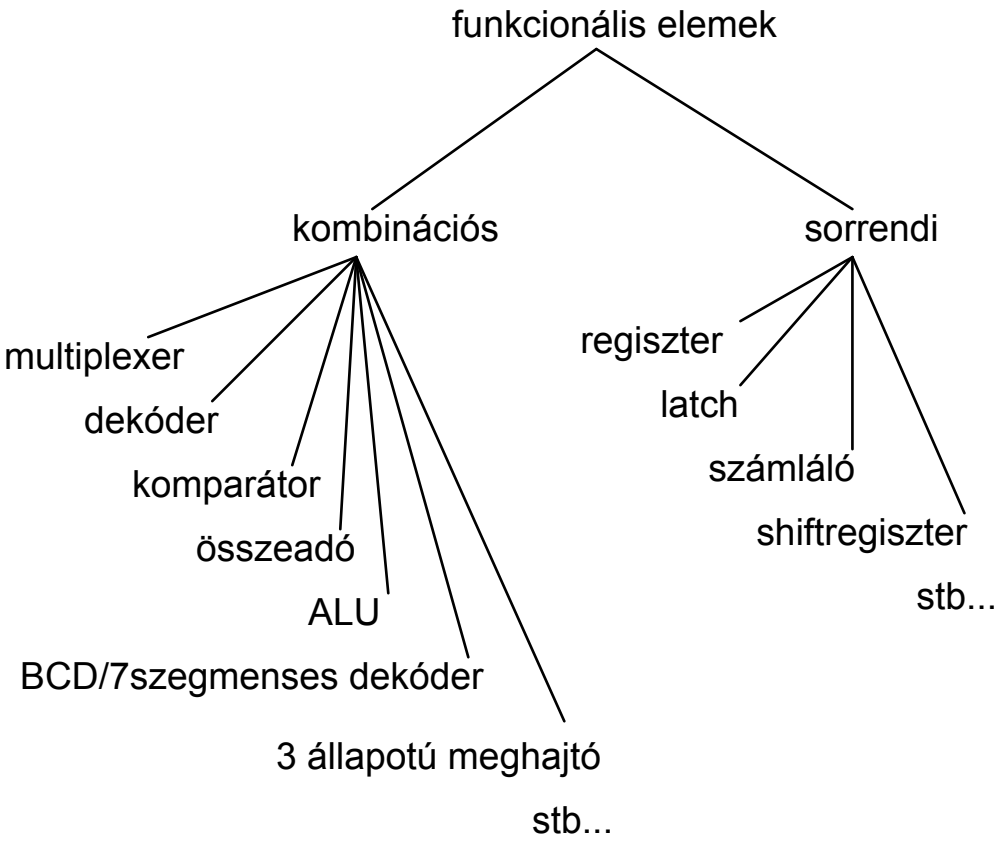
Funkcionális elemek

A funkcionális logikai elemek valamely az SSI (kapukból és flip-floppokból építkező) tervezési tapasztalatok során letisztult egyszerű funkciót valósítanak meg.



Kapuk, flip-flopok helyett ill. mellett a funkcionális elemeket használjuk a tervezés során.

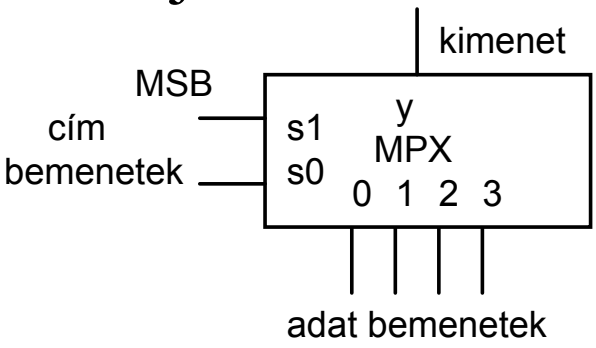
Multiplexer

Feladata: N bementből kiválasztja azt az egyet, amit a címbemenetei kijelölnek.

Elnevezése: N/1-es multiplexer

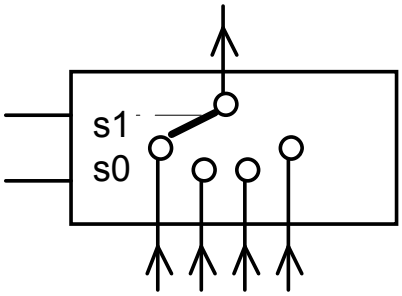
Pl: 4/1-es multiplexer

Rajzele:



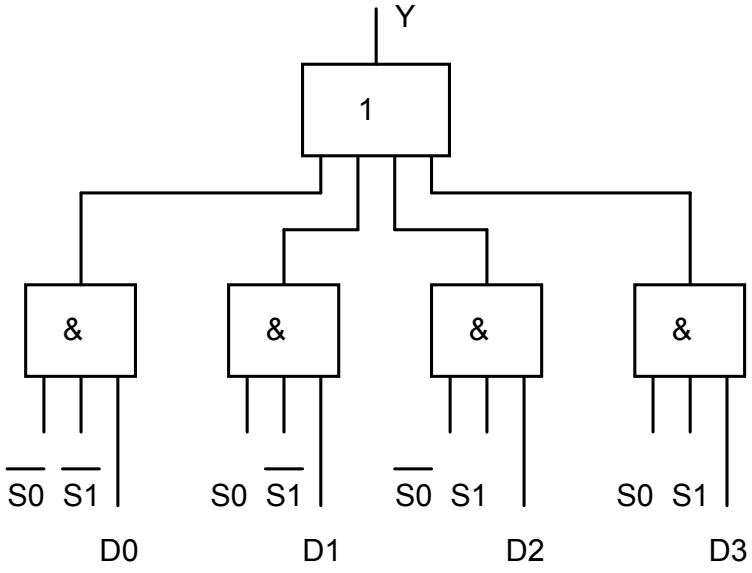
a)

Működése:



b)

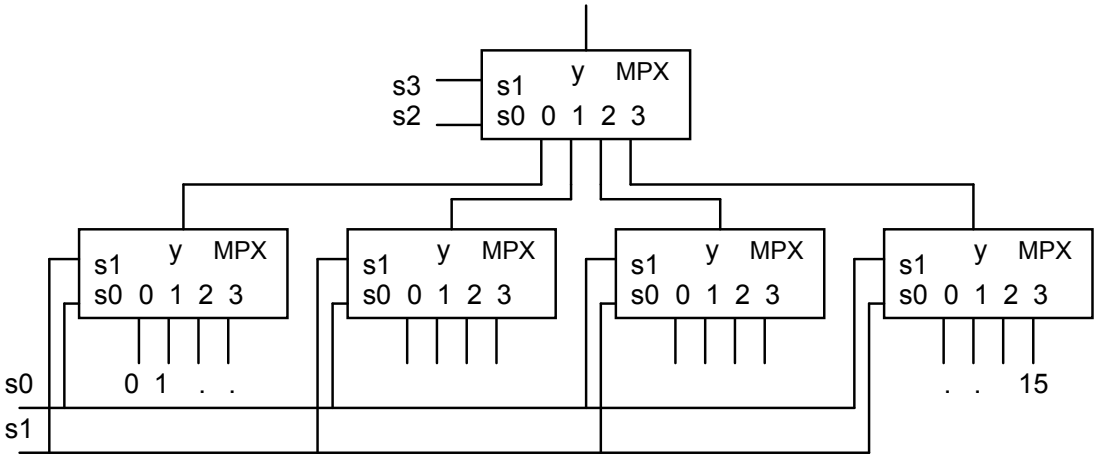
Belső felépítése:



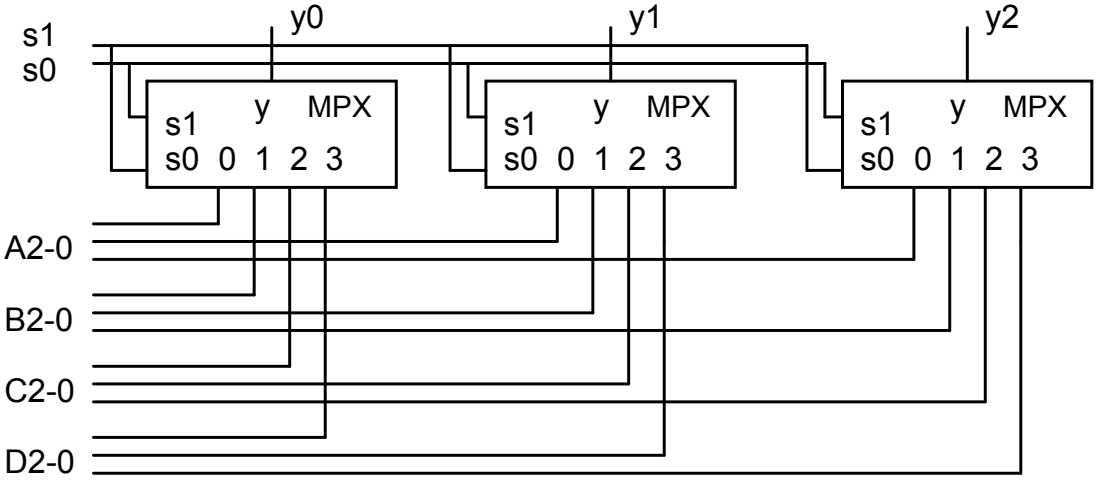
A 4/1-es multiplexer Verilog leírása

```
module mux4_1(I0, I1, I2, I3, S, Y);
    input I0;
    input I1;
    input I2;
    input I3;
    input [1:0] S;
    output Y;
    reg Y;

      always @(S,I0,I1,I2,I3)
      case (S)
          2'b00: Y = I0;
          2'b01: Y = I1;
          2'b10: Y = I2;
          2'b11: Y = I3;
      endcase
endmodule
```

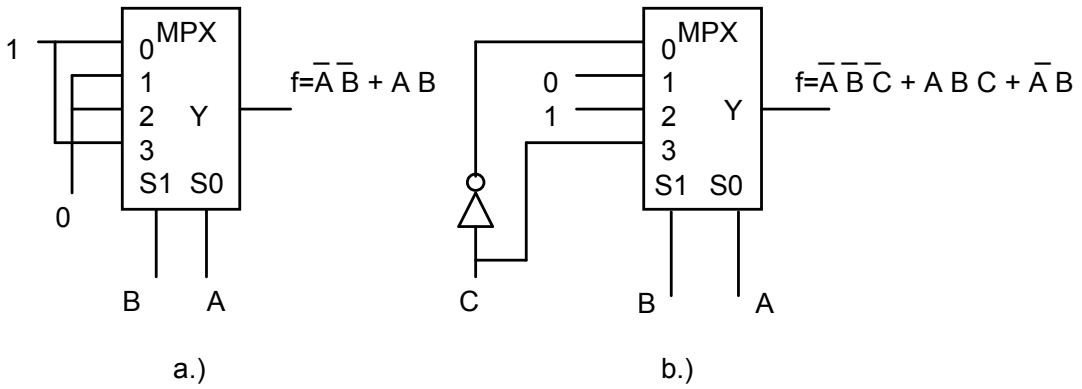
Csatorna számának növelése:



Csatorna szélességének bővítése:



Multiplexer mint univerzális KH:



Dekóder/demultiplexer

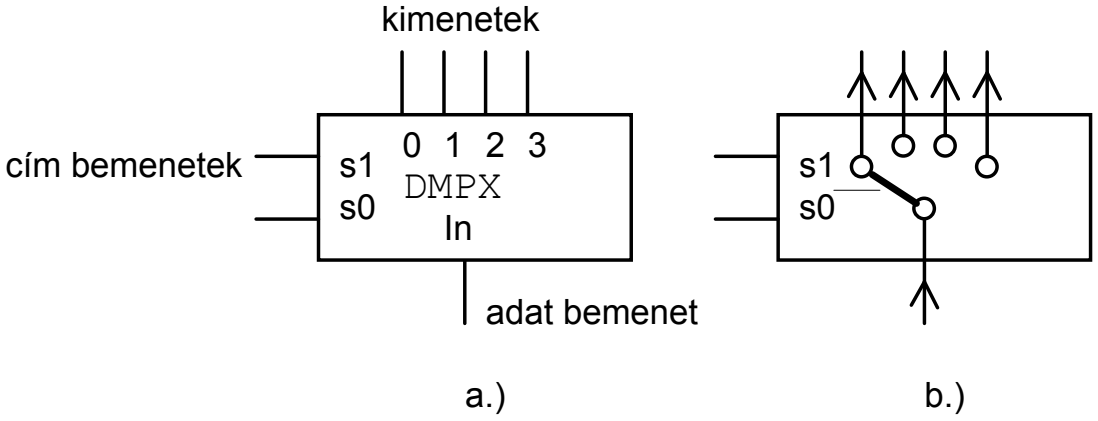
A demultiplexer feladata, egy bemeneti jelet kiadni a címbenetekkel kijelölt kimenetre.

A dekóder feladata a címbemenetekkel kijelölt kimenetének aktivizálása.

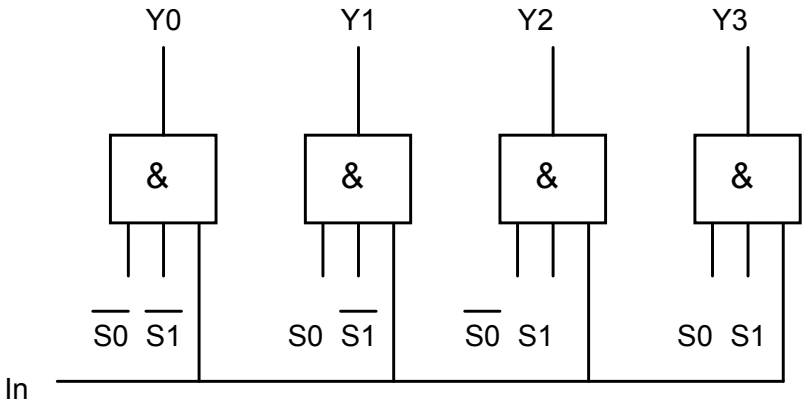
Inkább a dekóder funkciót használjuk.

