Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Digitális technika VIMIAA02 3. EA

Fehér Béla, Benesóczky Zoltán
BME MIT

Funkcionális egységek

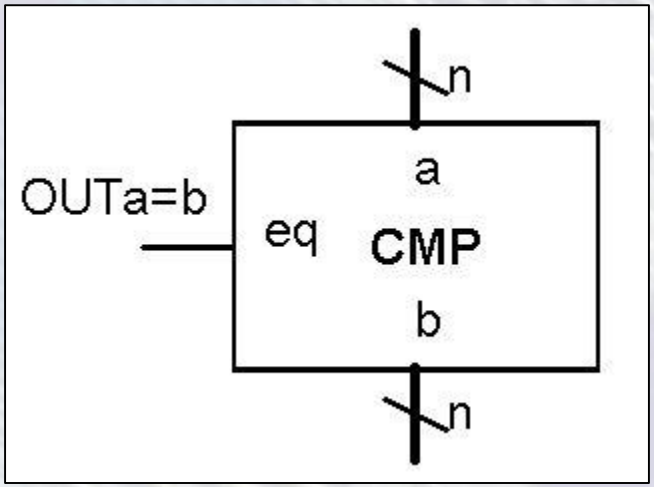
- **Az egyedi „kapuszintű” logikai függvények tervezésénél sokszor egyszerűbb/célravezetőbb szabványos modulokból építkezni és ezekből felépíteni a rendszert.**
- A szabványos modulok (**funkcionális egységek**) célja, hogy a leggyakoribb, ***tipikus feladatokra*** (funkciókra) ***biztosítsanak egyszerűen használható, megbízhatóan működő, skálázható (könnyen módosítható méretű) megoldási lehetőséget.***
- A ***logikai alapelemek*** (kapuk stb.) ***helyett*** azoknál akár sokkal komplexebb ***funkcionális elemekből*** (modulokból) ***építkezünk***.

Funkcionális egységek

- **Általános célú kombinációs logikai funkciók**
  - **Aritmetikai funkciók:**
    összeadó (**ADD**), kivonó (**SUB**), (komparátor (**CMP**)
  - **Logikai funkciók**:
    dekóder (**DEC),** demultiplexer (**DEMUX**), multiplexer **(MUX),** enkóder **(ENC),** prioritás enkóder **(PRI**),
  - Egyéb, esetleg szükséges funkciók
  - Hagyományos technológiában (TTL MSI IC-k) *minden funkcióra önálló alkatrész létezett*,
  - **HDL alapú tervezésnél a tanult *tervezési mintákat használunk.***
  - **Megtervezzük a** feladathoz szükséges **funkcionális egységeket megvalósító modulokat**, majd felhasználjuk **a modulok** szükséges számú **példányát**.
  - **A továbiakban a fenti funkcionális elemeket és Verilog modul megvalósításukat mutatunk be.**

Funkcionális egységek komparátor

- **Komparátor (CMP)**
  - Feladata számok 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



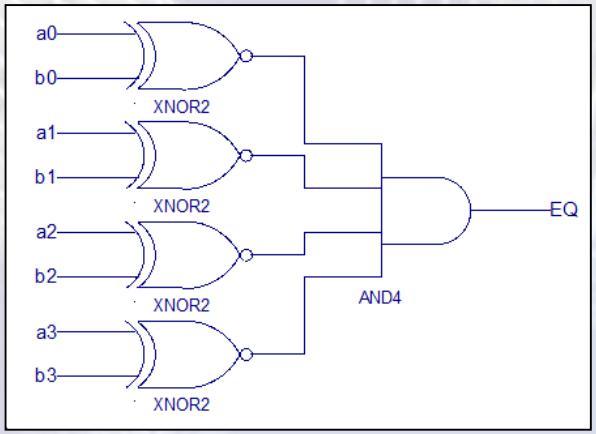
Funkcionális egységek komparátor

- **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

```
wire[3:0] a,b;
wire equ;
assign equ = (a == b);
```



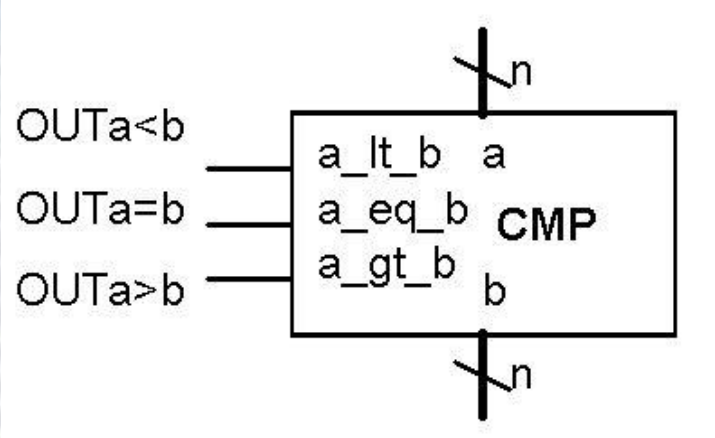
- Tetszőleges kódolásra működik.

