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

Digitális technika
VIMIAA01

Fehér Béla
BME MIT

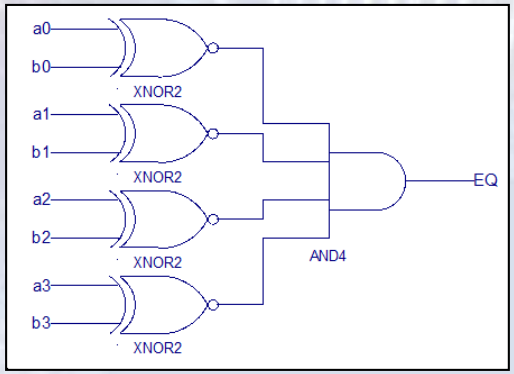
Funkcionális egységek

- **Kiegészítés az eddigi előadások anyagához**
- **Kombinációs funkcionális egységek eddig:**
  - DEK, ENK, PRI, MUX, DEMUX, SHR, ADD
- **További adatfeldolgozási funkciók, melyeket a számlálóknál alkalmaztunk:**
  - Értékfelismerés, adat összehasonlítás - COMP
  - Összeadó/kivonó ADD/SUB
  - Inkrementáló/Dekrementáló INC/DEC
- **Speciális tároló funkcionális egységek**
  - Regisztertömbök
  - Memóriák

Funkcionális egységek

- **Komparátor**
- **Értékek, adatok összehasonlítása**
  - Egyenlőség komparátor
  - Teljes funkciójú komparátor
- **Egyenlőség komparátor**
  - Logikailag a XNOR műveleten alapul
  - Két bitvektor azonos pozíciójú bitjeit vizsgálja, hogy minden biten teljesül-e az egyenlőség feltétel
  - **a**, **b** adatvektorok esetén **n** db két bemenetű XNOR kapu + 1 db **n** bemenetű ÉS kapu
  - Tetszőleges kódolásra működik

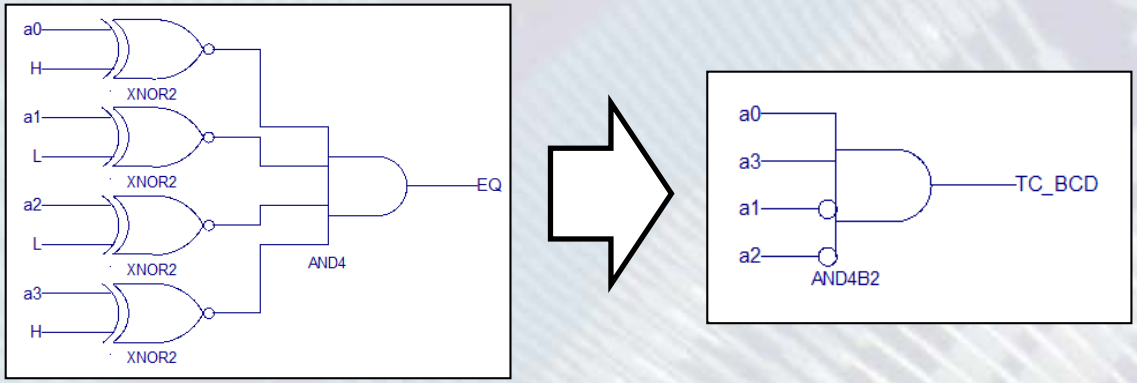
a	b	XNOR
0	0	1
0	1	0
1	0	0
1	1	1



Funkcionális egységek

- **Egyenlőség komparátor**
  - Fix érték vizsgálatára XNOR kapuk egyik bemenete fix 0 vagy 1 → programozott inverter
  - (Pl. a 9 vizsgálata BCD számlálónál)

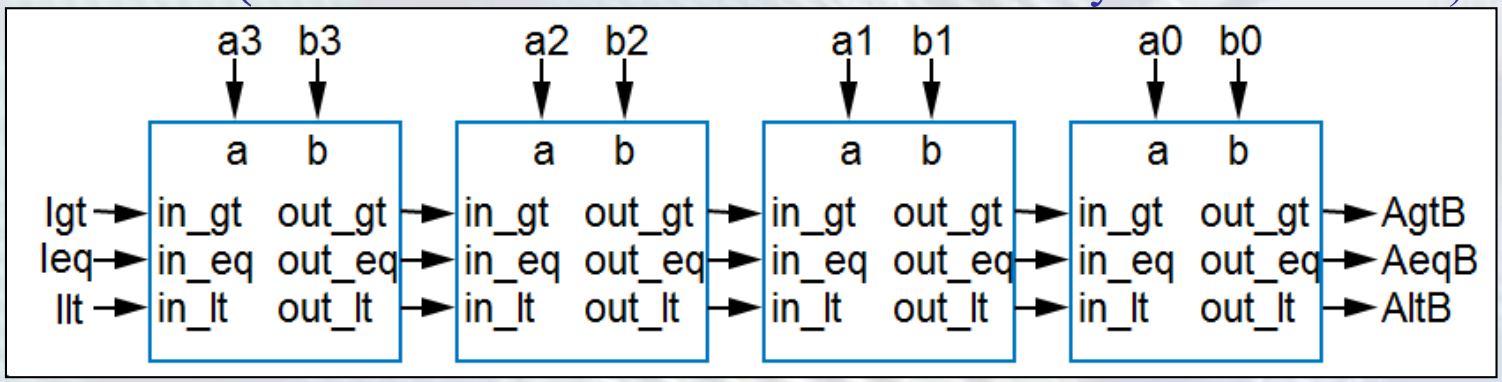
a	FIX	XNOR
a	0	/a
a	1	a



  - Használhatunk komparátort is, de a 4 bemenetű ÉS kapu természetesebb. Ilyenkor azt mondjuk, hogy dekódoljuk a bináris 9 értéket, mert valójában ez a teljes 4 változós függvény m_9 sorszámú mintermje, azaz egy 4:16 dekóder 9. kimeneti jele.