Elnevezés: n/N-es dekóder/demultiplexer

A dekóder rajzjele: Működése:



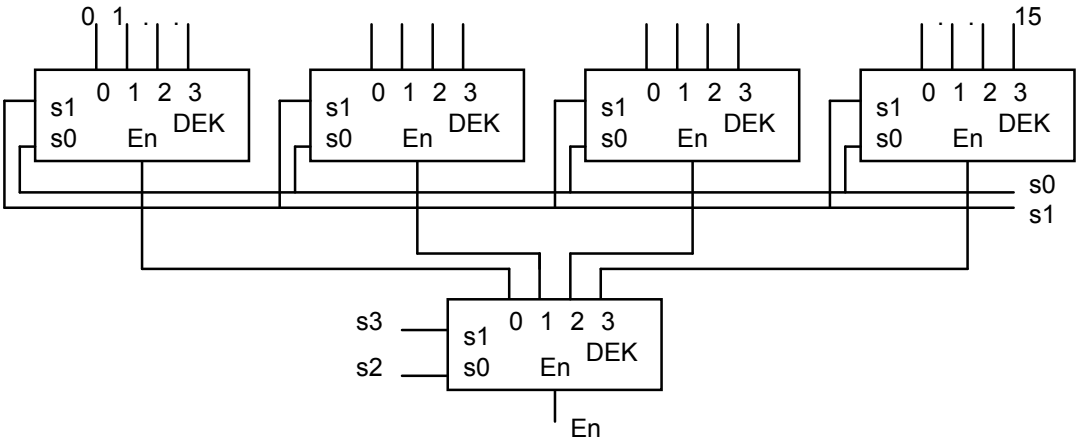
A dekóder belső felépítése:



Leírása hardver leíró nyellevvel, VERILOG-ban:

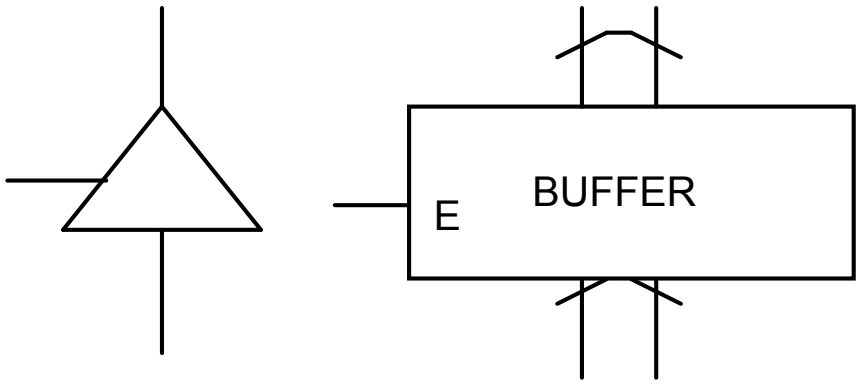
```
module dekoder2_4(E, S, Y);
    input E;
    input [1:0] S;
    output [3:0] Y;

    reg Y;

    always @(S,E)
      if (~E)
            Y = 4'b0000;
      else
            case (S)
                  2'b00 : Y = 4'b0001;
                  2'b01 : Y = 4'b0010;
                  2'b10 : Y = 4'b0100;
                  2'b11 : Y = 4'b1000;
            endcase
endmodule
```

A dekóder bővítése:



3 állapotú meghajtó

Funkciója, az engedélyező jellel a kimenete 3. állapotba hozható, ekkor lekapcsolódik a meghajtott vezetékről, ahová ezután más rákapcsolódhat.



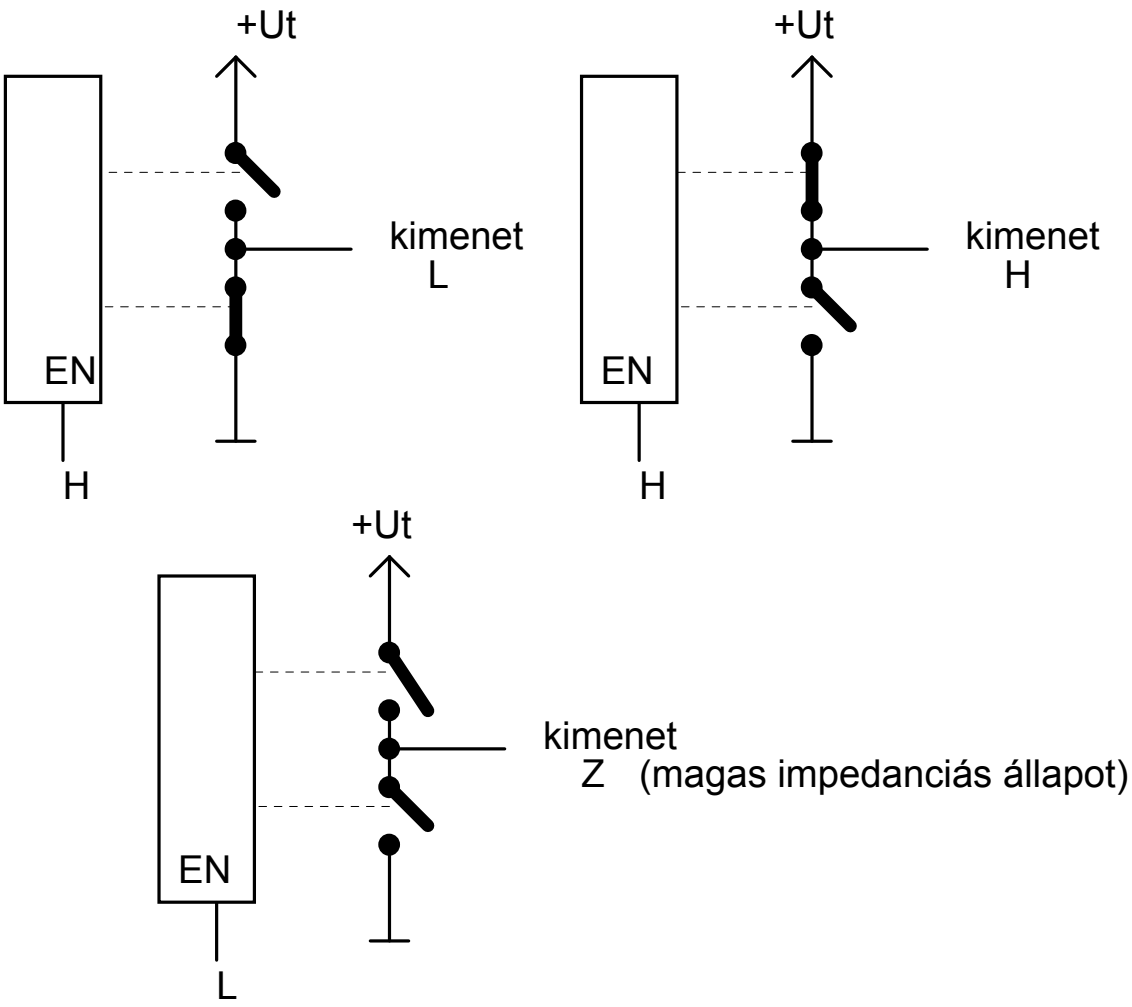
a.) 1 bites b.) több bites

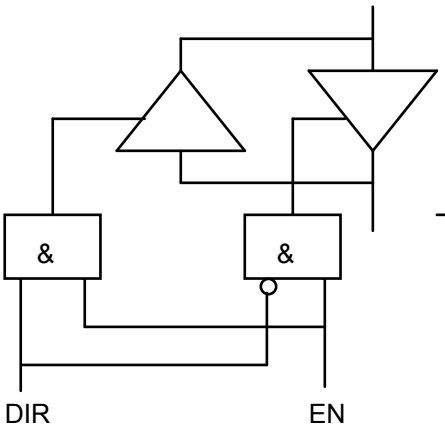
Leírása hardver leíró nyellevvel, VERILOG-ban:

```
module ThriStateBuff8(In, E, Y);
    input [7:0] In;
    input  E;
    output [7:0] Y;

    assign Y = E ? In : 8'bz;

endmodule
```

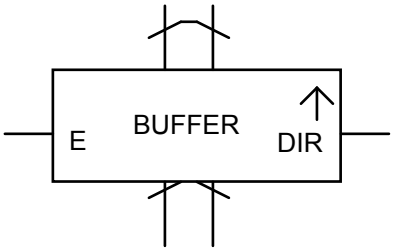
Három állapotú kimenet

Az L és H szintet aktív félvezető kapcsolók biztosítják, de a harmadik állapotban mindekttő kikapcsol.

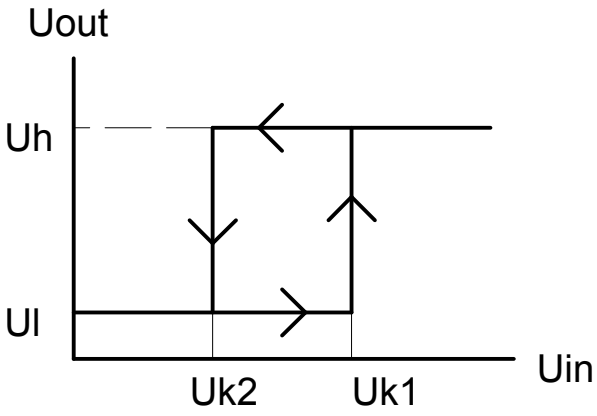




a.)

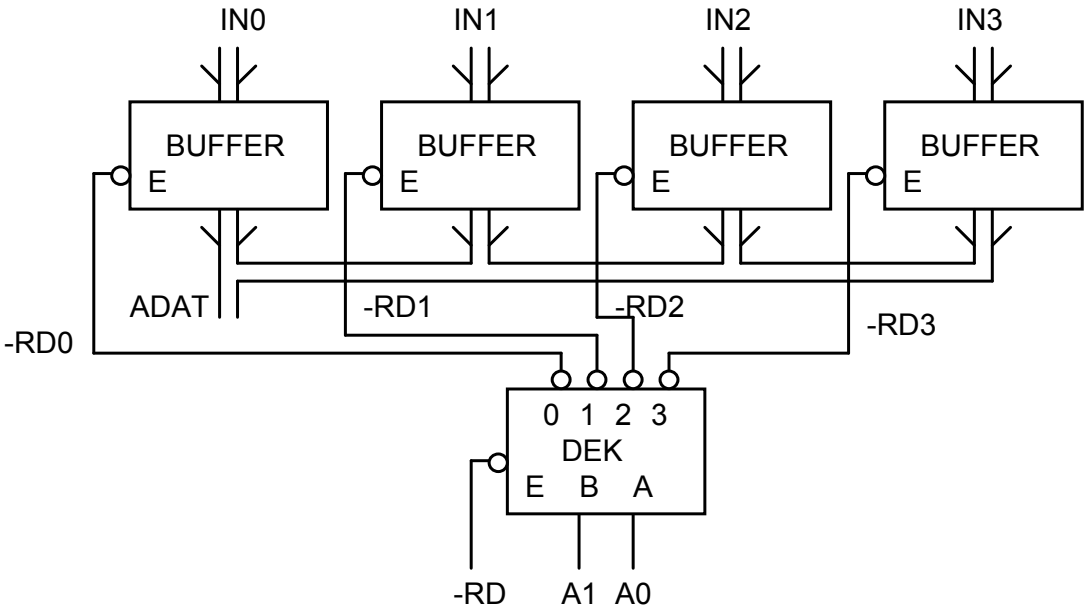


b.)

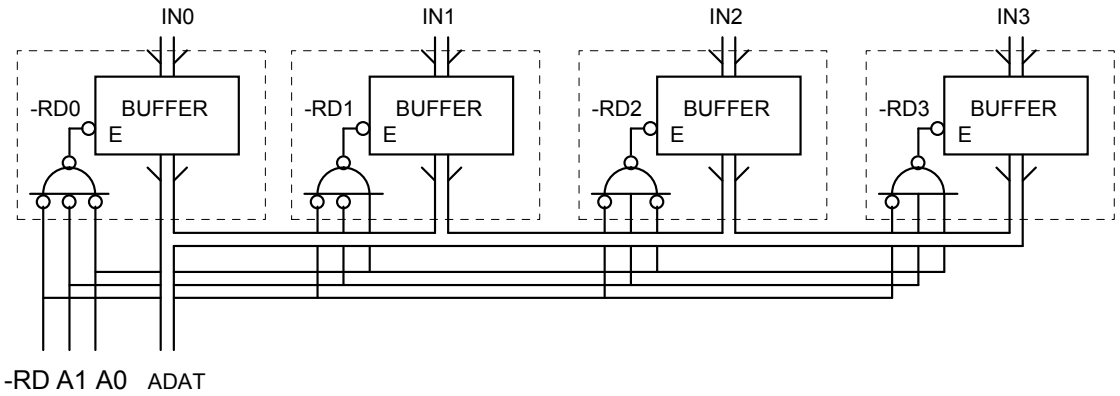


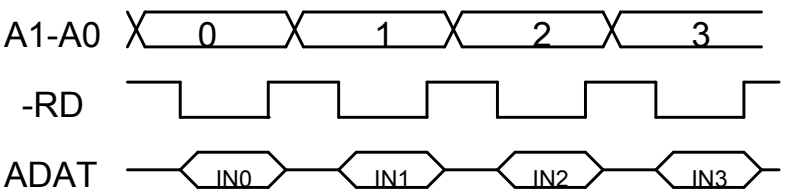
Multiplexer 3 állapotú meghajtóval

Centralizált felépítés:



Decentralizált felépítés:

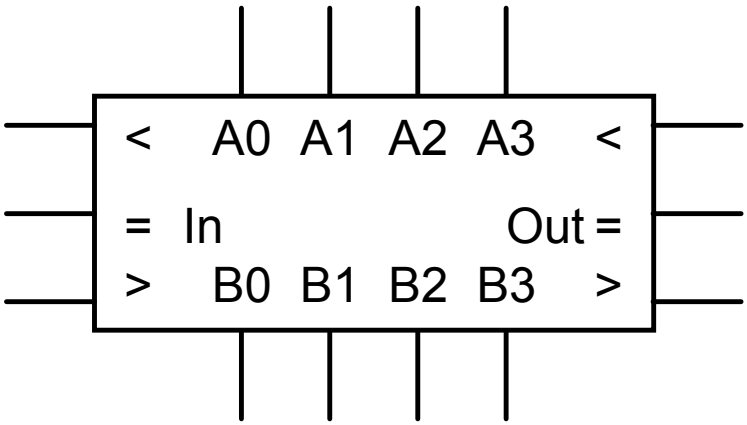




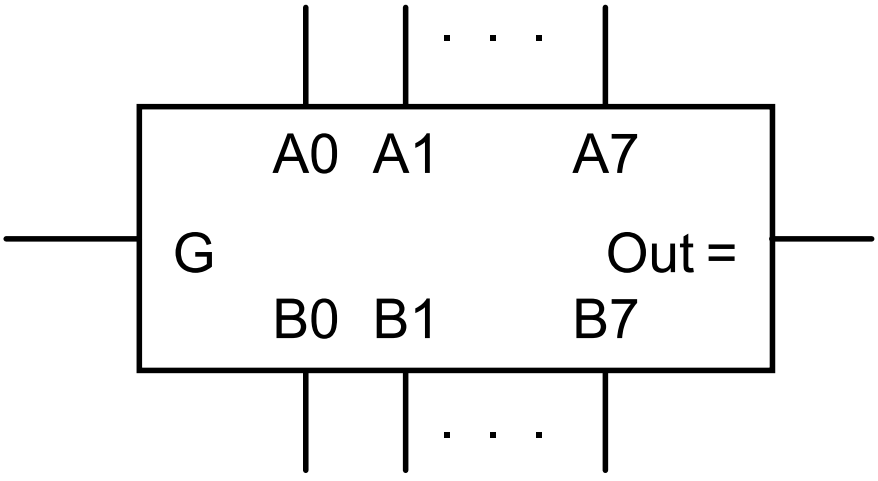
Komparátor

Funkciója a bementére adott számok összehasonlítása.

