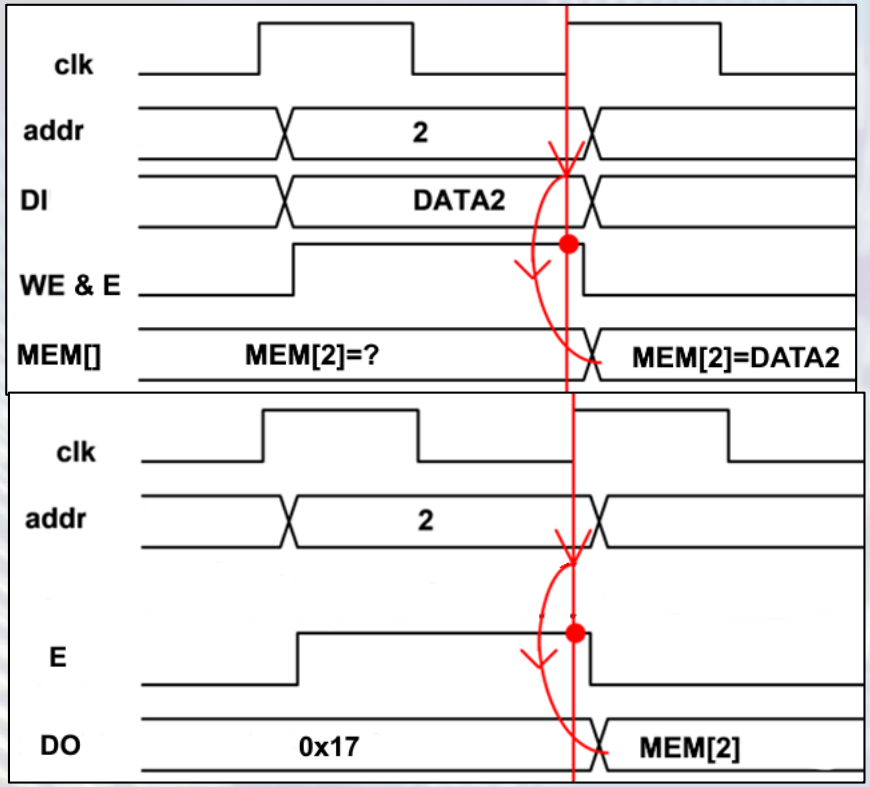
Elemi bittároló cella (SRAM)



Memória – Szinkron RAM

Szinkron RAM írása

Írás: ha E=1 és WE=1, akkor az órajel felfutó élére Di beíródik az ADDR címre



Szinkron olvasás:

- Ha E=1, akkor az órajel felfutó élére Do frissül az ADDR címen tárolt érték-kel
- Ha E=0, akkor Do megtart-ja a korábbi állapotát

Memória – Verilog leírás

A blokk RAM Verilog leírása:

- Példa: 2k x 16 bites RAM 1 írási és 1 olvasási porttal
- Írás: szinkron → always blokkban megvalósítva
- Olvasás: szinkron → always blokkban megvalósítva

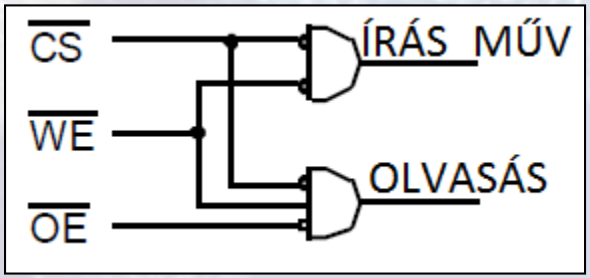
```
reg  [15:0] mem [2047:0];
wire [10:0] wr_addr;      //Írási cím
wire [10:0] rd_addr;      //Olvasási cím
wire [15:0] din;          //A beírandó adat
reg  [15:0] dout;         //Adatkimenet (szinkron olvasás -> reg)
wire        write_en;     //Írás engedélyező jel

//Írási port (szinkron)
always @(posedge clk)
  if (write_en)
    mem[wr_addr] <= din;

//Olvasási port (szinkron), lehetne az írási
//portot megvalósító always blokkban is
always @(posedge clk)
  dout <= mem[rd_addr];
```