Funkcionális egységek

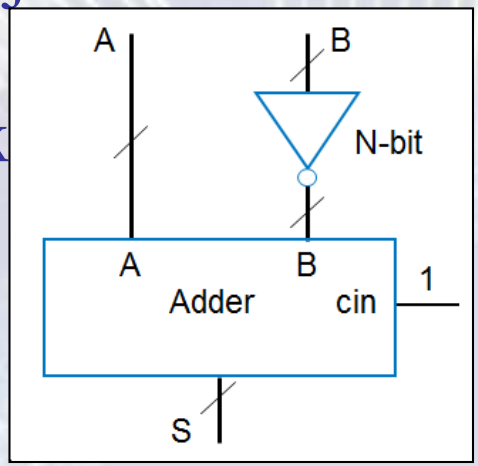
- **Teljes összehasonlító komparátor**
  - Valódi nagyság szerinti összehasonlítás
    - 3 kimenet, ai = bi, ai > bi, ai < bi, (nem függetlenek)
    - „Előző” bitpozícióról hasonló értelmű bemenetek
      - Melyik az előző? Hogyan kaszkádosítsunk? Melyik jobb?
      - Lehetséges MSb → LSb, de LSb → MSb irányba is
        (Az összeadás csak az LSb → MSb irányban működik!!!)



- Tehát egy adott szinten: 3 + 2 = 5 bemenet, 3 kimenet
- MEGTERVEZHETŐ, egy adott bitpozícióra felírhatók az összefüggések → néhány kapu (nem tervezzük meg!)

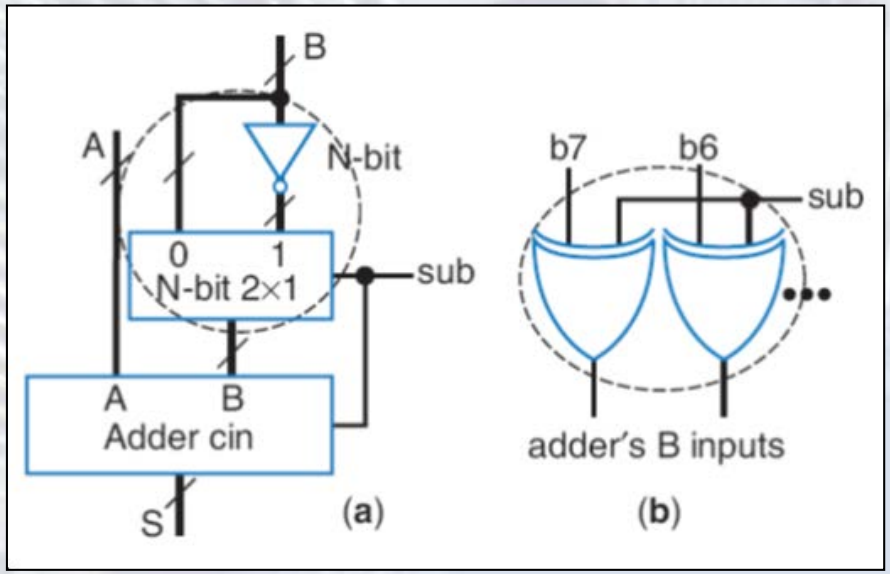
Funkcionális egységek

- **Kivonó**
  - Szerepelt korábban az 1 bites teljes összeadó
  - Ebből kaszkádosítással készítettünk több biteset
  - Az összeadó jól működik pozitív és 2's komplemenes negatív számokra is! (Ezért terjedt el a 2's komplemens számábrázolás)
  - Tudjuk képezni egy szám (-1) szeresét, ezért lehetséges az A-B művelet végrehajtása az ismert A + ((-1)*B) összefüggés alapján.
  - (-1)*B → Minden bitet invertálunk és hozzáadunk 1-et, ami pontosan a 0. pozíció Cin jele.
  - Ezt használjuk komparátor helyett!



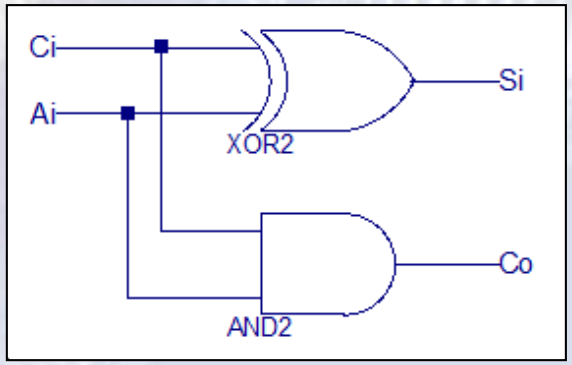
Funkcionális egységek

- **Egységes Összeadó/kivonó**
  - Az előző dia sugallja a megoldást
  - Művelettől függően B normál vagy invertált értéke jut az ADDER bemenetére, mialatt a Cin 0 vagy 1.
  - Megvalósítás: $\overline{ADD}$/SUB vezérlő bemenet $\overline{ADD}$/SUB = 0 összeadás, $\overline{ADD}$/SUB =1 kivonás
  - B bemenet kialakítása: B, $\overline{B}$ és 2:1 MUX
  - XOR kapu, mint vezérelhető INVERTER
  - A Cin közvetlenül az $\overline{ADD}$/SUB vezérlőjel



Funkcionális egységek

- **Inkrementer/Dekrementer**
  - Az összeadó/kivonó alapján
- **Ha B =0 és Cin = 1, akkor S = A + 1**
- **A bitenkénti FADD teljes összeadó egyszerűsíthető → HADD Half ADDER, fél összeadó (csak 2 bemeneti bitje van)**
- **Lényegesen egyszerűbb, mint a teljes összeadó**
- **A dekrementer ugyanígy származtatható**
- **Az INC/DEC pedig az ADD/SUB egységből**