Funkcionális egységek komparátor

Teljes funkciójú (más néven **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);
```



Kivonó is használható a nagysági viszony eldöntésére (lásd később).

Funkcionális egységek: Összeadó

- Az összeadás szabályai alapján készítünk egy 1 bites teljes összeadót. A teljes összeadó jelei:
  átvitel bemenet (carry in, **ci**), **összeadandók** (**a, b**)
  atvitel kimenet (carry output, **co**)
  - Most kivételesen megtervezzük, az igazságtáb-lával kezdve. Az összeadási szabályok 3 bitre
  - 0+0+0=0, co=0
    1+0+0=1 sorrend függetlenül, co=0
    1+1+0=0 co=1 -’’- , co=1
    1+1+1=1 co=1 -’’- , co=1
  - co, s 2 bites számként a,b,ci összege

a	b	ci	**co**	**s**
0	0	0	**0**	**0**
0	0	1	**0**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**0**
1	0	0	**0**	**1**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**1**	**1**

Funkcionális egységek: Összeadó

a	b	ci	**co**	**s**
0	0	0	**0**	**0**
0	0	1	**0**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**0**
1	0	0	**0**	**1**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**1**	**1**

- A sum függvény akkor ad 1-et, ha a 3 bemenet közül páratlan számú 1 értékű. Pontosan ilyen tulajdonságú az XOR függvény!

  s = a ⊕ b ⊕ ci
- A **co** akkor 1, ha a 3 bemenet közül legalább 2 db 1 értékű.
- A co-t 1-eseit felírjuk szorzatok összegeként DNF alakban:

1. 2. 3. 4. (4. 4.)

co = /a*b*ci + a*/b*ci +a*b*/ci +a*b*ci (+ **a*b*ci** + **a*b*ci**)

1.+4.: /a*b*ci + **a*b*ci** =b*ci 2.+4.: a*/b*ci + **a*b*ci** = a*ci

3.+4.: a*b*/ci +**a*b*ci** = a*b

Az egyszerűsítések után: **co = a*b + a*ci + b*ci**

Funkcionális egységek: Összeadó

a	b	ci	**co**	**s**
0	0	0	**0**	**0**
0	0	1	**0**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**0**
1	0	0	**0**	**1**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**1**	**1**

sum = a ⊕ b ⊕ ci

co = a*b + a*ci + b*ci

Verilogban:

wire a, b, ci, s;

assign s = a ^b^ci;

assign co = a&b | a&ci | b&ci;

Ugyanez a + operátorral:

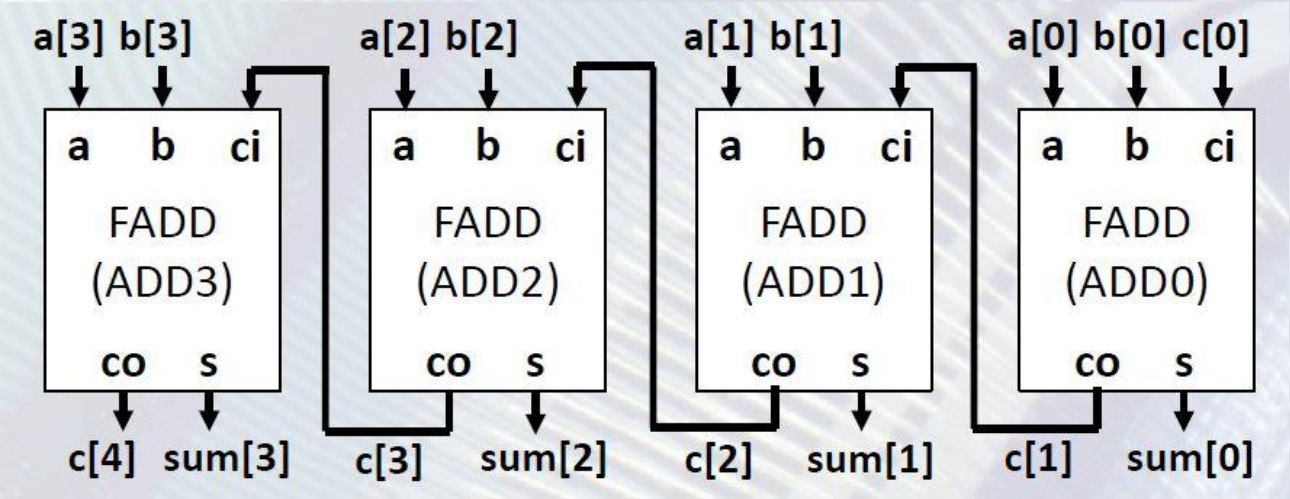
assign {co, s} = a+b+ci;

A **{}** jelek (**konkatenálás**) közé vesszövel felsorolt 1 vagy több bites változókból egytelen annyi bites változó lesz, amennyi a bitszámok összege. A bitek sorrendje is a felsorolás szerinti lesz.

Tehát, ha **a+b+ci;** összeg 2 vagy 3, akkor a 2 bites **{co, sum}**-ban co értéke 1 lesz, egyébként 0.

Funkcionális egységek: Összeadó

Egy bites összeadókból készít hetünk több bitest, ahogy az alábbi ábrán látható. A legkisebb bit ci bemenetére 0-át kell kötni (c[0] = 0).