Ha azt is jelzi, hogy melyik a nagyobb, akkor magnitudo komparátor.



Ha csak egyenlőséget jelez, akkor egyenlőség komparátor.

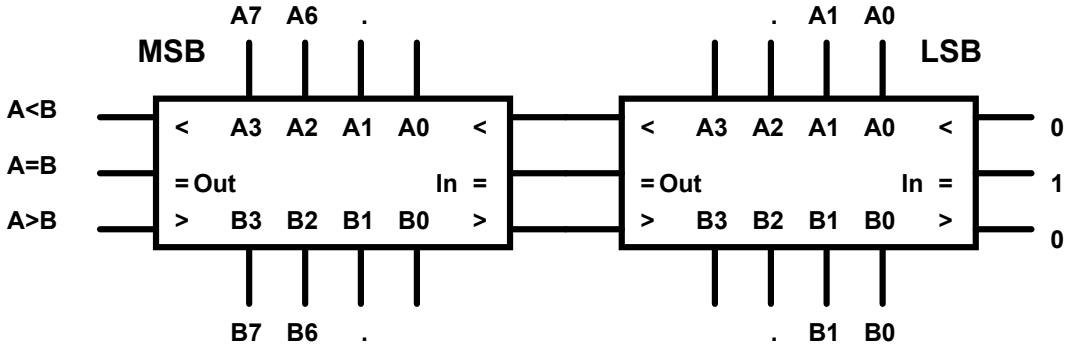


4 bites komparátor VERILOG leírása:

```
module mag_komp4(A, B, LTI, EQI,
GTI, LTO, EQO, GTO);
    input [3:0] A;
    input [3:0] B;
    input LTI;
    input EQI;
    input GTI;
    output LTO;
    output EQO;
    output GTO;

    assign EQO = A==B & EQI;
    assign LTO = A<B | A==B & LTI;
    assign GTO = A>B | A==B & GTI;
endmodule
```

Hagyományos kaszkádosítás

A legelső komparátor kaszkádosító bemeneteit úgy kell beállítani, mintha egy előző komparátor egyenlőséget jelezne.

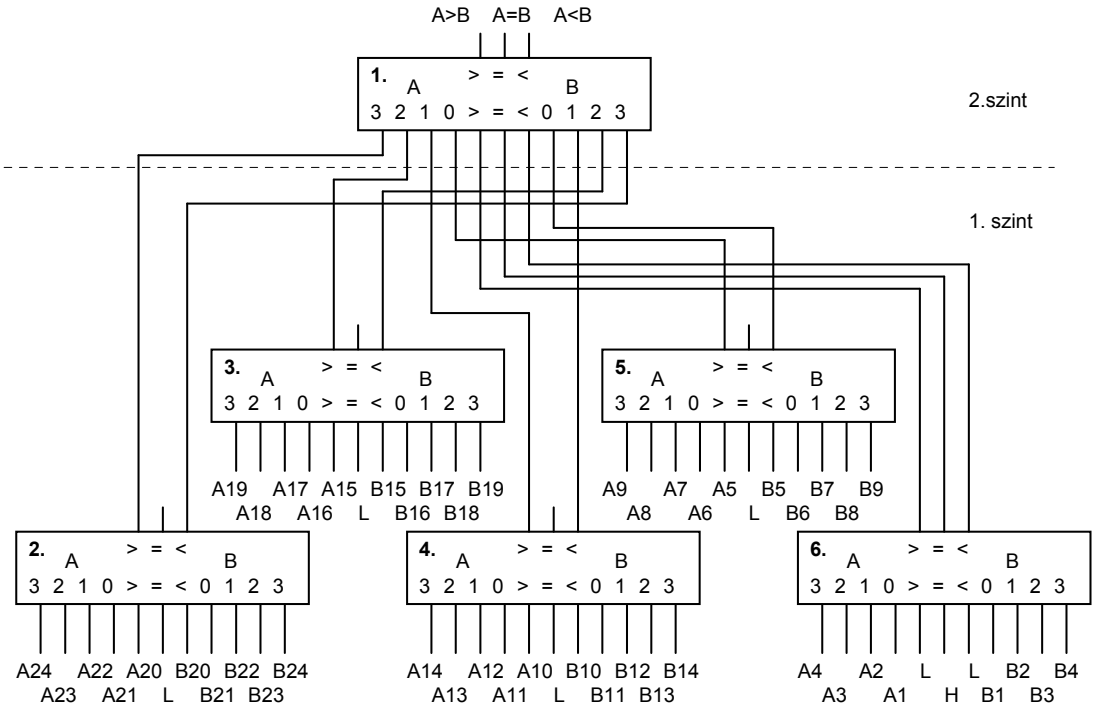


Komparátorral a 2-es komplemensben ábrázolt számokat az előjel bit invertálása után tudjuk összehasonlítani (offszet kóddá transzformálás).

Gyorsított kaszkádosítás

A hagyományos kaszkádosítás hátránya, hogy a kaszkádosított egységek késletetési ideje összeadódik.

Itt 4 bites komparátorokkal gyors 24 bites komparátor.
A kapcsolás olyan komparátort tételez fel, amelynek < és > bementére egyszerre ugyanazt kapcsolva, annak < és > kimenete egyforma logikai értéket ad, ha A=B.



Összeadó

Funkciója:

Az összeadó két n bites (előjel nélküli abszolútértékes vagy 2-es komplemens ábrázolású) bináris szám összeadása.

0+0=0 C=0, 0+1=1+0=1 C=0,
1+1=0 C=1
Pl: **1001**
+1011

10100

Paramétere:

- szószélesség (hány bites): n

Bemenetei:

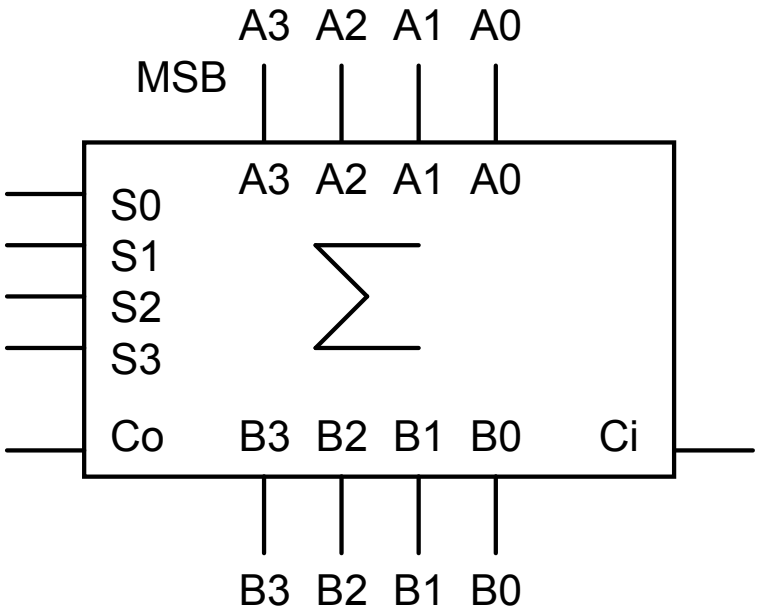
- összeadandó számok (A, B) 2 x n bit

Ci átvitel bemenet

Kimenetei:

- eredmény S (SUM) n bit
- Co átvitel

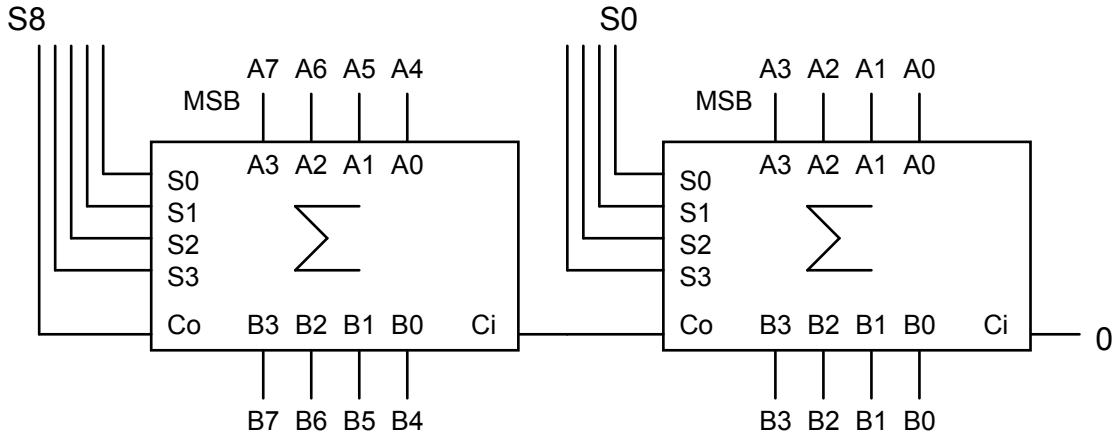
Rajzjele:



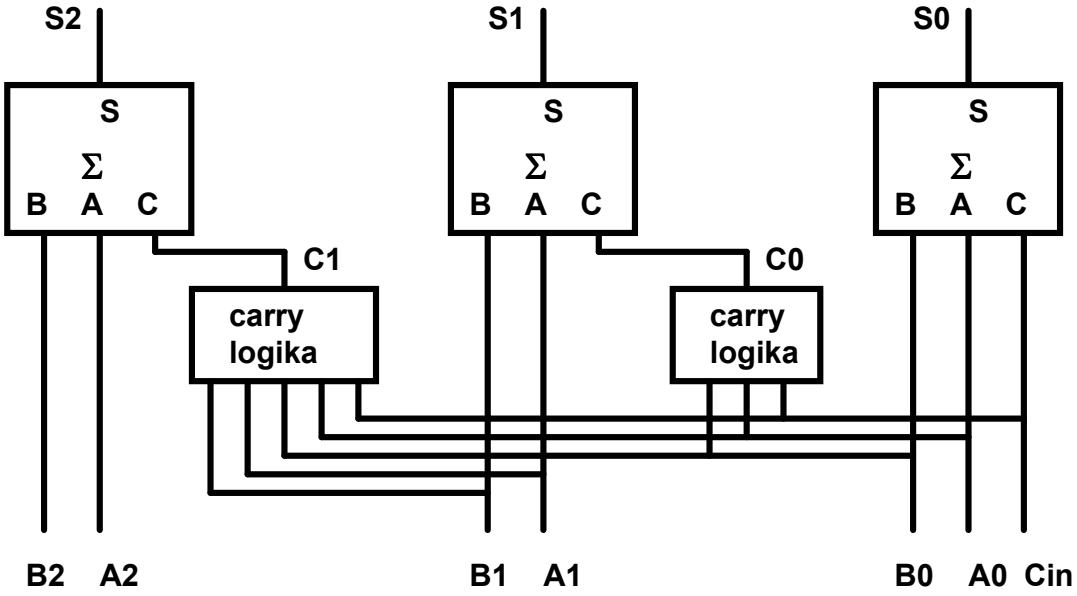
4 bites összeadó VERILOG leírása:

```
module add4(A, B, CI, S, CO);
    input [3:0] A;
    input [3:0] B;
    input CI;
    output [3:0] S;
    output CO;

    assign {CO,S} = A + B + CI;
endmodule
```

Két 4 bites összeadó kaszkádosítása
A legelső összeadó Cin bemenetére 0-át kell kötni.

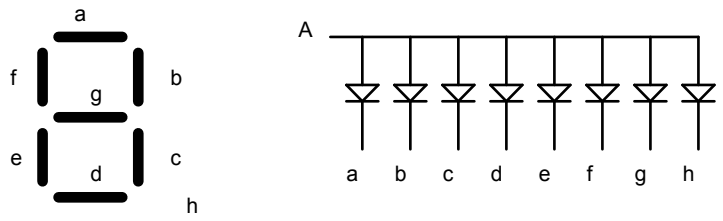


Sok bit esetén az átvitel soros terjedése nagy késleltetést okoz.
Elkerülése: átvitel gyorsítással

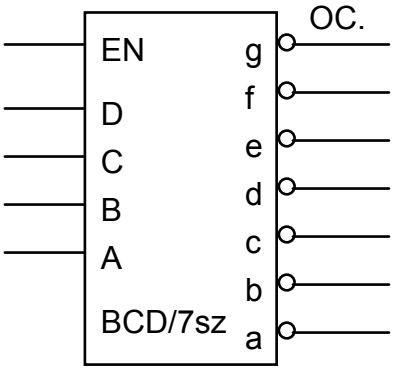


BCD/7szegmenses dekóder

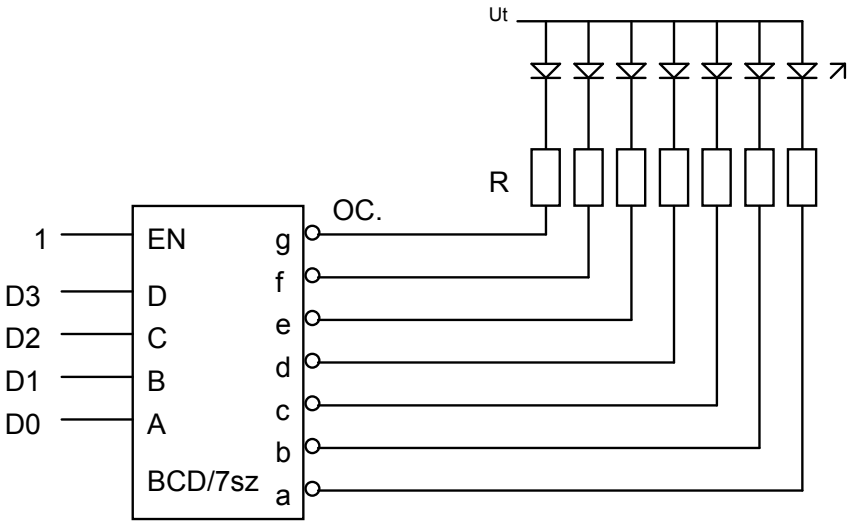
A meghajtandó LED kijelző:



A dekóder rajzjele:



Alkalmazása:



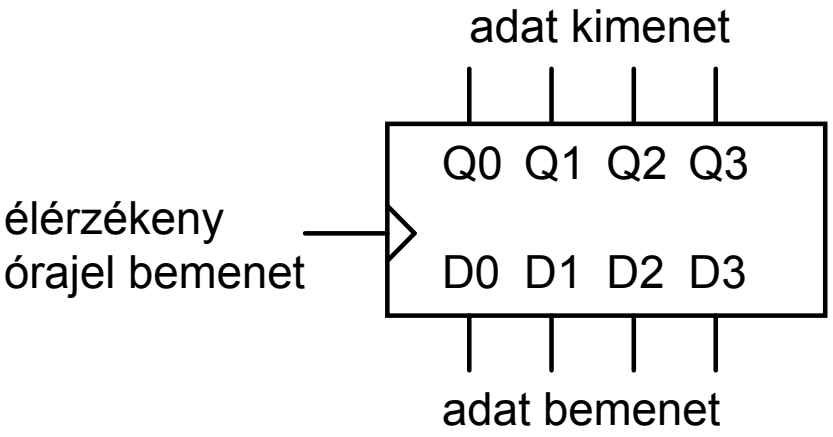
7 szegmenses dekóder Verilog leírása

```
module Seg7Dek(HEXin, Seg7Ou);
    input [3:0] HEXin;
    output [6:0] Seg7Ou;
         reg Seg7Ou;
// 7-segment encoding
//      0
//     ---
//  5 |   | 1
//     --- <--6
//  4 |   | 2
//     ---
//      3
   always @(HEXin)
      case (HEXin)
         4'b0001 : Seg7Ou = 7'b1111001;   // 1
         4'b0010 : Seg7Ou = 7'b0100100;   // 2
         4'b0011 : Seg7Ou = 7'b0110000;   // 3
         4'b0100 : Seg7Ou = 7'b0011001;   // 4
         4'b0101 : Seg7Ou = 7'b0010010;   // 5
         4'b0110 : Seg7Ou = 7'b0000010;   // 6
         4'b0111 : Seg7Ou = 7'b1111000;   // 7
         4'b1000 : Seg7Ou = 7'b0000000;   // 8
         4'b1001 : Seg7Ou = 7'b0010000;   // 9
         4'b1010 : Seg7Ou = 7'b0001000;   // A
         4'b1011 : Seg7Ou = 7'b0000011;   // b
         4'b1100 : Seg7Ou = 7'b1000110;   // C
         4'b1101 : Seg7Ou = 7'b0100001;   // d
         4'b1110 : Seg7Ou = 7'b0000110;   // E
         4'b1111 : Seg7Ou = 7'b0001110;   // F
         default : Seg7Ou = 7'b1000000;   // 0
      endcase
endmodule
```

Sorrendi funkcionális elemek

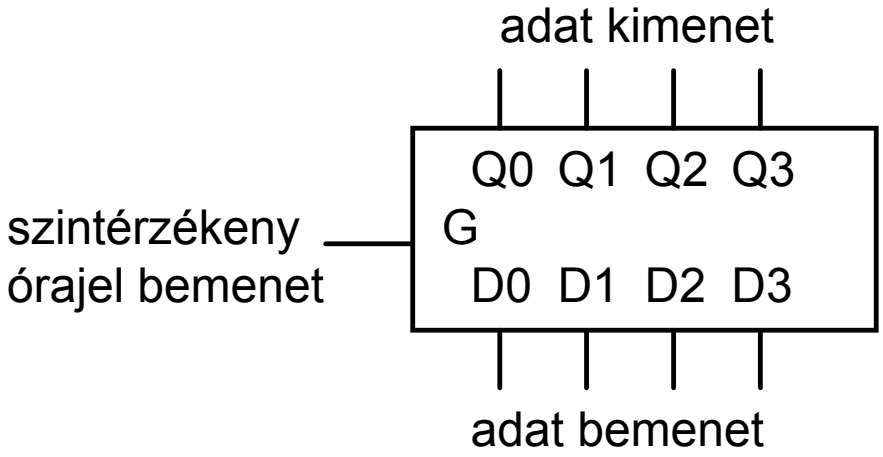
(tároló tulajdonságú elemek)

Regiszter (közös órajelű D flip-flopok)



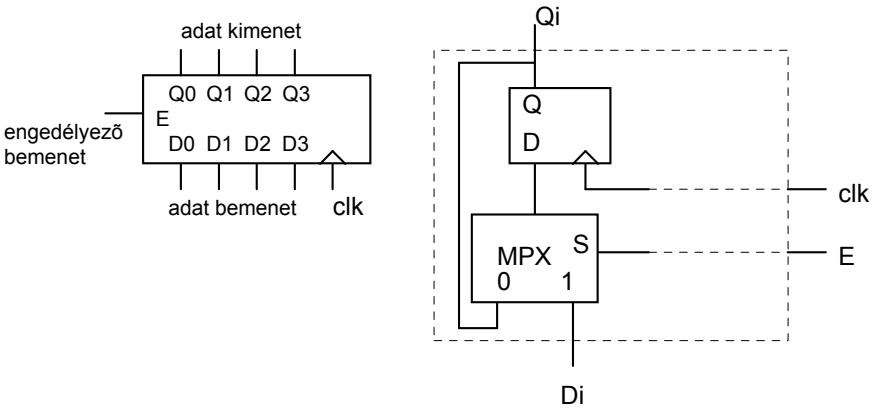

```
module D_REG(clk, D, Q);
    input clk;
    input [3:0] D;
    output [3:0] Q;
    reg Q;

always @(posedge clk)
        Q <= D;
endmodule
```

Latch (közös G-jű D-G flip-flopok)

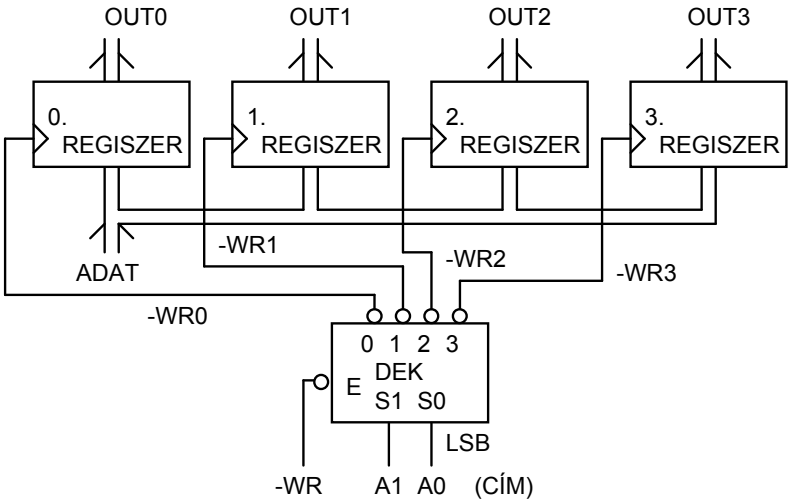


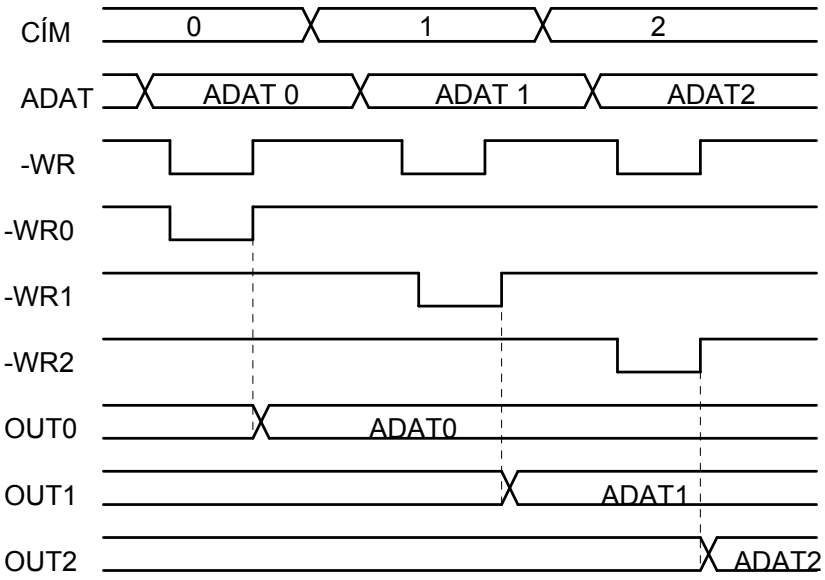

```
module D_LATCH(G,D, Q);
    input G;
    input [3:0] D;
    output [3:0] Q;
    reg Q;

always @((G or D))
     if (G)     Q = D;
endmodule
```

Engedélyezhető regiszter

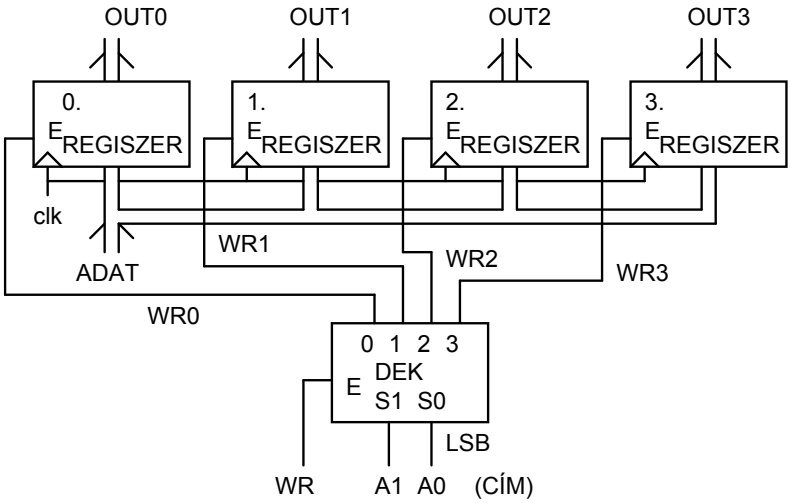


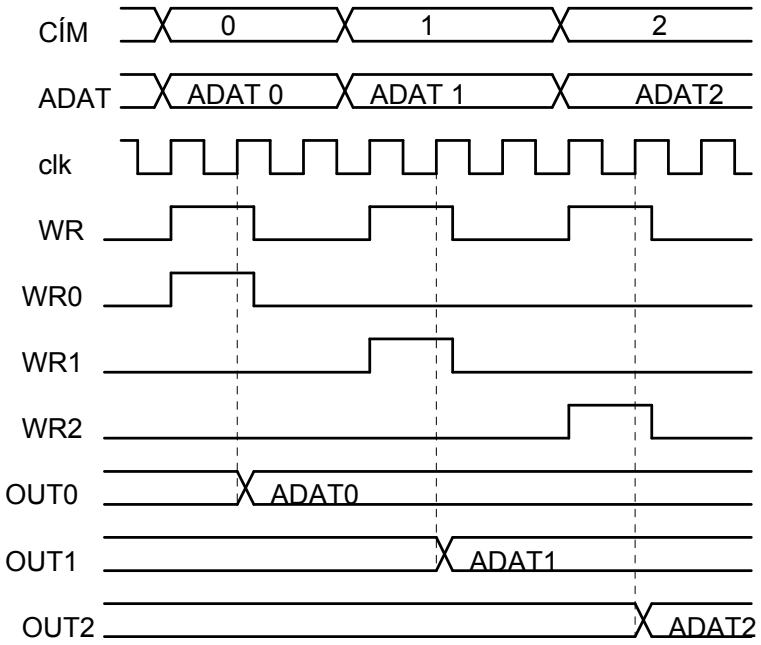
Regiszterek beírása buszról (centralizált felépítés):



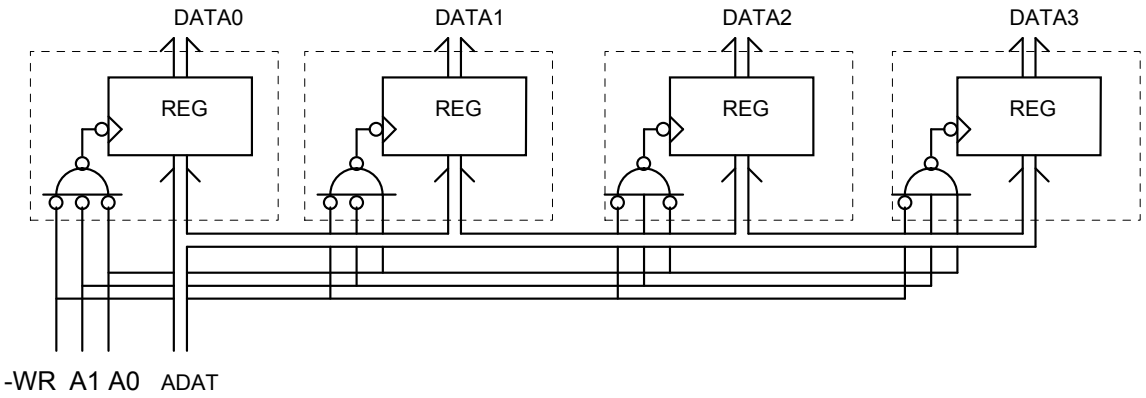


Engedélyezhető regiszterek beírása buszról (centralizált felépítés):





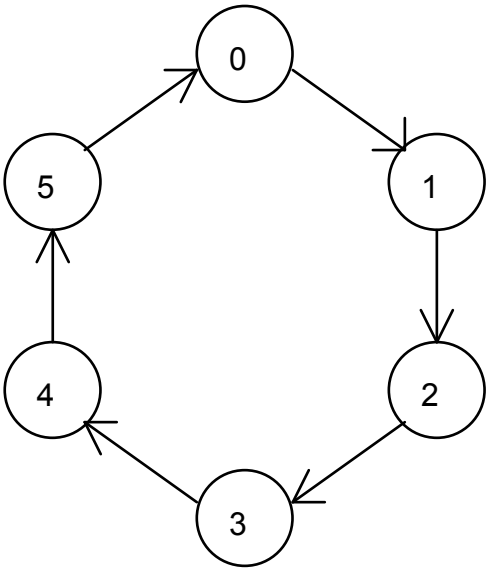
Decentralizált felépítés:



Számlálók

A számlálók (számláló üzemmódjára jellemző) állapotgráfja gyűrű alakú.