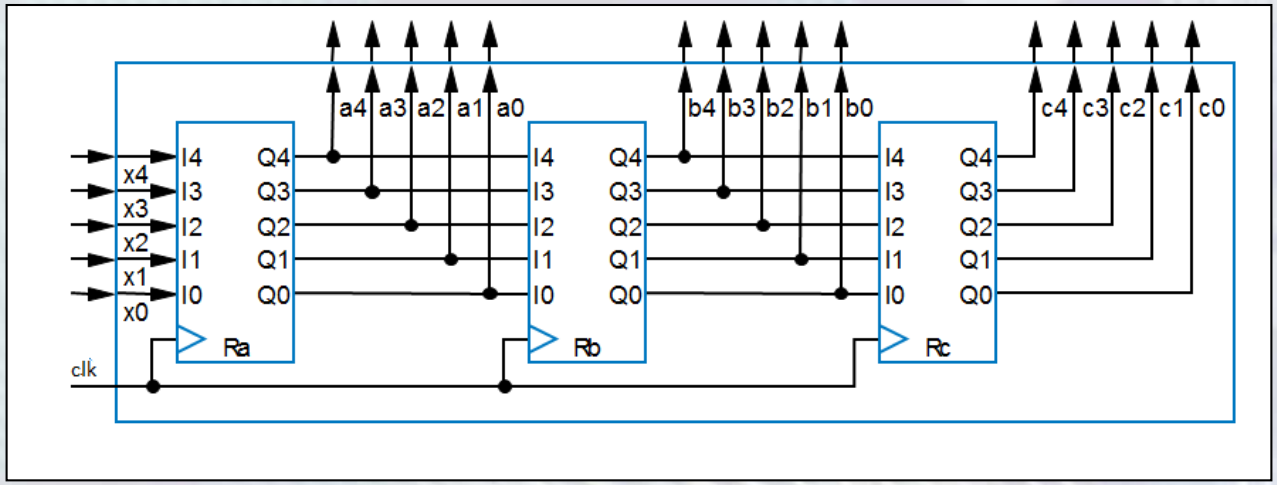


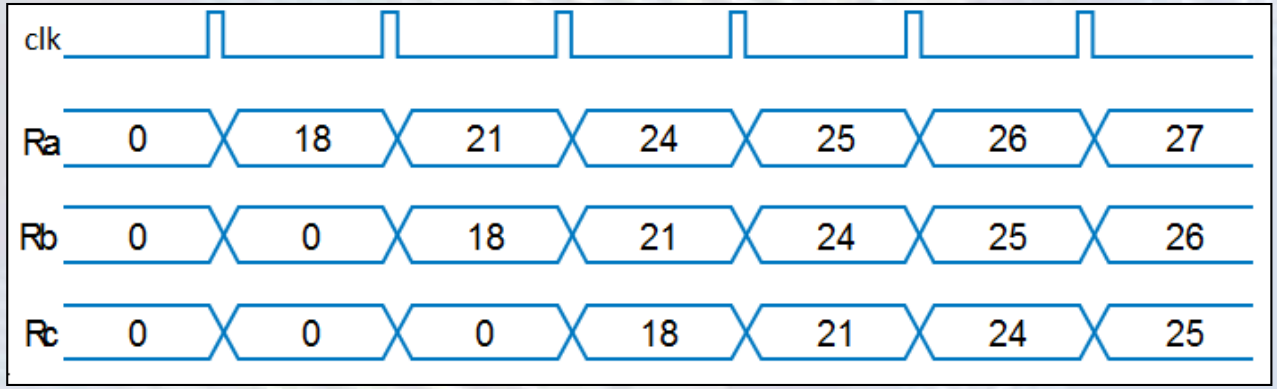
Funkcionális egységek

- **Regiszterek, adattárolók használata**
- **Eddig olyan regiszteralkalmazásokat 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)
  - RAM (szinkron)

Funkcionális egységek

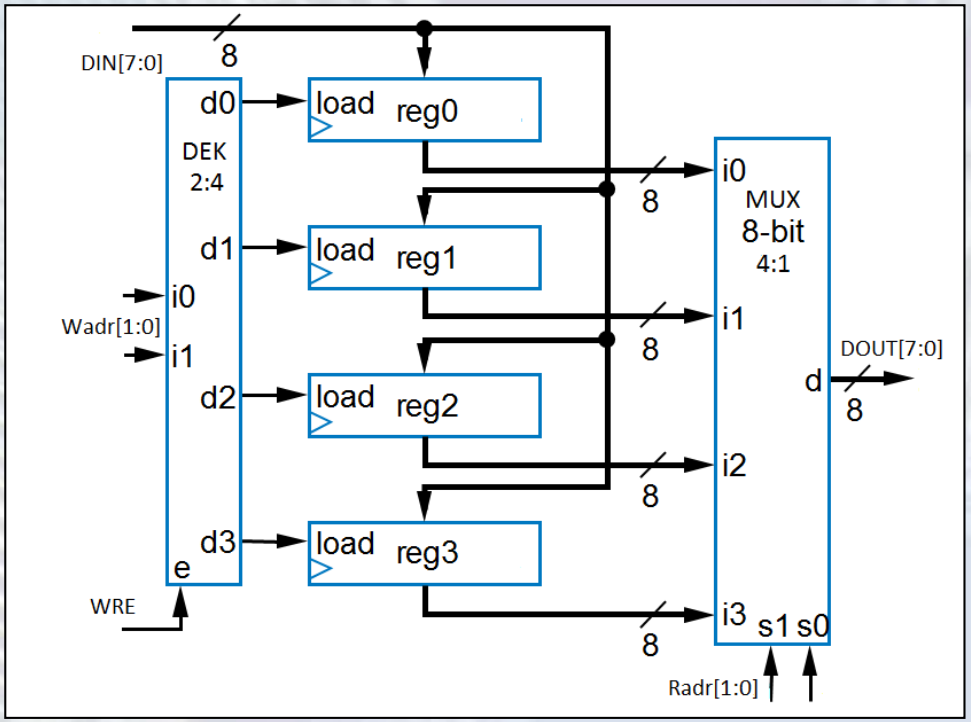
- **Regiszteres késleltető sor**
- **A bemeneti adatokból minden órajelben mintát vesz, a legutolsó n mintát tárolja**