Memória

- **Önálló memória IC tokok interfészei**
  - Pl. LOGSYS Spartan3E kártyán felforrasztva
- **SAMSUNG KR1008V1D-UI10 (1Mi bit, 10ns)**
  - 128Ki*8bit High-Speed CMOS Static RAM (3,3V)
  - Címbusz (17 vonal, 2^{17}=128Ki), Adatbusz (8 vonal)
  - Vezérlőjelek (általában negált értelműek)
    - /CS: Chip kiválasztás/engedélyezés

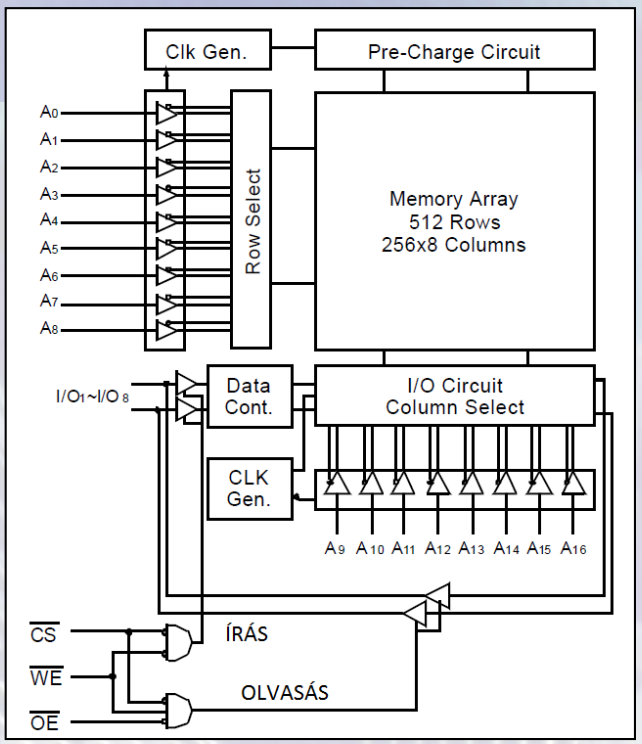
      csökkenti a fogyasztást, nincs működés, ha /CS = 1
    - /OE: Kimenet engedélyezés, ill. leválasztás, HiZ állapot
    - /WE: Írás engedélyezés
  - A vezérlőjeleket a használat során ennek megfelelően kell vezérelni (szerencsére hibatűrő, /WE tilt!)