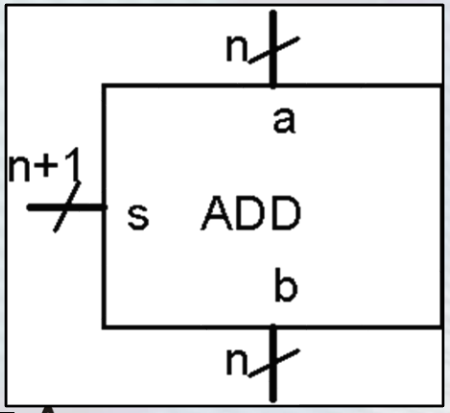
Ilyet készíteni nem célszerű, mert optimálisabb megoldást ad a CAD rendszer, ha rögtön a kívánt bitszámú összeadót készítjük el.

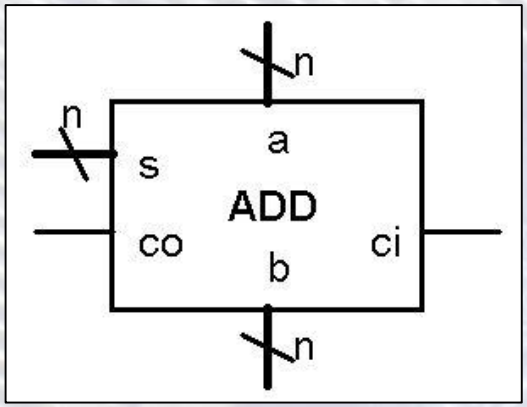
Funkcionális egységek: Összeadó

Ha az eredmény és az operandusok azonos bitszámúak, akkor az eredmény nem mindig férne el a kimenet bitjein pl. $1000_2+1000_2=10000_2$
Ha az eredmény bitszáma 1-el nagyobb, mint az operandusoké, akkor nincs probléma.

Pl: wire [3:0] a,b;
wire [4:0] s;
assign s = a + b;

Ha több bites kaszkádosítható megoldás kell akkor azt így lehet megadni:
wire [3:0] a,b, s;
wire co, ci;
assign {co, s} = a + b + ci;





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 előjel nélküli számok esetén ***csak akkor helyes***, **ha az eredmény nem negatív.**
- **Több bites kivonásnál az átvitelt először ki kell vonni a kisebbítendő következő bitjéből és az így módosult bitből kell kivonni a kivonandó aktuális bitjét.**

Funkcionális egységek: Kivonó

Kivonás példa magyarázattal

	0. bit
	0-1=1
	1
6	110
- 3	- 011
3	1

Menetközben a kisebbítendő helyett a belőle keletkezett részeredményt mutatjuk. Az átvitel miatt megváltoznak egyes bitek, ezeket vastaggal jelöltük.

1.bit	**1. bit**
átvitel	0-1=1
kivonása	
1-1=0	**1**
1**0**0	1**0**0
- 011	- 011
1	11

0. bit: 1-hez, hogy 0-legyen kell 1, az átvitel 1.

Az átvitelt kivonjuk a kisebbítendő 1. bitjéből és az eredménnyel helyettesítjük azt.
1-1=0,
Eztán elvégezzük a kivonást az 1 bitek között.
1-hez, hogy 0 legyen kell 1, az átvitel 1

2. bit	**2. bit**
átvitel	0-0=0
kivonása	nincs
1-1=0	átvitel
000	**00**0
- 011	- 011
11	011

Az átvitelt kivonjuk a kisebbítendő 2. bitjéből és az eredménnyel helyettesítjük azt.
1-1=0,
Eztán elvégezzük a kivonást a 2 bitek között.
0-hez, hogy 0 legyen kell 0.

Ezzel megkaptuk a végeeredményt.

110
- 011
011

Funkcionális egységek: Kivonó

Kivonás (előjel nélküli számok esetén)

Példa Verilog megadásra

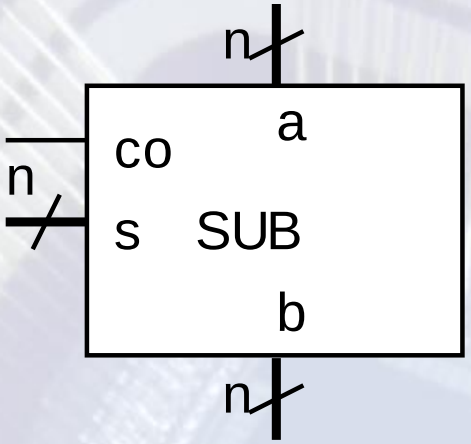
```
module ADD4(input [3:0] a, b, output [3:0] s)
        assign s = a – b;
endmodule
```

Funkcionális egységek: Kivonó

Ha **előjel nélküli** számok kivonásánál, ha a kivonadó nagyobb a kisebbítendőnél, vagyis, **ha az eredmény negatív lenne (a<b)**, akkor az utolsó átvitel értéke **co =1** és az eredmény hibás. **Tehát, ha a-b kivonás után co=1, akkor a<b.** A **kivonás** tehát **nagyság komparátorként is használható**. Szoftverben ezt használják.

Pl.