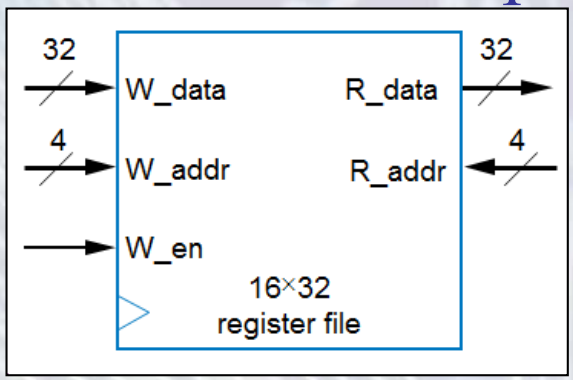
Funkcionális egységek

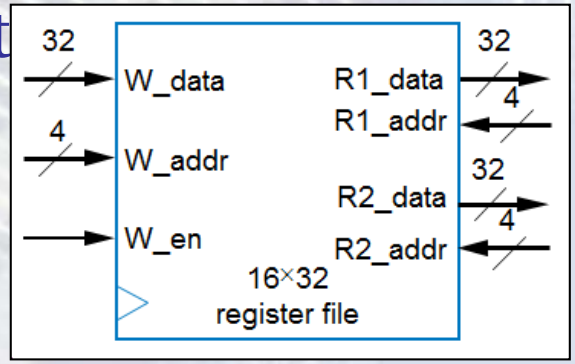
- **Regisztertömb**
- **Több regiszterből álló egység (ált. 4-8-16-32-64)**
- **Kiválasztás az írási és olvasási címmel**
  - Írási cím: Az írás engedélyező jel kiadása (DEK)
  - Olvasási cím: Kimeneti adat kiválasztás (BUS_MUX)
  - Címbitek: 2,3,4,5,6
- **A regiszterek egyedileg írhatók, olvashatók**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



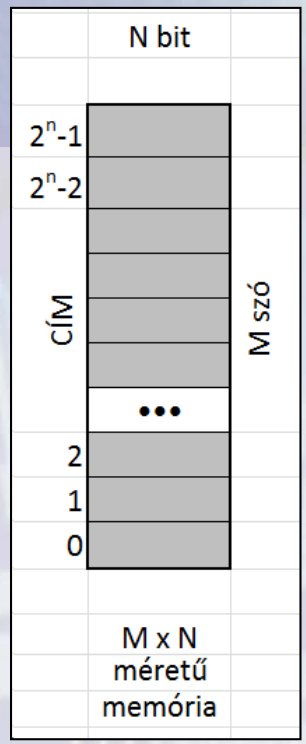
Funkcionális egységek

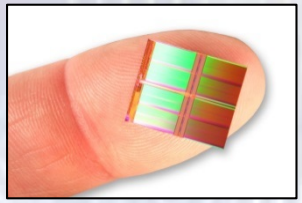
- **Regisztertömb**
  - A CPU egységekben az elsődleges adattárolók
  - Az ALU operandusa(i) és az eredmény tárolóhelye, ennek megfelelően független írási és olvasási portok
- **Típusai:**
- **1W1R: Egy írási cím és egy olvasási cím**
- **1W2R: Egy írási és két olvasási cím**
  - Csak az olvasási multiplexert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





Memória





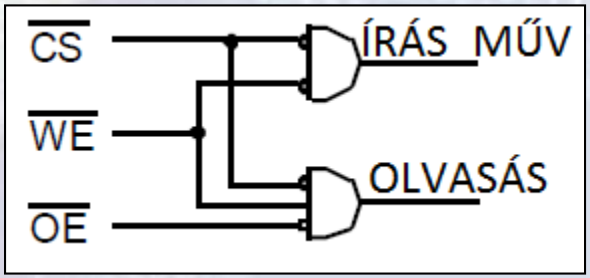
- **Logikailag a regisztertömbre hasonlít**
- **Dimenzióban jelentősen eltér**
- **Jellemző méretek:**
  - Adatszélesség: (4), 8-16-32 bit
  - Adatszavak száma: $2^{10} – 2^{20}$, (1Ki – 1Mi adatszó), technológia függő
    - Hogyan lehetséges? Nem DFF bittároló, hanem kifejezetten a legegyszerűbb megoldások 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

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

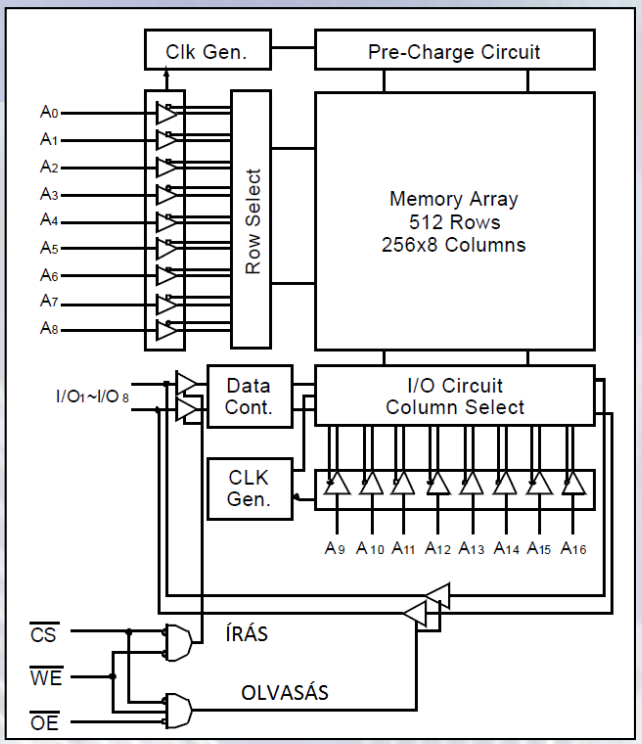
      csökkenti a fogyasztást, nincs működés, ha /CS = 1
    - /OE: Kimenet engedélyezés, ill. leválasztás, HiZ állapot
    - /WE: Írás engedélyezés
  - A vezérlőjeleket a használat során ennek megfelelően kell vezérelni (szerencsére hibatűrő, /WE tilt!)