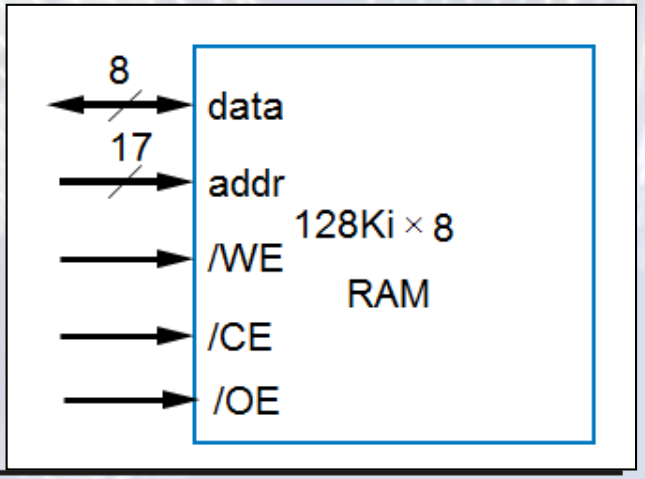
Részlet az adatlapból nagyítva



Memória

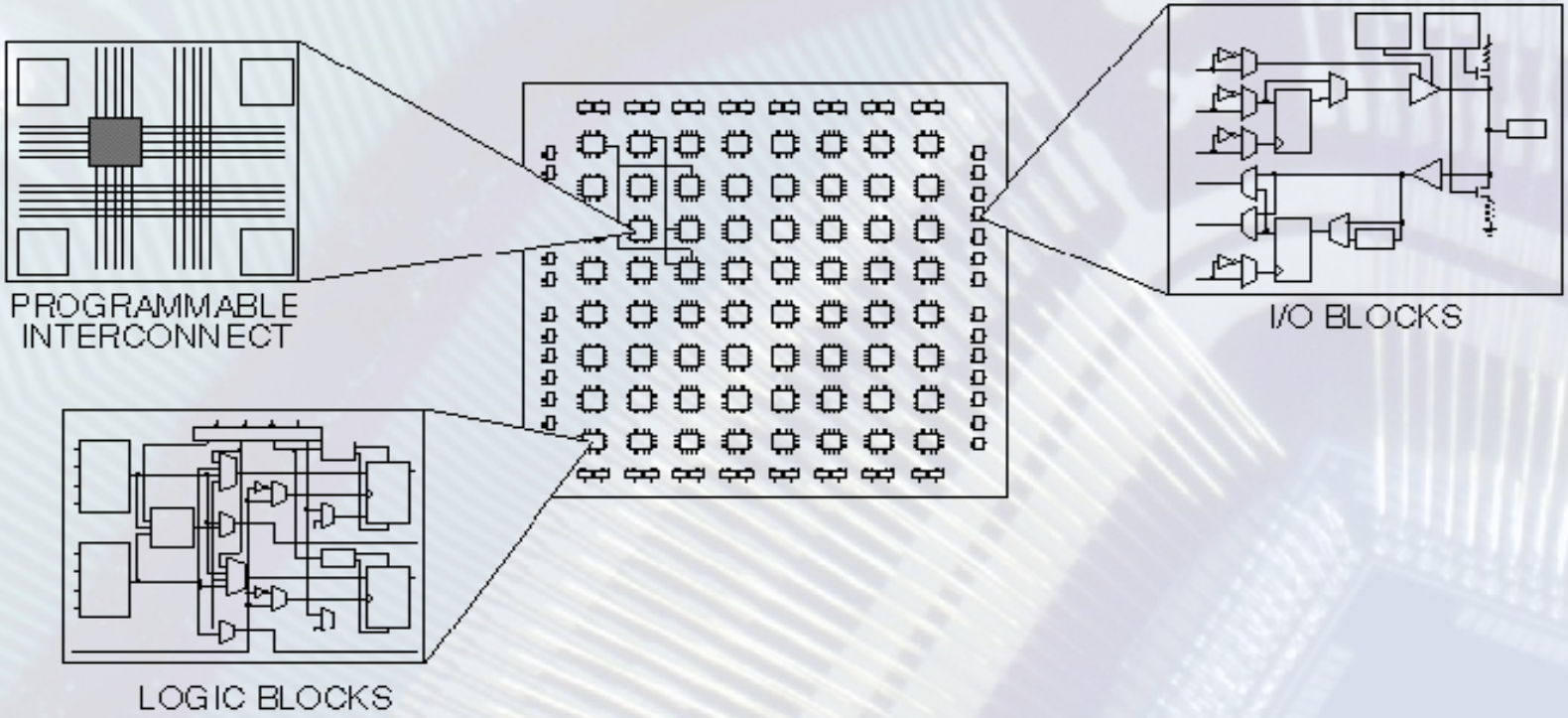
- **128Ki x 8 SRAM blokkvázlata**
  - Látható az I/O adatbusz kétirányú meghajtása, (íráskor a kimenet HiZ, nagyimpedanciás, letiltva)
  - A sok címvonal 2 csoportra van osztva, sor-oszlop címzés, 512x256x8 memória tömb
  - Látható a belső jelek meghajtása, címvonalak ponált/negált értéke
  - (Belső Clk. Gen, és Pre-Charge nem fontos, egyedi specialitás.)
  - Az általános interfész modell az ábrán látható blokk