Pl. 6-os számláló állapotgráfja:

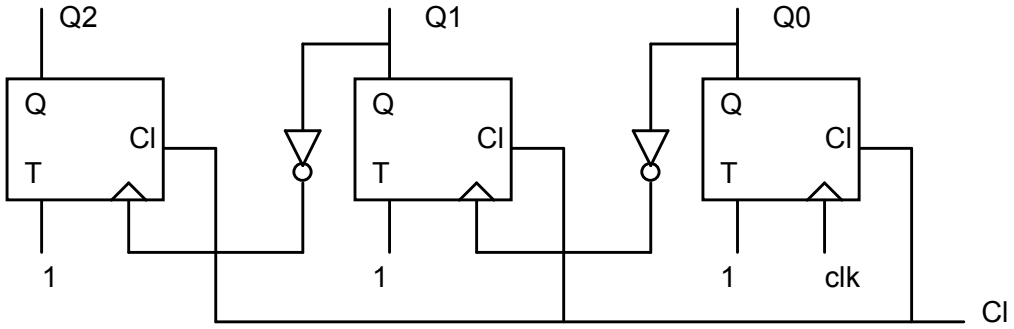


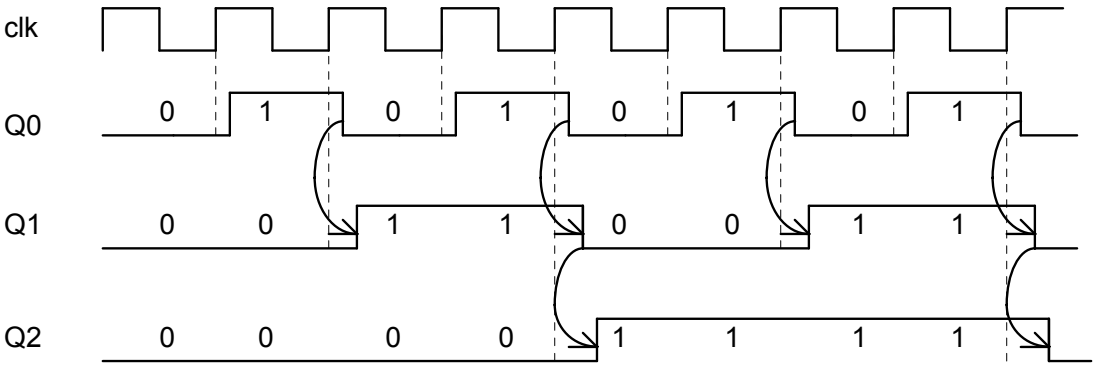
A számláló az órajeleit számolja. A ***modulus***a, a ciklus hossza.

Modulus alapján: bináris, decimális, egyéb (12-es, 6-os stb.)

Aszinkron számláló

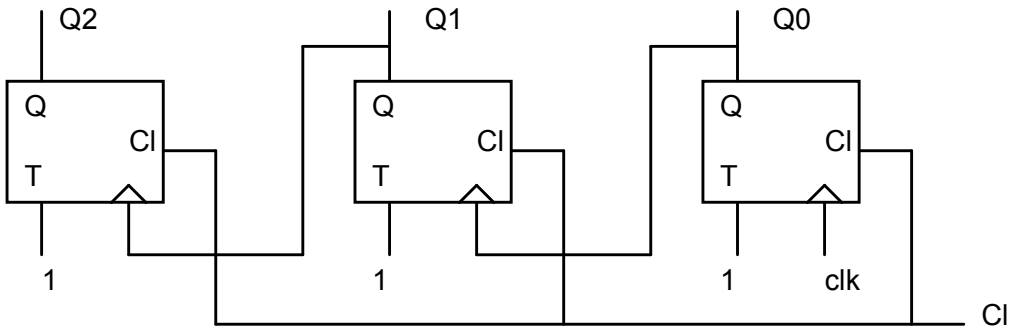
Aszinkron bináris ***felfele*** számláló, aszinkron törléssel:

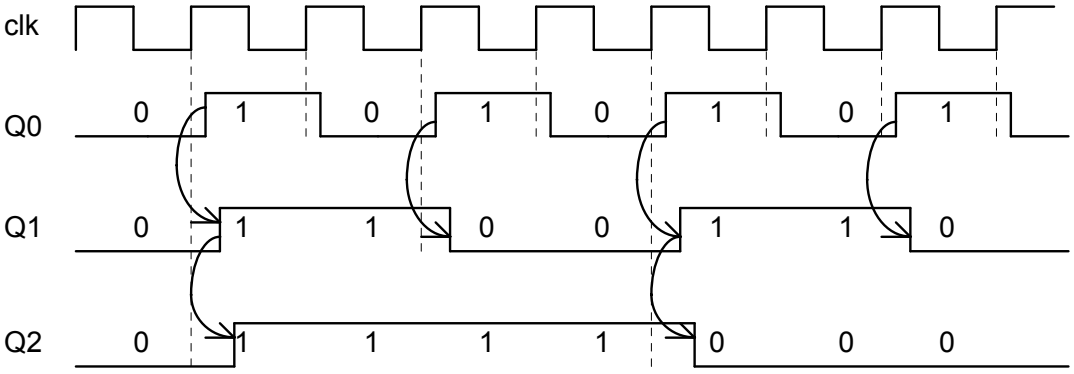




Az egyes kimenetek egyre jobban késnek az órajelhez képest.

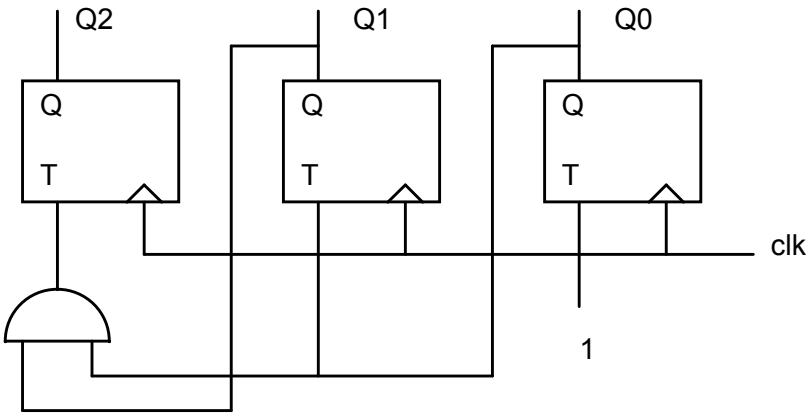
Aszinkron bináris ***lefele*** számláló, aszinkron törléssel:

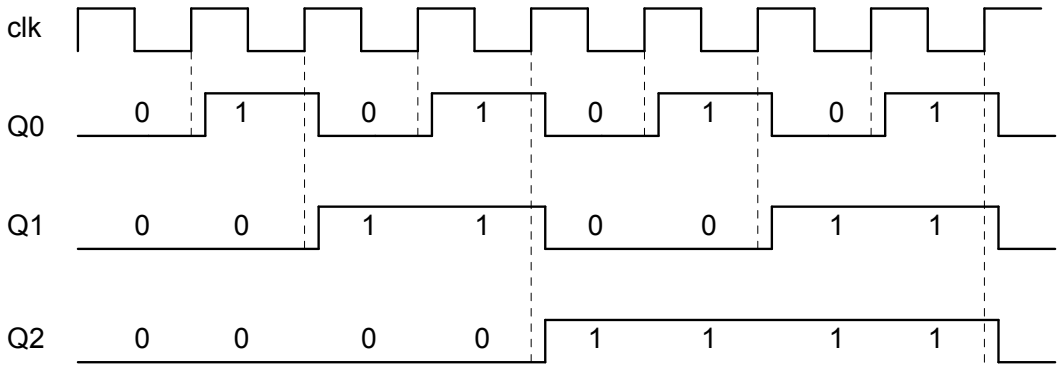




Szinkron számláló

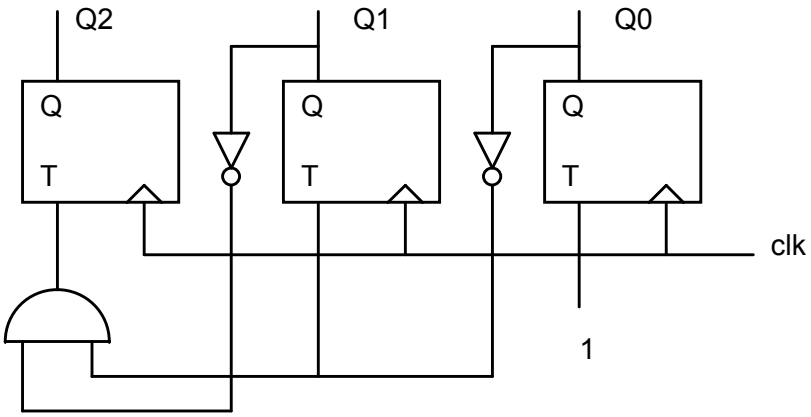
Szinkron bináris ***felfele*** számláló belső felépítése:



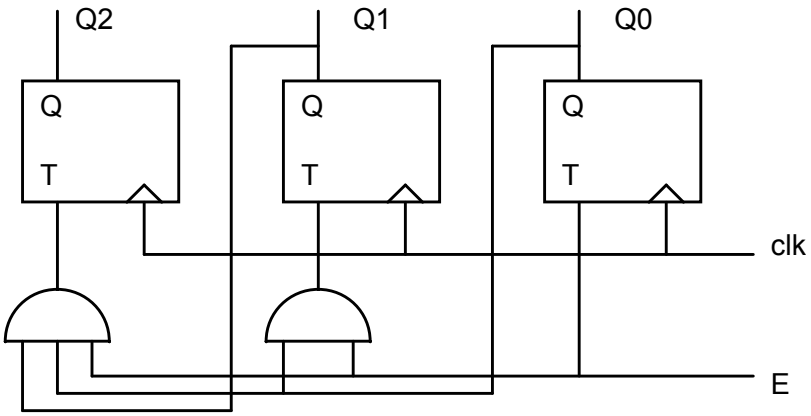


A kimenetek egyszerre váltanak, késleltetésük kicsi az órajelhez képest.

Szinkron bináris ***lefele*** számláló belső felépítése:

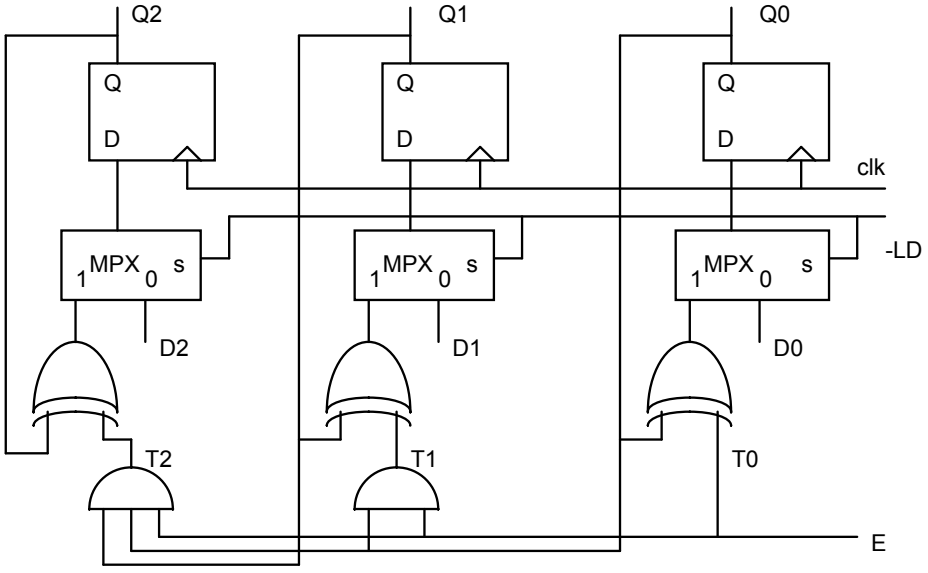


Szinkron engedélyezhető bináris felfele számláló belső felépítése:



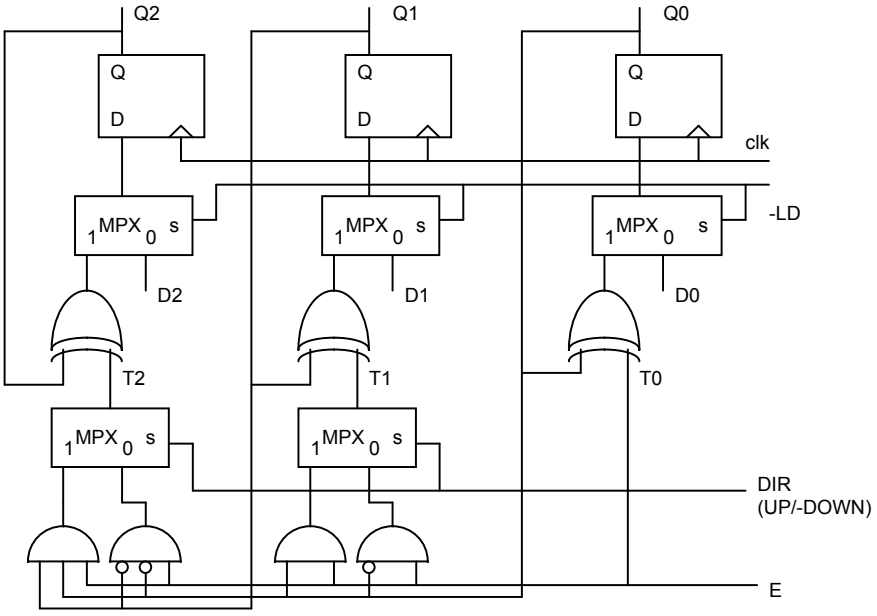

```
module UPCOUNTER(clk, E,  Q);
    input clk;
    input E;
    output [3:0] Q;
    reg Q;

always @(posedge clk)
        if (E)  Q <= Q+1;
endmodule
```

Szinkron bináris, engedélyezhető és tölthető felfele számláló funkcionális felépítése:




```
module UPCOUNTERLD(clk, E, LD, D, Q);
    input clk;
     input E;
    input LD;
    input [3:0] D;
    output [3:0] Q;
    reg Q;

always @(posedge clk)
if(LD) Q <= D;
else if (E) Q <= Q+1;
endmodule
```

Szinkron bináris, engedélyezhető és tölthető fel-le számláló funkcionális felépítése:




```
module UDCOUNTERLD(clk, E, LD, DIR, D, Q);
input clk;
input E;
input LD;
input DIR;
input [3:0] D;
output [3:0] Q;
reg Q;

always @(posedge clk)
if (LD) Q <= D;
else   if (E)
       begin
            if (DIR) Q <= Q+1;
            else     Q <= Q-1;
       end
endmodule
```

Számlálók vezérlő jelei lehetnek:
CL: törlés
LD: betöltés
En: engedélyezés (csak szinkron esetben)
DIR: számlálási irány

Számlálók kimeneti jelei lehetnek:
CY(carry): felfele számláló végállapota
Bináris számlálónál
CY = Q0.Q1.Q2….