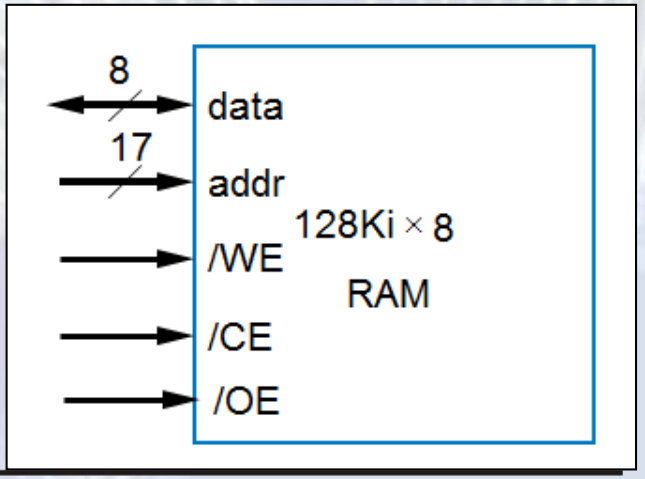
Részlet az adatlapból nagyítva



Memória

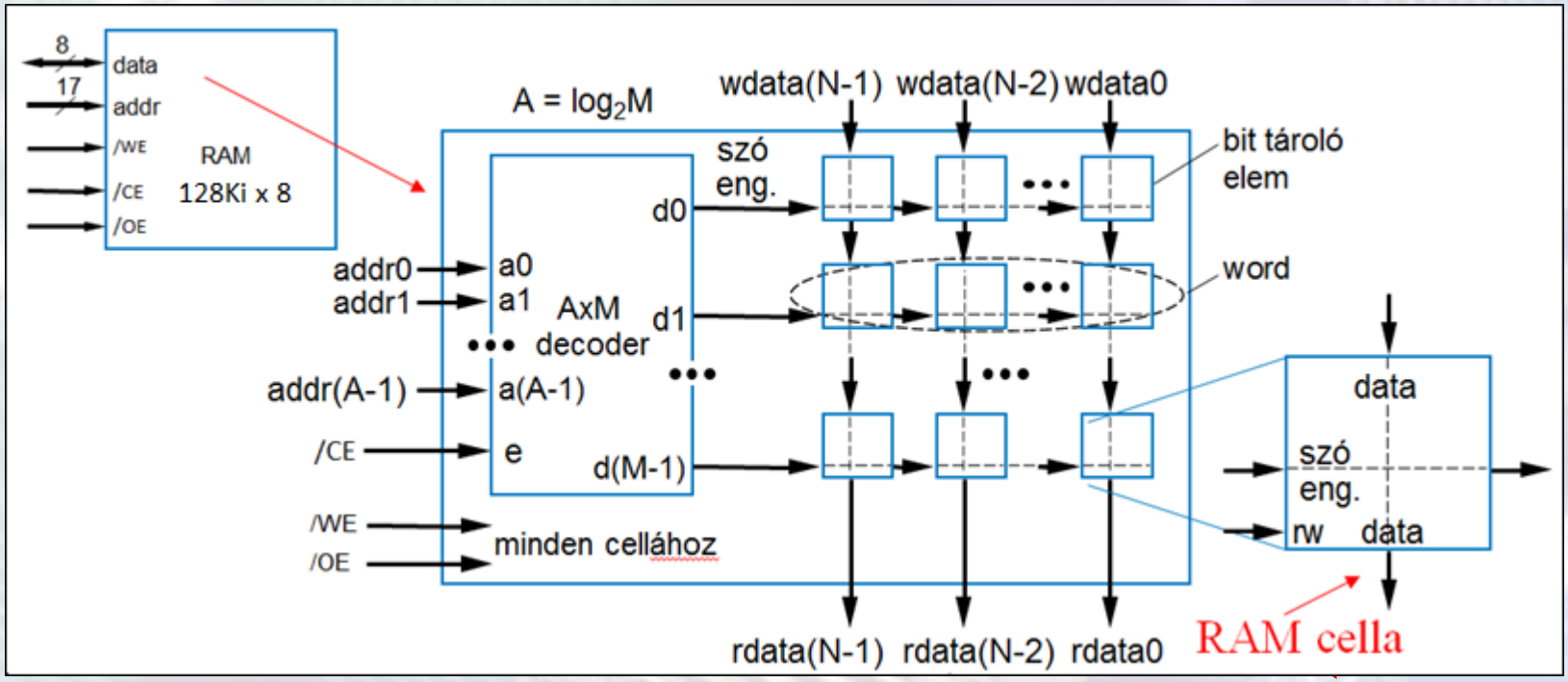
- **128ki x 8 SRAM blokkvázlata**
  - Látható az I/O adatbusz kétirányú meghajtása, (íráskor HiZ, letiltva)
  - A sok címvonal 2 csoportra van osztva, sor-oszlop címzés, 512x256x8 memória tömb
  - Látható a belső jelek meghajtása, címvonalak ponált/negált értéke
  - (Belső Clk. Gen, és Pre-Charge nem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



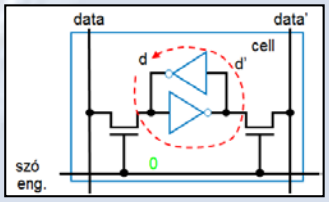


Memória

- **Működés:**
  - Hasonló a regiszter tömb működéséhez, csak…

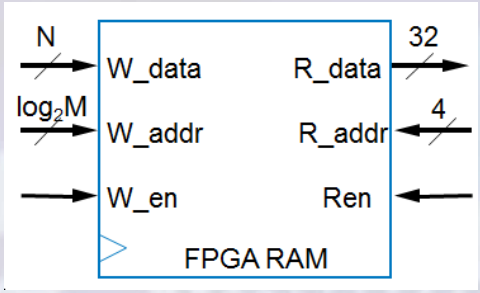


- Az ábra nem mutatja az I/O adatvonalakat, csak a belső struktúrát.



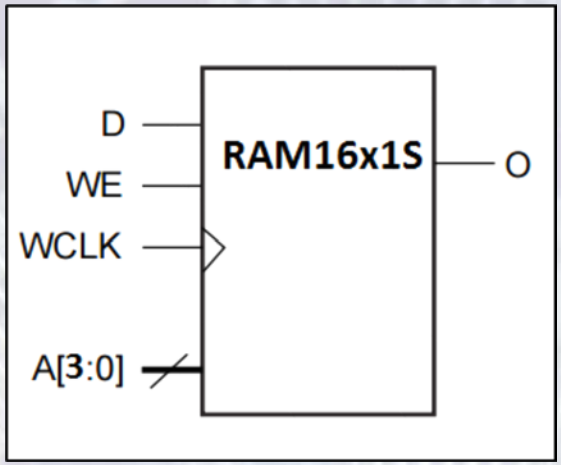
Memória

- **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



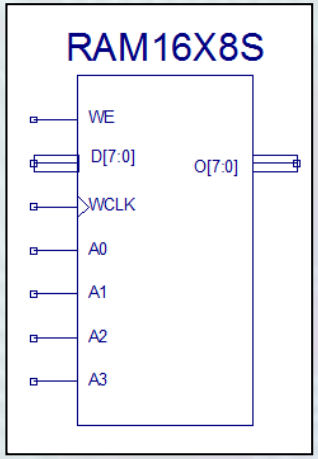
FPGA elosztott memória

- **Kisméretű, belső RAM tároló elem**
- **16x1 bites szelet, ebből tetszőleges méret felépíthető (de általában 2 Kibit alatt, azaz 256 bájt)**
- **1W1R, azaz 1 írás és 1 olvasás portja van**
  - Létezik dual portos verzióban is (1WR 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.