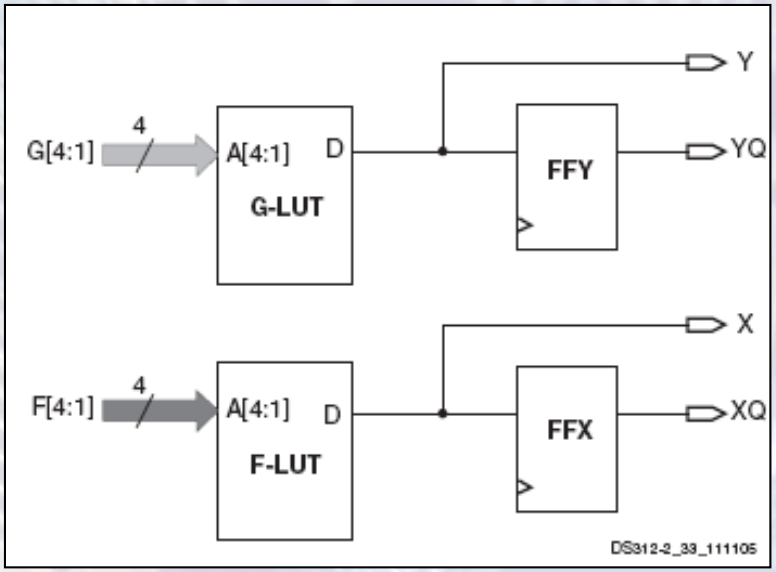
FPGA felépítés

- **Általános felépítés**
  - Logikai blokkok, huzalozás, I/O blokkok



A logikai blokk

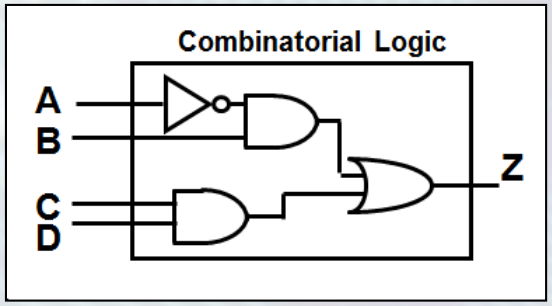
- **A logikai blokk felépítése**
- **Elemi kombinációs és elemi szekvenciális alapelemek**
- **Az alap erőforrás a Logic Cell LC = 1 LUT + 1 FF**
- **LUT4**
  - tetszőleges 4 változós függvény
  - 1 változóra hazárdmentes
  - Végrehajtási idő bemeneti jel és logikai komplexitás invariáns
- **DFF**
  - Élvezérelt, ↑ vagy ↓, órajel eng.
  - Szink/Aszink. SET/RESET



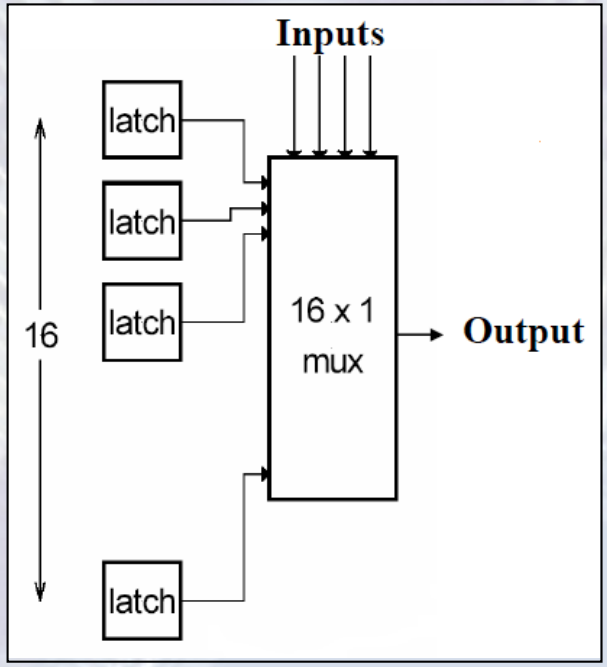
- **Független kombinációs és regiszteres kimenet, vagy 2:1 MUX választ, hogy kombinációs vagy szekvenciális kimenet**