BO(borrow): lefele számláló végállapota
BO = /Q0./Q1./Q2….

Max/min: fel-le számláló végállapotai
Bináris számlálónál
M/m=DIR.Q0.Q1.Q2..+/DIR./Q0./Q1./Q2..

RCY: ripple carry
RCY = E.CY

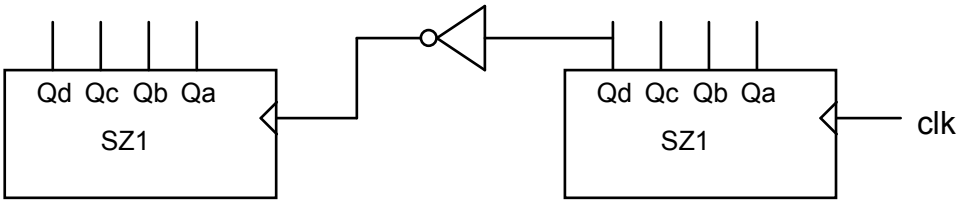
RCO: ripple clock
RCO = /(E.CY./CLK)

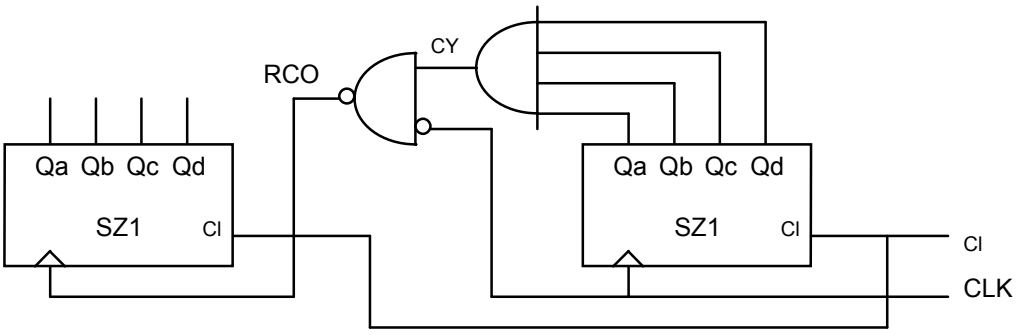
Számlálók kaszkádosítása *(modulus növelés)*

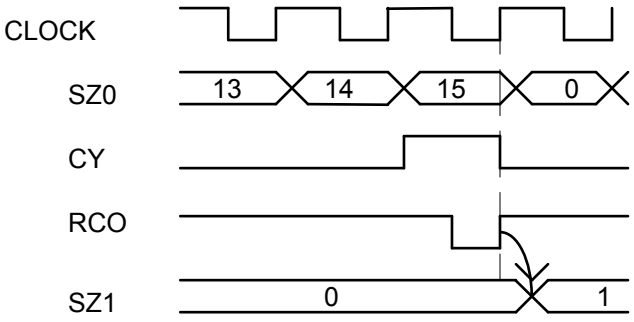
3db 4 bites bináris számláló működése

3.	2.	1.	0.
0000	1111	1111	1111
0001	0000	0000	0000

Aszinkron kaszkádosítások:

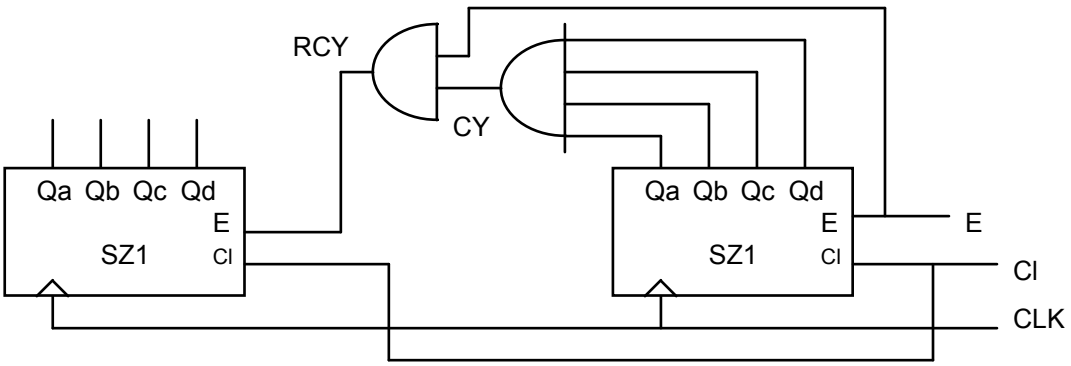


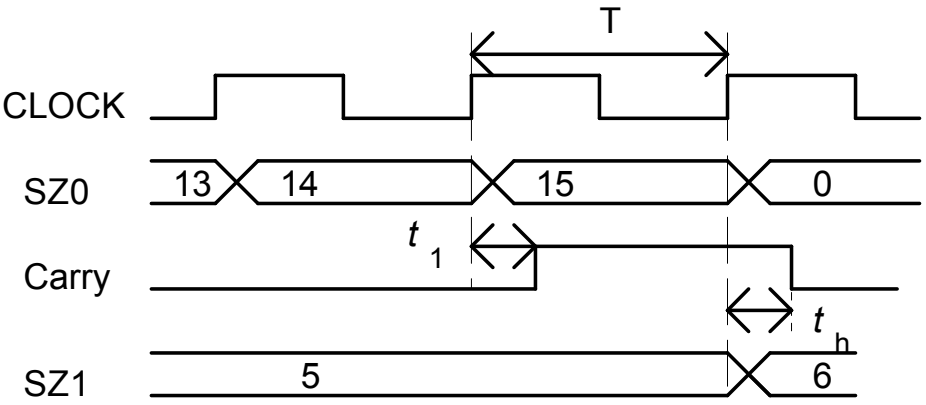




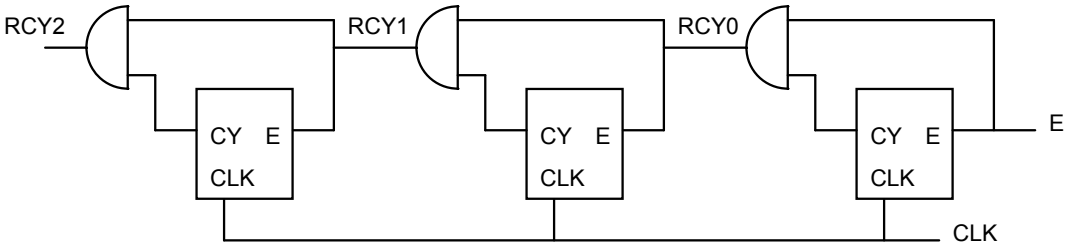
A kaszkádosított számlálók modulusai összeszorzódnak.

Szinkron kaszkádosítások:





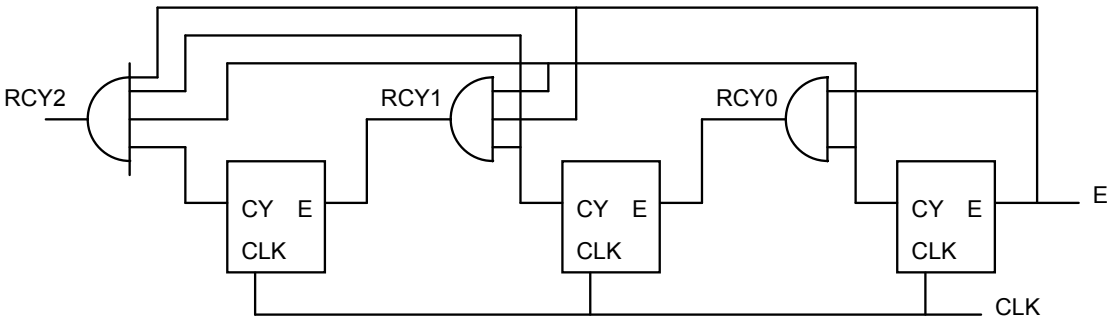
A carry soros terjesztése:



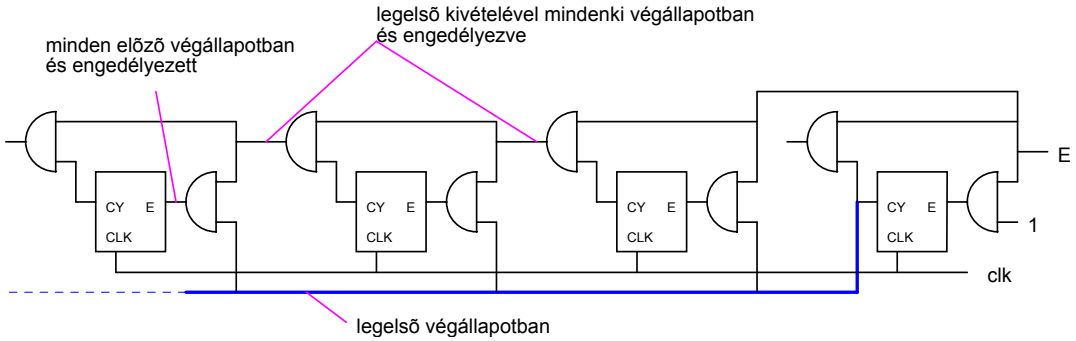
Az engedélyezés az MSB felé egyre jobban késik az órajelhez képest.

Gyorsított kaszkádosítások

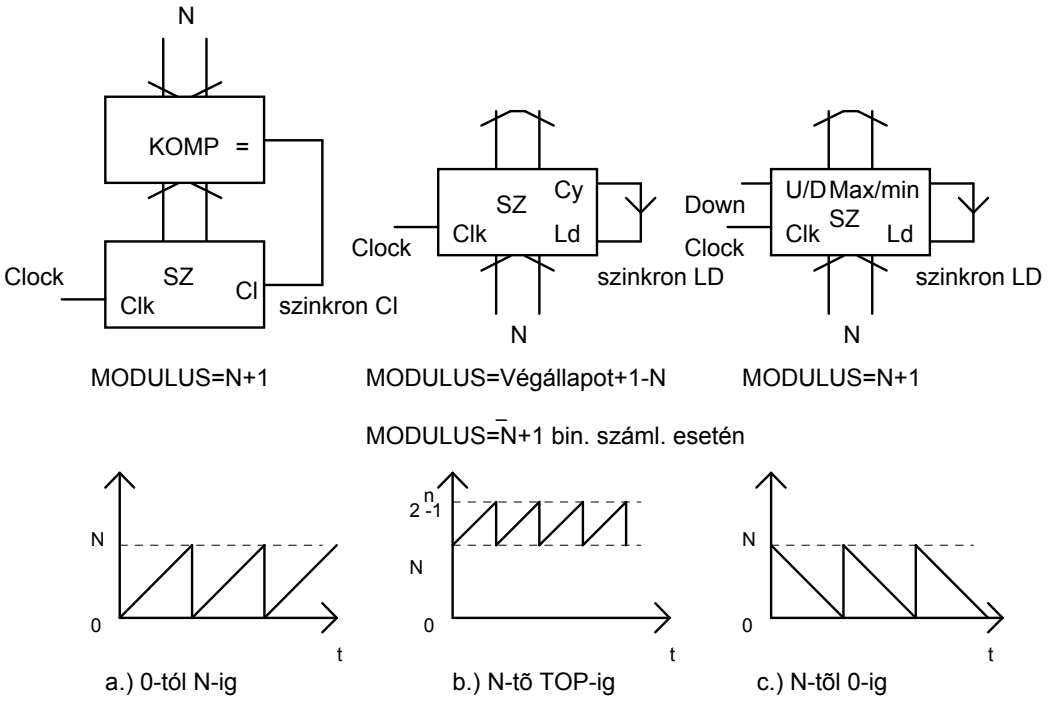
A CY párhuzamos előállításával:



A legelső CY előrecsatolásával:

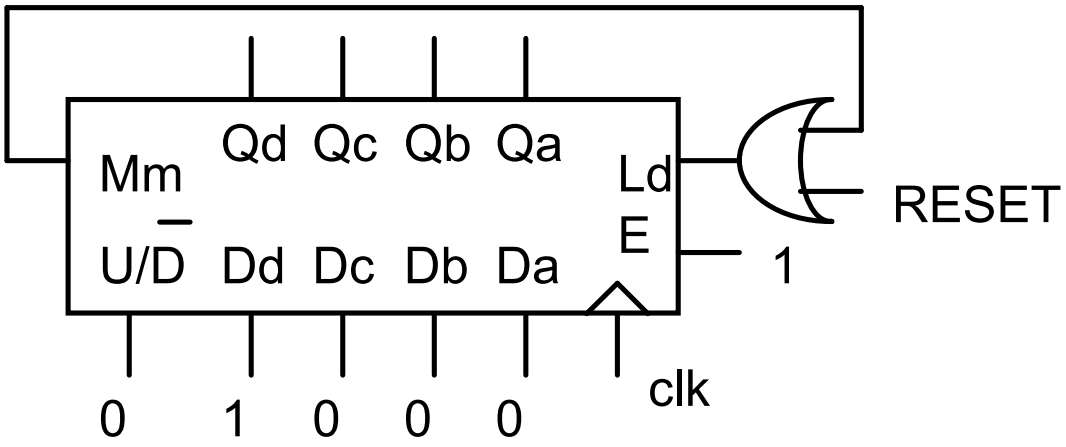


Moduluscsökkentési eljárások

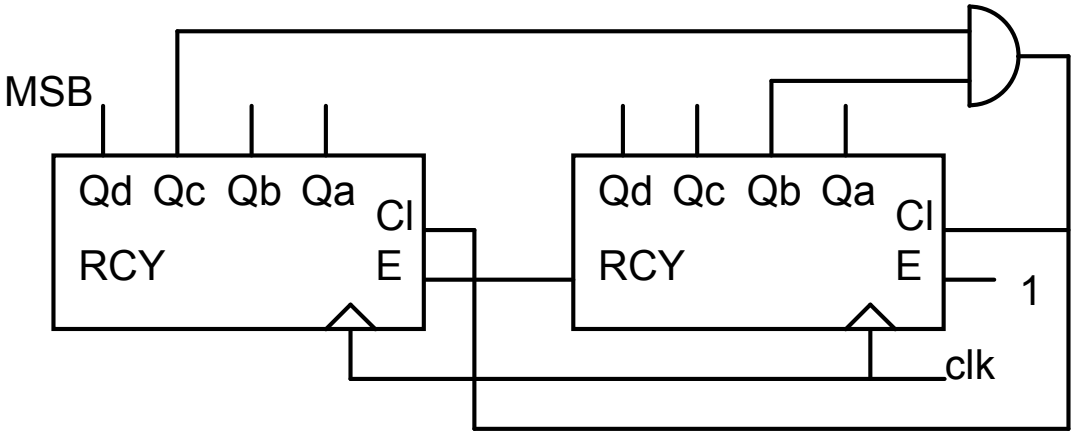


Mintapéldák:

Készítsünk 16-os szinkron tölthető lefele számlálóból 9-es modulusút. A kódolása 8-0.