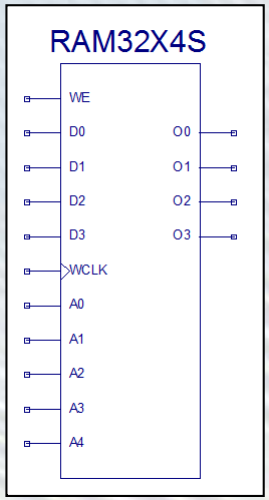


FPGA elosztott memória

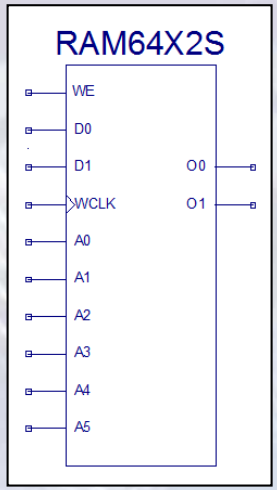
- **Jellemző kialakítások: adatok száma*adatszélesség**
- **Pl. 16x8 bit, 64x2bit, 32x4bit (mind 128 bit kapacitás)**
- **Belső kiegészítő áramkörök:**
  - RAM32X4S: 2:1 cím DEK, 4 bites 2:1 BUSZ MUX
  - RAM64X2S: 4:1 cím DEK, 2 bites 4:1 BUSZ MUX



8db 16x1 elem



2x4 db 16x1 elem

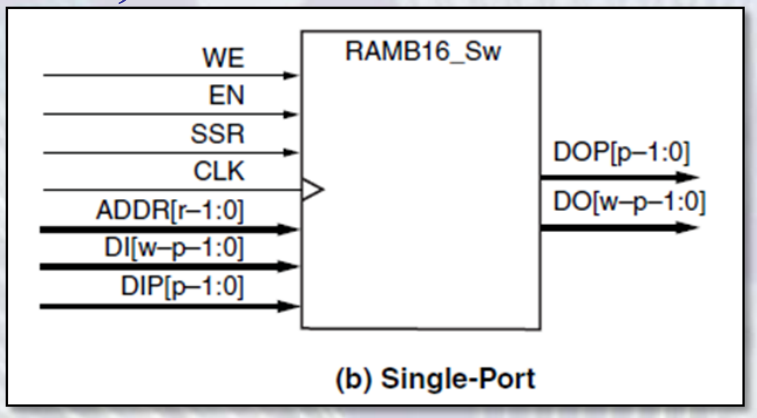


4x2 db 16x1 elem

- 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**



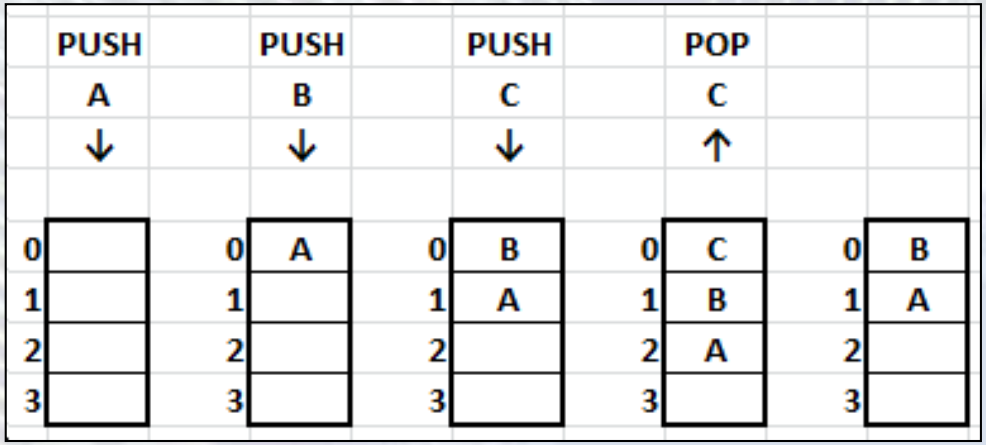
- **A két független port mindegyike írás/olvasás képes**
  - Írás/olvasás szinkron (CLK ↑ élre, ha WE = 1 ill. EN = 1)
  - Bővítési lehetőségek a normál memóriákhoz hasonlóan 2 dimenzióban, itt nagy segítség az eleve biztosított „területarány” választási lehetőség
- RÉSZLETEK A VERILOG HDL diákon

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 szoftver vezérelt realizáció is a számítógép rendszermemóriáját használva
  - Mi most a kisméretű, autonóm, hardver alapú megoldásokat mutatjuk be

Funkcionális egységek

- **Veremtár (STACK) LIFO (Last In First Out)**
- **A működés modellje a következő:**
  - Két művelet végezhető: PUSH és POP
  - A PUSH adatot helyez a verem tetejére, mialatt az addigi tartalmat egy szinttel lejjebb lépteti
  - A POP adatot vesz el a verem tetejéről (destruktív olvasási művelet, az adat elveszik a tárolóból) és a többi tartalmat egy szinttel feljebb lépteti.
  - Hibás műveletek:
    - PUSH teli STACK-en (adatvesztéshez vezet)
    - POP üres STACK-en (érvénytelen adat)
    - Előzetes ellenőrzés?