A logikai blokk

- **A LUT4 funkcionalitása: Egy táblázatnak tekintjük**
- **Tetszőleges tartalom betölthető**
  - A komplexitás a bemenetek számában korlátos, nem a logikai függvény összetettségében

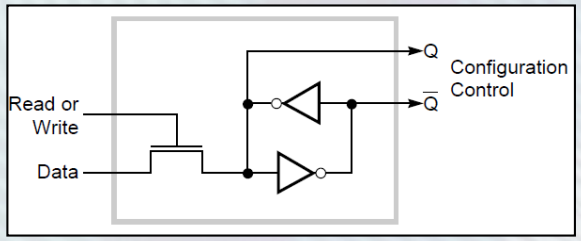


BEMENETEK				KIM
A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

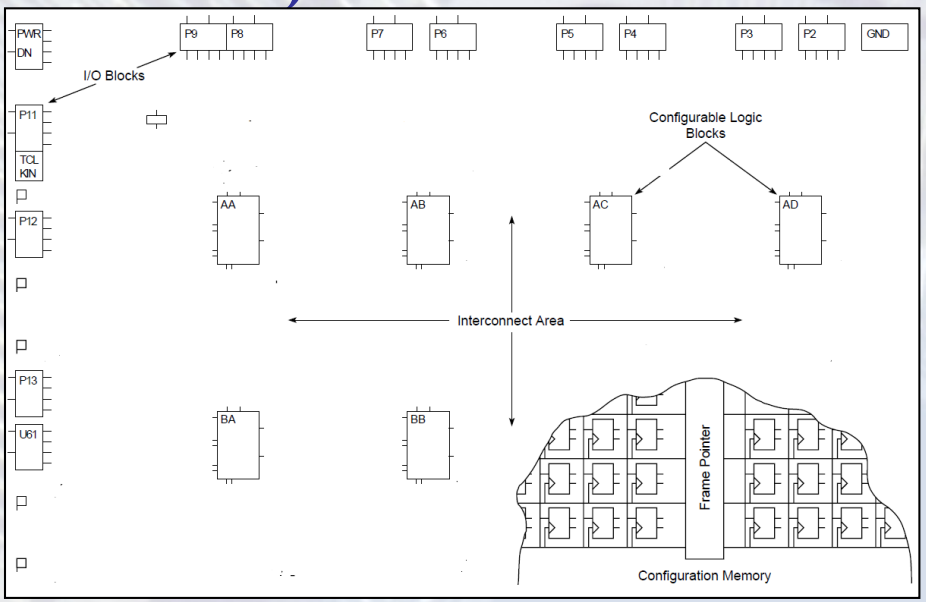


FPGA felépítése

- **A valódi komplexitás részben rejtve van**
- **Két logikai réteg: Konfigurációs + Felhasználói**
  - Mi csak a felhasználóit szeretnénk látni
- **Konfigurációs logika: Shiftregiszter egyszerű SRAM latch tárolókból → A teljes tartalom beléptetése induláskor (CLB, IOB, huzalozás)**