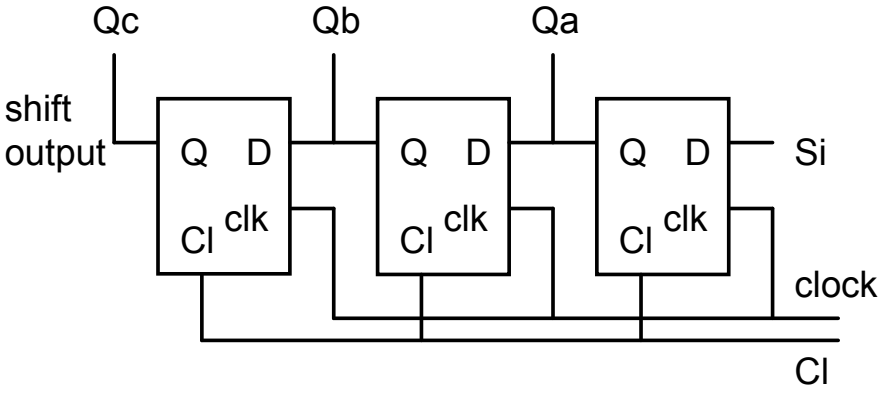


Készítsünk 16-os szinkron törlésű, engedélyezhető felfele számlálókból 67-os modulusút. A kódolása legyen 0-66.
66_D=00100 0010_B
Kaszkádosítunk, majd modulust csökkentünk.

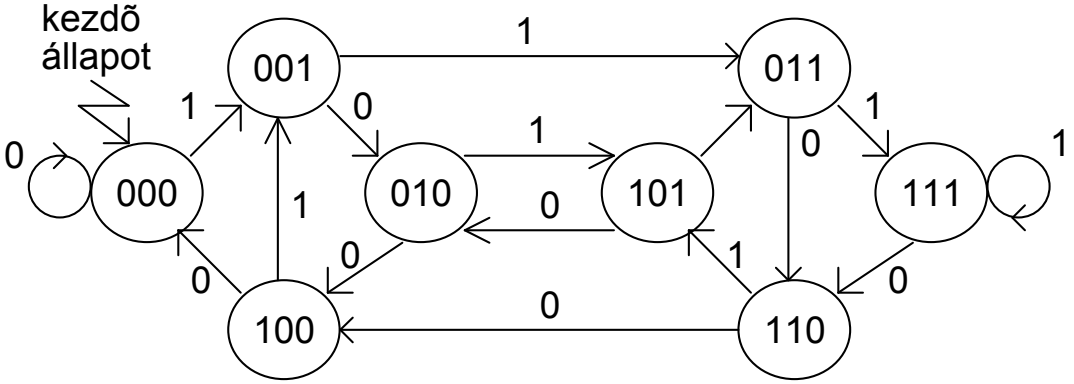


A shiftregiszter

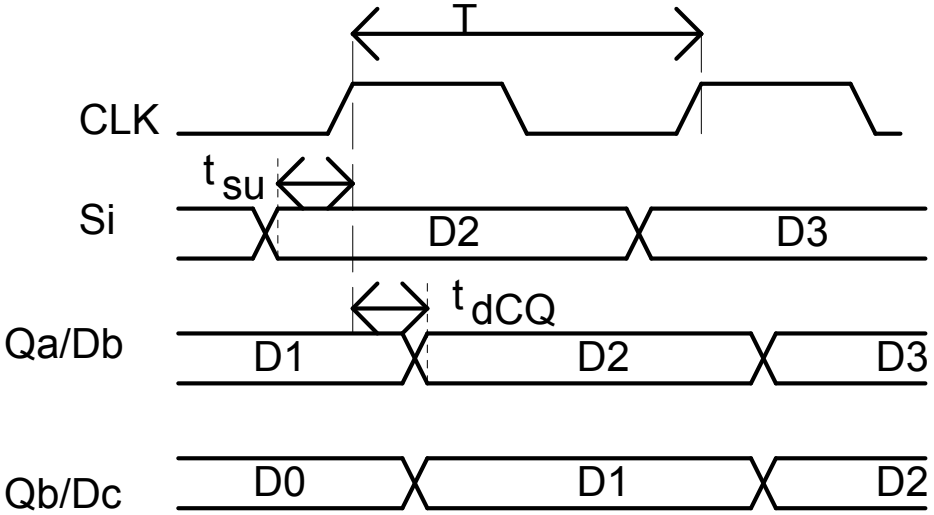
Egy balra shiftelő SHR belső felépítése:



Állapotgráfja:



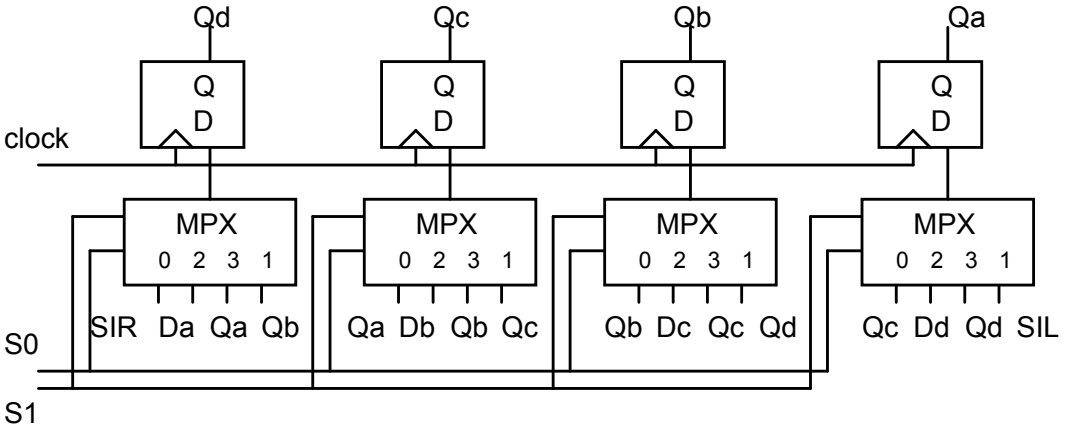
Működésének időd iagramja:



Tölthető 4 bites SHR Verilog leírása

```
module SHR16(clk, E, LD, SI, D, Q);
    input clk;
     input E;
    input LD;
    input SI;
    input [3:0] D;
    output [3:0] Q;
    reg Q;

   always @(posedge <clock>)
      if (LD)
              Q <= D;
      else
       if (E)
             begin
             Q[3:1]=Q[2:0];
             Q[0] <= SI;
            end
endmodule
```

Tölthető, engedélyezhető, jobbra-balra shiftelő shiftregiszter belső felépítése:



S1S0

00 jobbra shift (SHR)

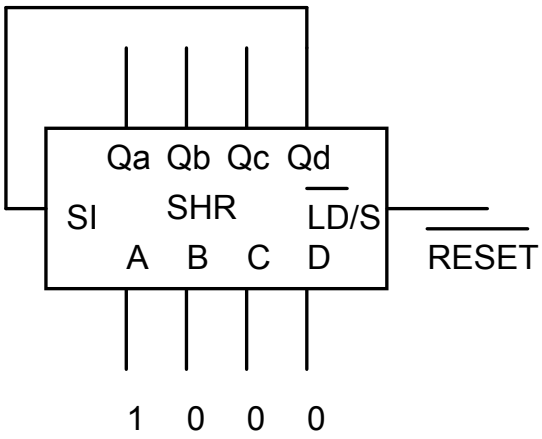
01 balra shift (SHL)

10 betölt (LOAD)

11 tart (HOLD)

Shiftregiszter mint számláló

Gyűrűs számláló:

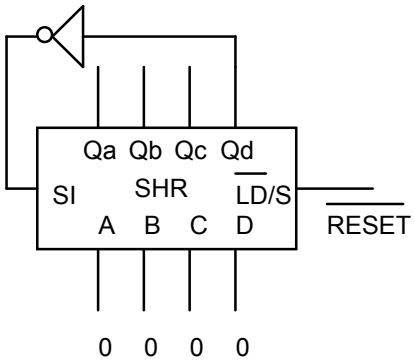


A számláló kódolása Qa, Qb, Qc, Qd:

1000, 0100, 0010, 0001

N bites SHR esetén a modulusa: N

Johnson számláló:

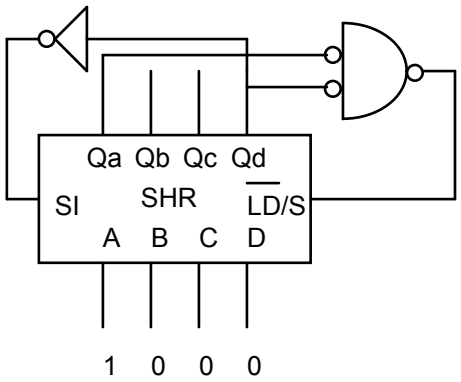


A számláló kódolása Qa, Qb, Qc, Qd:

0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001

Modulusa N bites SHR esetén: 2N

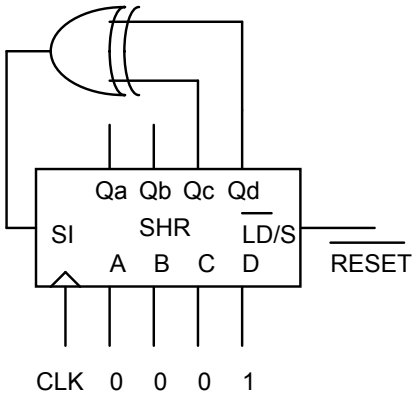
Önkorrigáló Johnson számláló



Néhány órajelen belül tetszőleges állapotból beletalál a normál ciklusba.

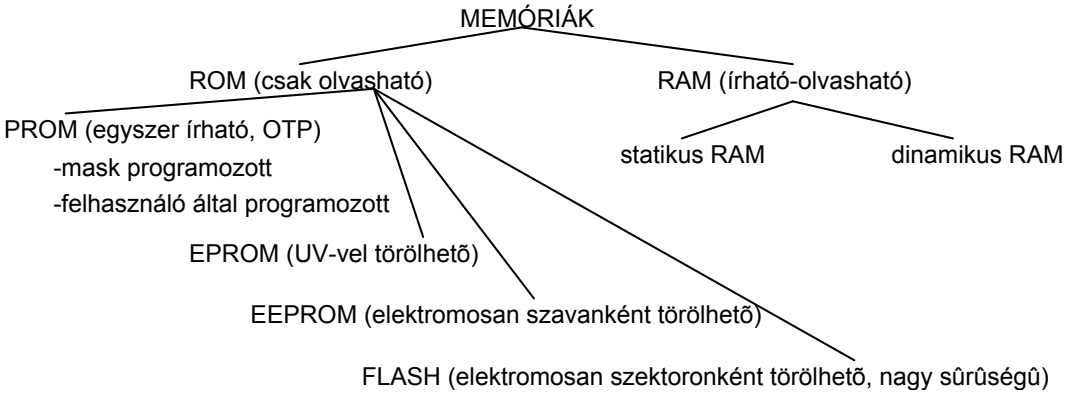
Pl: 1010, 1101, 0110, 1000,…

Álvéletlen generátor MOD 2 visszacsatolt shiftregiszterrel



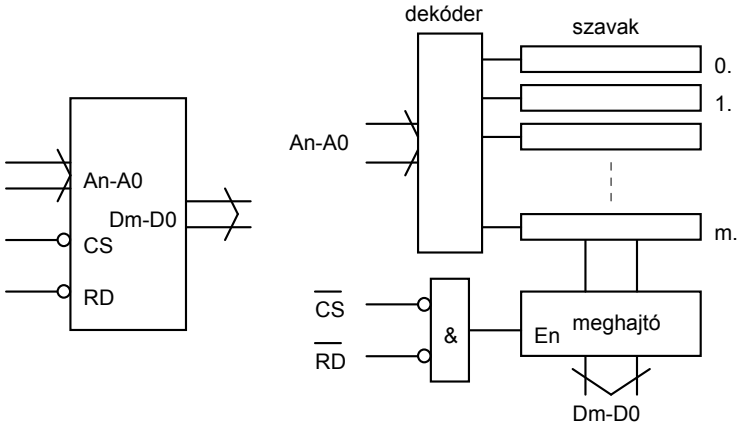
Kódolása:
0001, 1000, 0100, 0010, 1001, 1100, 0110, 1011, 0101, 1010, 1101, 1110, 1111, 0111, 0011
Modulusa N bites SHR és maximális ciklushosszú visszacsatolás esetén: 2^N-1

Memória elemek:



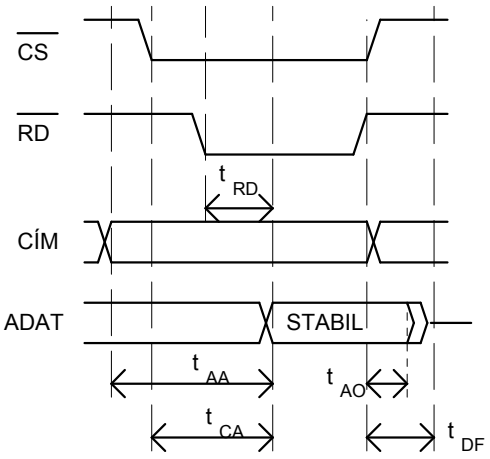
Jellemzők:
szószélesség, byte szám, hozzáférési idő

ROM

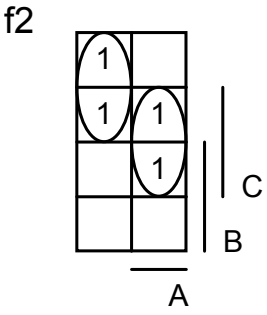


a.) b.)