Funkcionális egységek

- **Veremtár (STACK), LIFO (Last In First Out)**
- **Realizációs lehetőségek:**
  - Multifunkciós regiszterekkel
  - Shiftregiszterrel
  - Memóriában speciális címaritmetikával
- **Minden szinten egy 4:1 MUX a regiszter bemenetén**
- **A MUX vezérlése és így a STACK funkciói a következők:**

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

- **Minden regiszter művelet természetesen közös felfutó órajel élre történik**

Funkcionális egységek

- **Veremtár (STACK), LIFO (Last In First Out)**
- **Multifunkciós regiszterekkel Verilog HDL leírás**

```
////////////////////////////////////////////////
// 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
```

Funkcionális egységek

- **Veremtár (STACK), LIFO (Last In First Out)**
- **Shiftregiszterrel, léptetés az adatméret szerint**

```
//////////////////////////////////////////////////////////////////////////////////
// 2. verzió
// A teljes adattárolót egyetlen kétirányú shiftregiszterként kezeljük, ami a
// shifteléskor az aktuális adatméretnek megfelelő lépésekben shiftel.
// Tehát a PUSH művelet mint SHIFT RIGHT 4, a POP művelet, mint SHIFT LEFT 4
// hajtódik végre. Az egyéb paraméterek, működési mód azonosak.
//////////////////////////////////////////////////////////////////////////////////

reg [15:0] shr;             // Belső regiszter, 4 bites, 4 szintű stack-hez

assign dout = shr[15:12];  // A stack kimeneti adata a négy MSB bit

always @ (posedge clk)                  // A shiftregiszter felfutó él vezérelt
   if (rst)   shr <= 16'b0;             // Aktív resetre törlődik
   else                                 // Ha a reset nem aktív, akkor
      begin
      if (push & ~pop)                  // érvényes push parancsra (ha nincs pop)
            shr <= {din, shr[15:4]};    // az adatbitek 4-gyel jobbra lépnek
                                        // és felül beírjuk a din értékét
      else                              // vagy ha
      if (~push & pop)                  // érvényes pop parancs (tehát nincs push)
            shr <= {shr[11:0], 4'b0};   // az adatbitek 4-gyel balra lépnek és
      end                               // alul beírunk 4 nullát.
```

Funkcionális egységek

- **Veremtár (STACK), LIFO (Last In First Out)**
- **Memóriával és címmutatóval ill. címaritmetikával**
  - A memória egy 16x4 bites elosztott memória
  - Íráskor az aktuális címre (az első következő üres helyre) ír, majd növeli a cím mutatót (auto post inkremens címzés)
  - Olvasáskor az utolsónak beírt adatra lép vissza, kiolvassa és rögzíti a cím mutató dekrementált értékét (auto pre dekremens címzés)

```
reg [3:0] mem [3:0];       // 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.
```

Funkcionális egységek

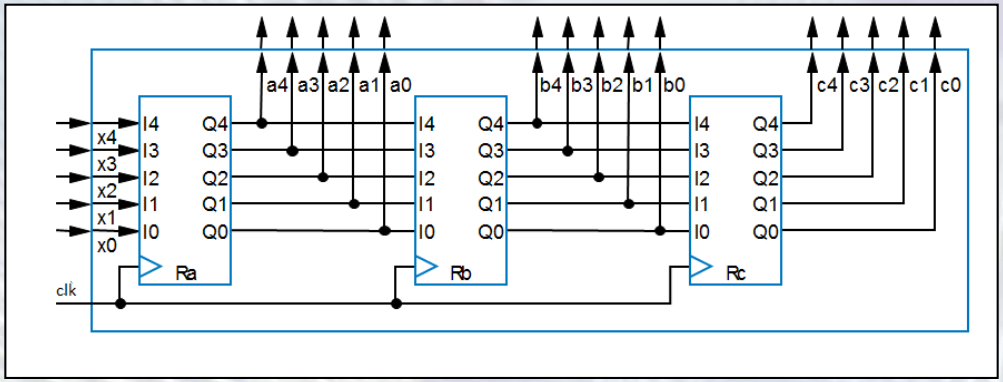
- **Veremtár (STACK), LIFO (Last In First Out)**
- **A címmutató számláló kódja**
  - A számláló bitszámot és a memória méretet átírva tetszőleges méretű LIFO realizálható

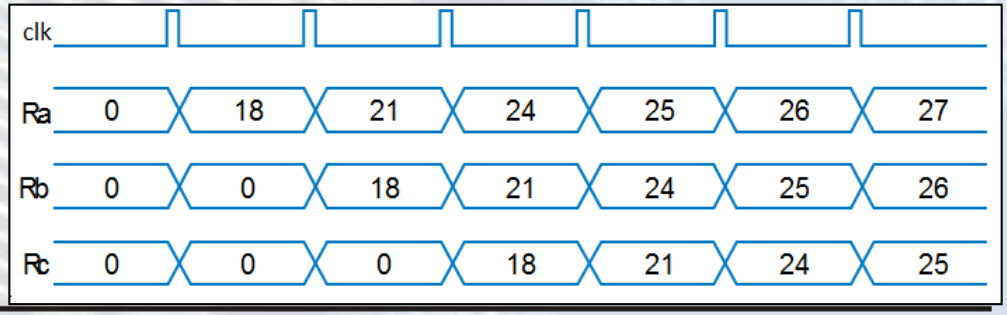
```
reg [3:0] cnt;
wire [3:0] address;

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
```

Funkcionális egységek

- **Sor (Queue), FIFO First-In-First-Out tároló**
  - A legegyszerűbb verziója a késleltető regiszter sor
  - A bemeneti adatokból minden órajelben mintát vesz, a legutolsó **n** mintát tárolja (x(t-1), x(t-2), ..x(t-n).
  - Beírás, kiolvasás folyamatos, periodikus, egyidejű
  - Mindig **n** adat van benne. Hasznos elem, de nem tud be-/kimeneti adat-sebességet kiegyen-líteni
- **A FIFO egyik fontos alkalmazása: rugalmas puffer**