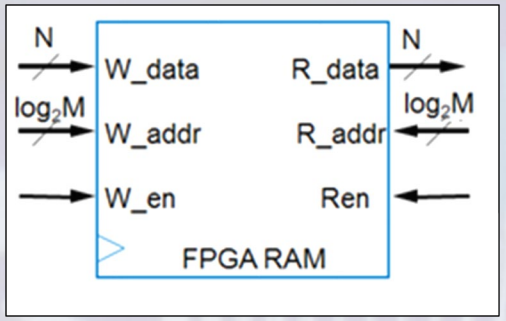


- **Beírás, visszaolvasás, ellenőrzés, indítás**



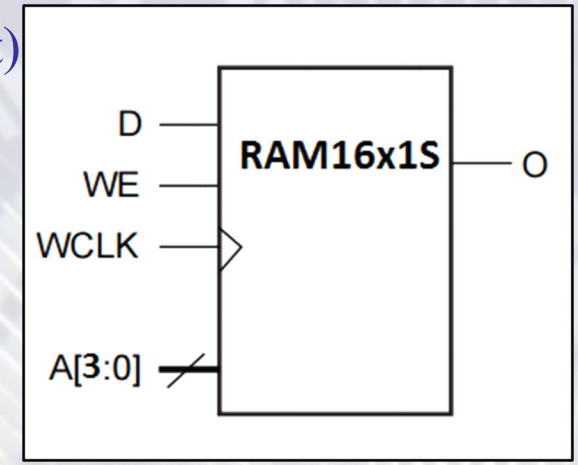
Memória

- **FPGA belső memóriái**
  - Jobban hasonlítanak a regisztertömbre, de ezek is inkább memóriák
  - Lehet egy címbuszuk, vagy kettő
  - Adat interfészük mindig szétválasztott, külön bemeneti (írás) és kimeneti (olvasás) adatbusz
  - Engedélyezés írásra, (esetleg olvasásra)
  - Írás mindig SZINKRON, órajel felfutó élre
- **Memória típusa:**
  - Elosztott memória: kis méretre, tip. max. 256 bájt
  - Blokk memória: 2Ki*(16+2) bites méret
  - RÉSZLETEK A VERILOG HDL diákon



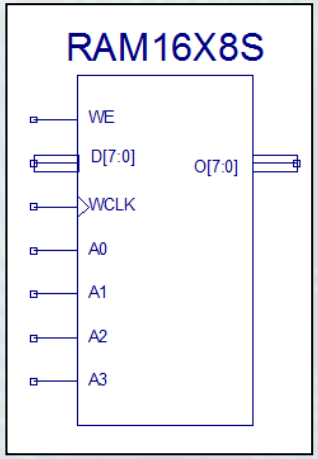
FPGA elosztott memória

- **Kisméretű, belső RAM tároló elem**
- **16x1 bites szelet, ebből tetszőleges méret felépíthető (de általában 2 Ki bit alatt, azaz ~ max. 256 bájt)**
- **1W1R, azaz 1 írás és 1 olvasás portja van**
  - Létezik dual portos verzióban is (1W1R és 1R port)
  - Írás szinkron (WCLK ↑ élre, ha WE = 1)
  - Olvasás aszinkron, az adat azonnal megjelenik
  - Bővítési lehetőségek 2 dimenzióban
    - Adatszélesség (N): Egymás mellé helyezéssel, közös cím és vezérlőjeleket használva, annyi bitet, amennyi kell
    - Adatmennyiség (M): Egymás „fölé” helyezéssel, az írás engedélyezést és a kimeneti adatkiválasztást a 16-os címblokkok dekódolásával megoldva (azaz az A3 feletti címbiteket használva → DEK és MUX.