Olvasás időd iagramja:



ROM mint univerzális KH:



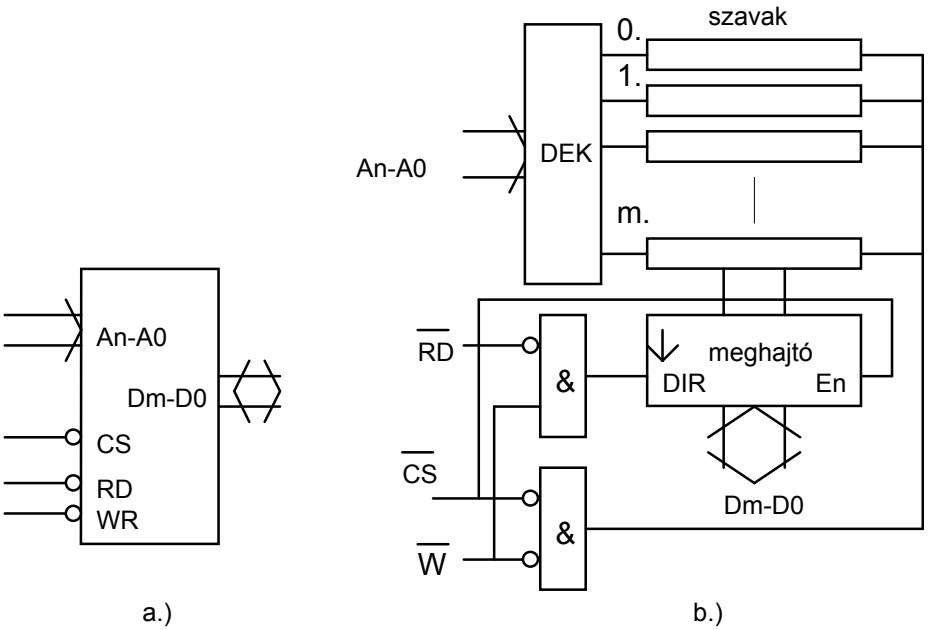
cím	D3	D2	D1	D0		
0.	x	x	1	0	A3	0
1.	x	x	1	0	A2	A
2.	x	x	0	1	A1	B
3.	x	x	0	0	A0	C
4.	x	x	0	1		
5.	x	x	1	0		
6.	x	x	0	0		
7.	x	x	1	1		
8.	x	x	x	x		
.	.	.	.	.		
.	.	.	.	.	OE	0
.	.	.	.	.		
15.	x	x	x	x	CS	0

a.) b.)

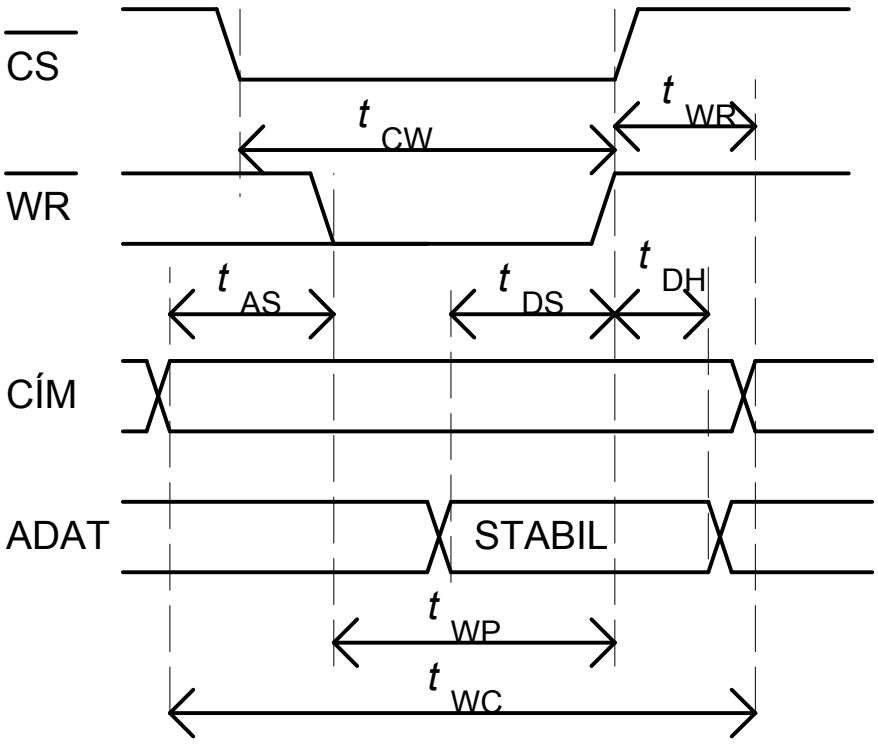
$$f_2 = \overline{A}\overline{B} + AC = \overline{A}\overline{B}\overline{C} + ABC + A\overline{B}C + \overline{A}\overline{B}C$$

A tartalom az igazságtábla.

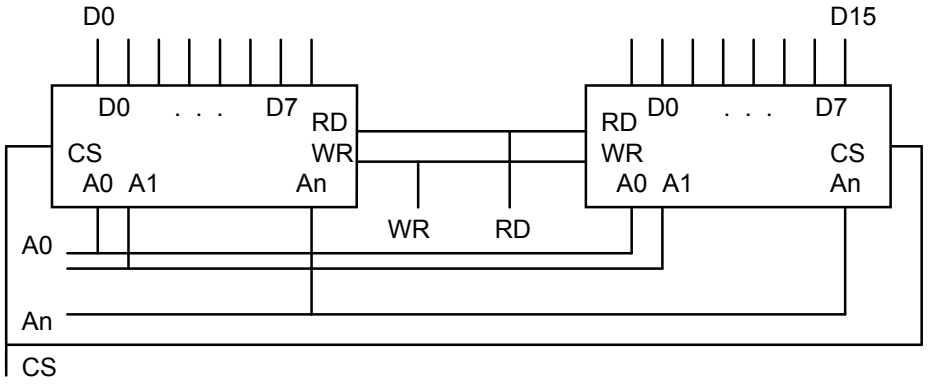
RAM



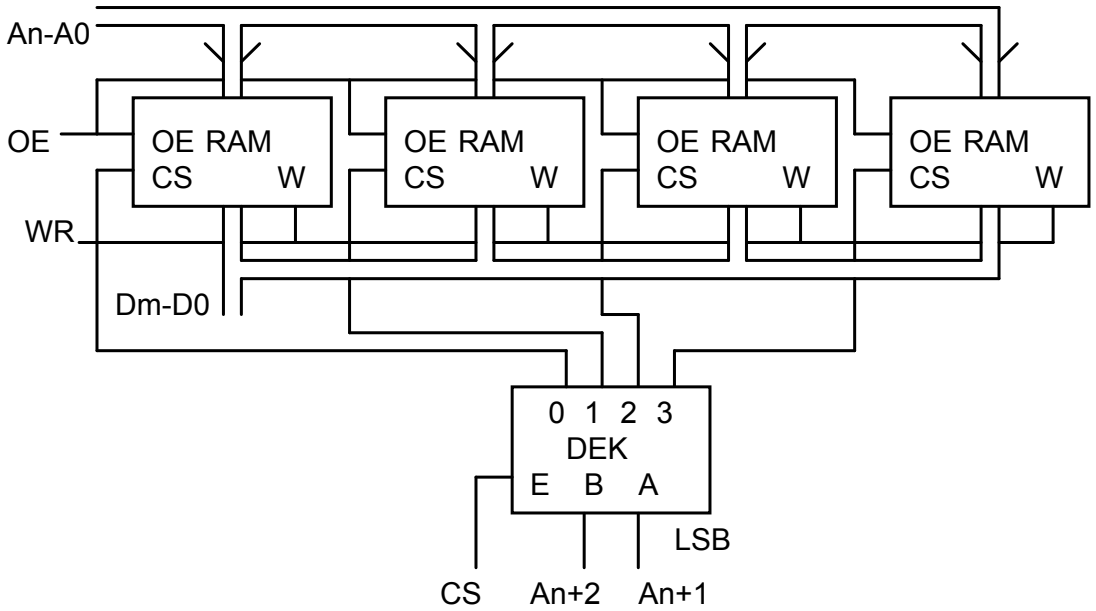
Statikus RAM írási ciklusa:



Memóriák szószélességének növelése

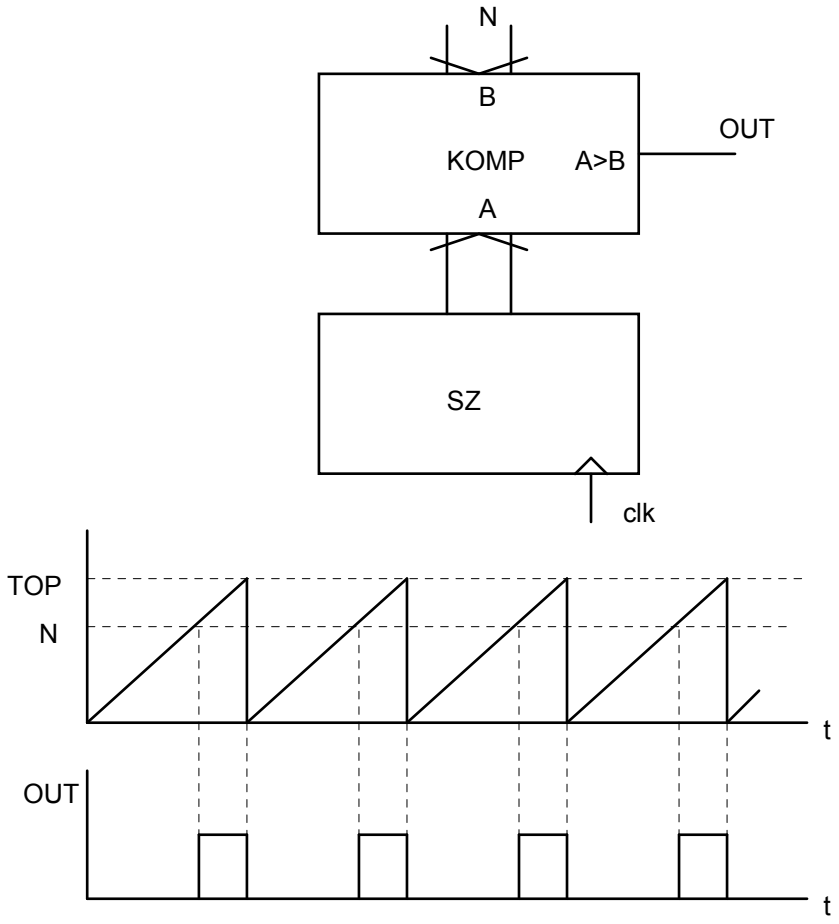


Memóriák kapacitásának növelése

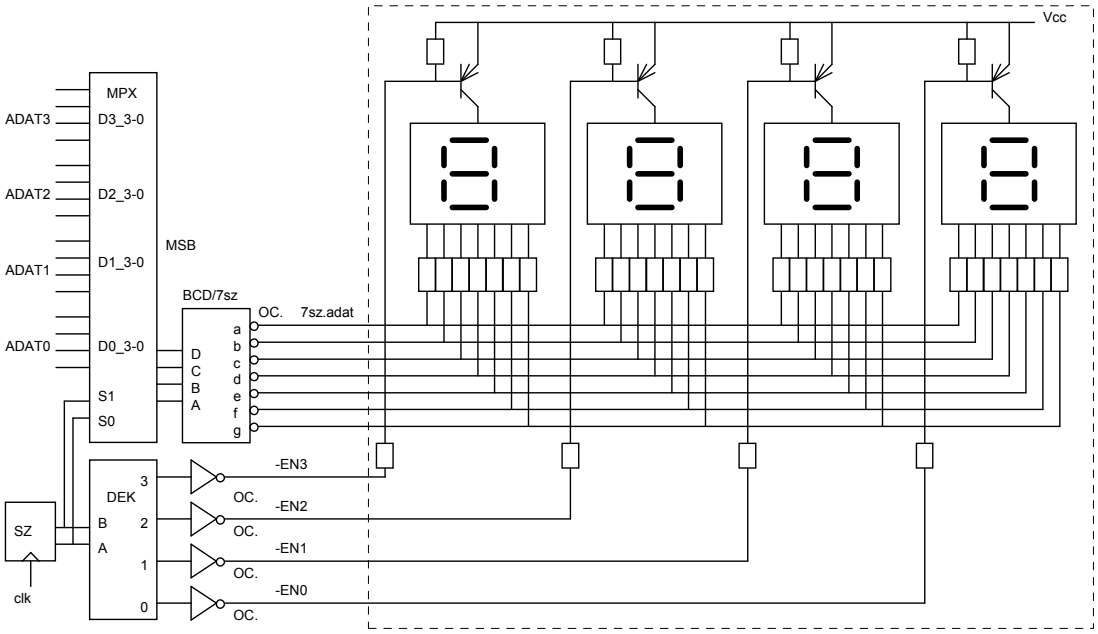


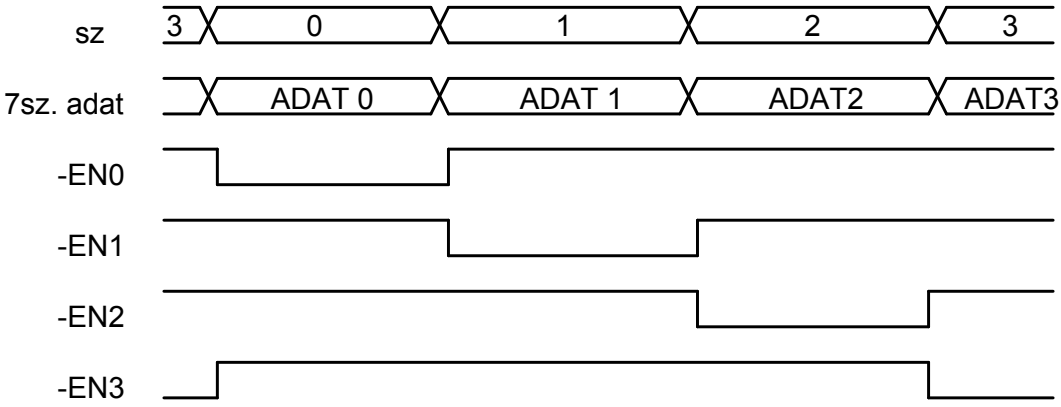
Példák funkcionális elemekkel felépített logikákra:

PWM (változtatható kitöltési tényezőjű jel) előállítása.



Időmultiplexált kijelző megvalósítása





Mátrix billentyűzet kezelése