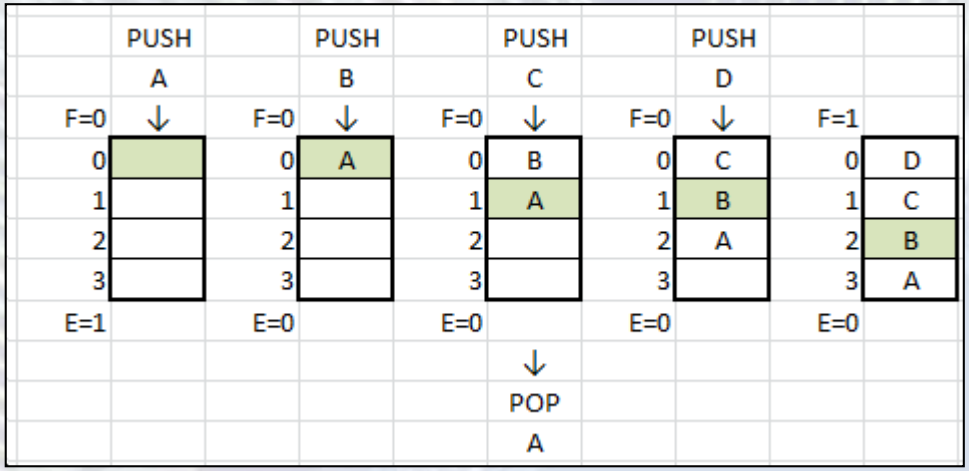


Funkcionális egységek

- **Sor (Queue), FIFO First-In-First-Out tároló**
- Műveletek: PUSH/POP, PUT/GET, WRITE/READ
  - Késleltető regisztersor alapú FIFO kis kiegészítéssel
  - A beírás legyen engedélyezhető: csak ha van új adat
  - A kiolvasás legyen címezhető: A legrégebben beírt adatra mutasson
  - Ez kis méretben megvalósítható egy engedélyezhető beírású, késleltető regiszter sorral
  - Pl. egy soros kommunikációs interfész adási vagy vételi pufferjéhez 8-16 bájt elegendő 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

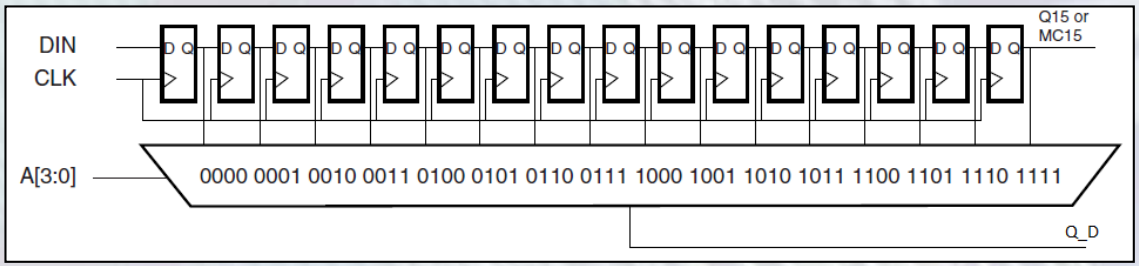
Funkcionális egységek

- **Sor (Queue), FIFO First-In-First-Out tároló**
  - Késleltető regisztersor alapú FIFO
    - Írási cím mutató nincs, mindig a sor elején ír
    - Beíráskor minden adat lép, az olvasási cím inkrementálódik
    - Olvasáskor az olvasási cím dekrementálódik
    - Kölcsönös írás/olvasás eredménye változatlan mutató érték
    - Kezdetben EMPTY = 1, FULL = 0.
    - Normál működés esetén mindkét jelzés 0.
    - Ha az olvasási cím eléri a végértéket, akkor FULL = 1



Funkcionális egységek

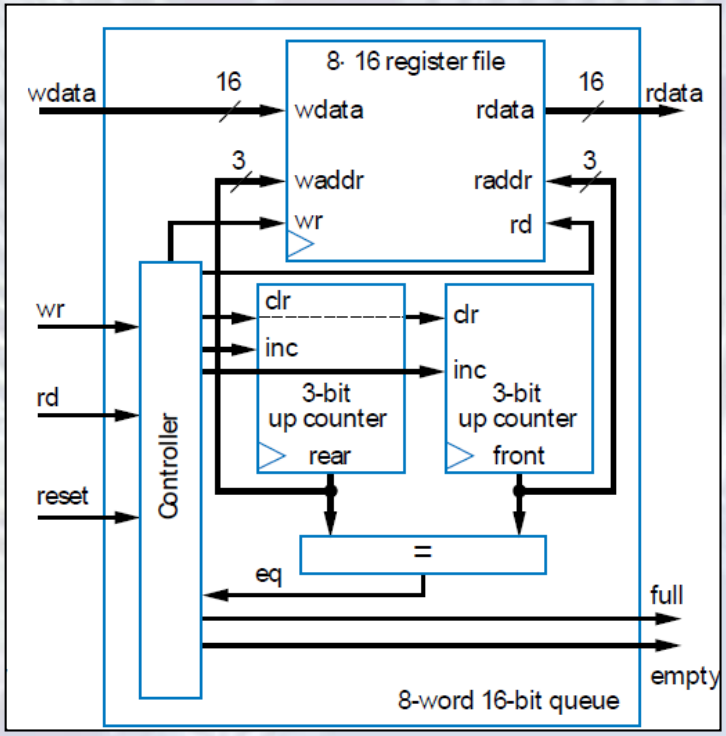
- **Sor (Queue), FIFO First-In-First-Out tároló**
  - Késleltető regisztersor alapú FIFO
    - Írási cím mutató nincs, mindig a sor elején ír
    - Olvasási cím mutató egy 4 bites BIN előre/hátra számláló
    - Beíráskor minden adat lép, az olvasási cím A[3:0] inkrem.



    - Olvasáskor az A[3:0] olvasási cím dekrementálódik
    - Kölcsönös írás/olvasás eredménye változatlan mutató érték
    - Kezdetben EMPTY = 1, FULL = 0.
    - Normál működés esetén mindkét jelzés 0.
    - Ha az olvasási cím eléri a végértéket, akkor FULL = 1

Funkcionális egységek

- **Sor (Queue), FIFO First-In-First-Out tároló**
  - Memória/regisztertömb alapú FIFO
  - Írási port: FIFO bemenet, Olvasási port: FIFO kimenet
  - Írási és olvasási cím mutatók: Moduló 8 bináris számlálók 012345670123… átfordulás → körpuffer
  - Vezérlés és státuszjelzés a számlálók értéke alapján
    - Ha írás után a egyenlők → FULL
    - Ha olvasás után egyenlők → EMPTY
  - A FIFO nagyon hasznos elem



Digitális technika
6. EA vége