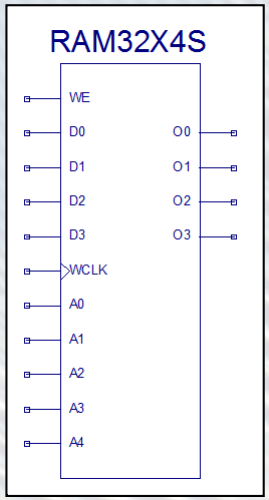


FPGA elosztott memória

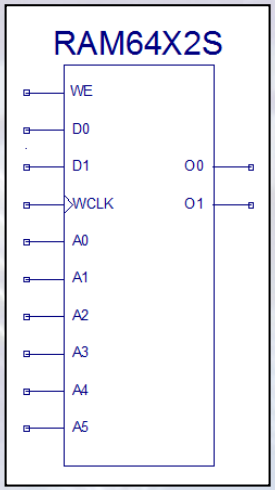
- **Jellemző kialakítások: adatok száma*adatszélesség**
- **Pl. 16x8 bit, 64x2bit, 32x4bit (mind 128 bit kapacitás)**
- **Belső kiegészítő áramkörök:**
  - RAM32X4S: 2:1 cím DEK, 4 bites 2:1 BUSZ MUX
  - RAM64X2S: 4:1 cím DEK, 2 bites 4:1 BUSZ MUX



8db 16x1 elem



2x4 db 16x1 elem

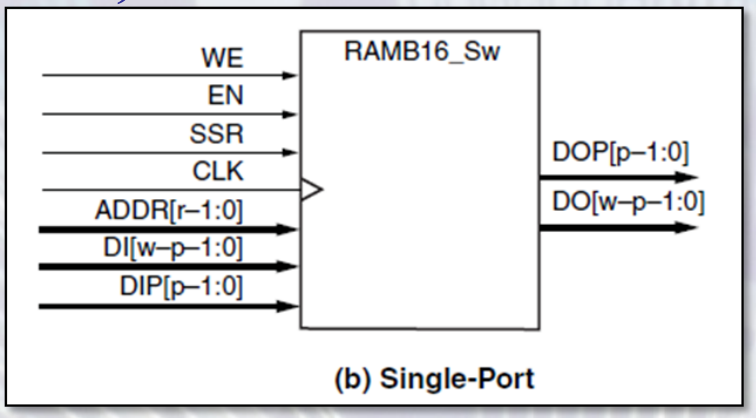


4x2 db 16x1 elem

- RÉSZLETEK A VERILOG HDL diákon

FPGA Blokk memória

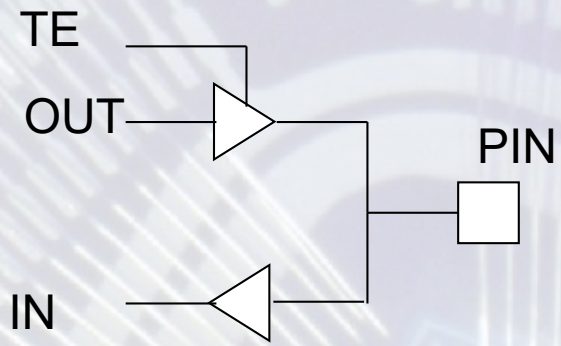
- **Nagyobb méretű teljesen szinkron dual-port RAM (de használható egy portosan is)**
- **Az építőelem 2048x(8+1) bites blokk, azaz 8 bitenként van egy paritásbit is**
- **Lehetséges „területarányok”: 16Ki*1bit, 8Ki*2bit,4Ki*4bit, 2Ki*(8+1) bit, 1Ki*(16+2) bit, 512*(32+4) bit**



- **A két független port mindegyike írás/olvasás képes**
  - Írás/olvasás szinkron (CLK ↑ élre, ha WE = 1 ill. EN = 1)
  - Bővítési lehetőségek a normál memóriákhoz hasonlóan 2 dimenzióban, itt nagy segítség az eleve biztosított „területarány” választási lehetőség
- RÉSZLETEK A VERILOG HDL diákon

I/O funciók

- **Alapvetően minden felhasználói láb I/O, tehát lehet kimenet, bemenet**
- **Nem használt lábak fix értéken**
- **Gyakran többfunkciós lábak**
  - Konfiguráció alatt
  - Normál használat alatt



- **A valódi I/O blokkok sokkal bonyolultabbak**
  - Tartalmaznak bemeneti/kimeneti DFF-okat is
  - Különböző kimeneti opciók (Jelszint, sebesség, meghajtás erősség, felhúzó/lehúzó ellenállás)

Digitális technika
6. EA vége