```
 11
  0011
- 0101
-------
  1110
```



Verilogban:

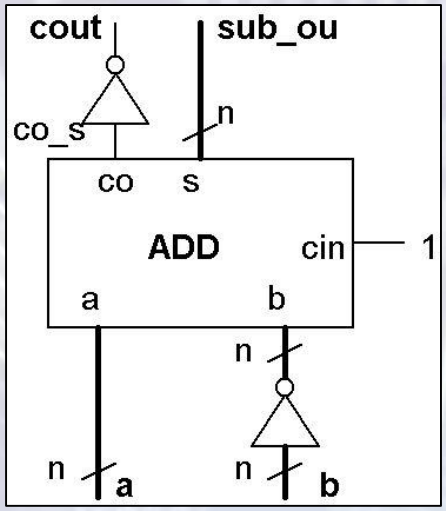
```
wire [3:0] a, b, s;
wire co;
    assign {co, s} = a – b; // Ha co = 1 akkor a < b
```

Funkcionális egységek: Kivonó

- **Kivonó – kettes komplemens hozzáadásával**
  - ***Az összeadó jól működik előjel nélküli (pozitív) és kettes komplemens 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: Összeadó

- **Kettes komplemens kódú számok összeadásására is alkalmas az összeadó.** Kivonni is tudunk vele, ha előtte a kivonandó 2-es komplemensét képezzük.
- 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) 0111 + (-1) +1111 +6 0110	(-7) 1001 + (-2) +1110 -9 0111 +7 hibás!	(+2) 0010 + (+7) +0111 +9 1001 -7 hibás!

- Akkor keletkezik 2-es komplemens túlcsordulás, ha az összeadandók előjele azonos, de az eredmény előjele ettől eltérő.

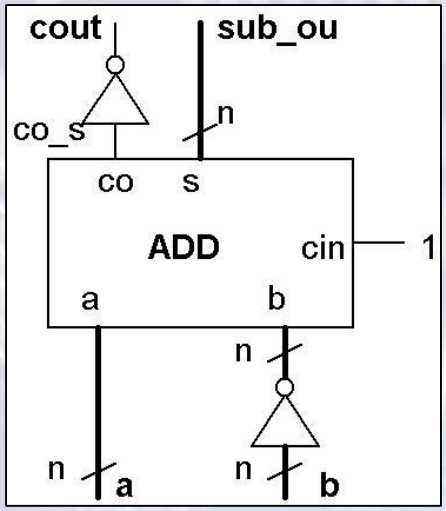
  4 bites számok esetén: OVF = a[3]*b[3]*/s[3] + /a[3]*/b[3] *s[3]
- ***Probléma elkerülése***: használjunk annyi bites számábrázolást, amiben az eredmények elférnek.

Funkcionális egységek: Kivonó

- **Kivonó – kettes komplemens hozzáadásával**
  - ***Az összeadó jól működik előjel nélküli (pozitív) és kettes komplemens 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)

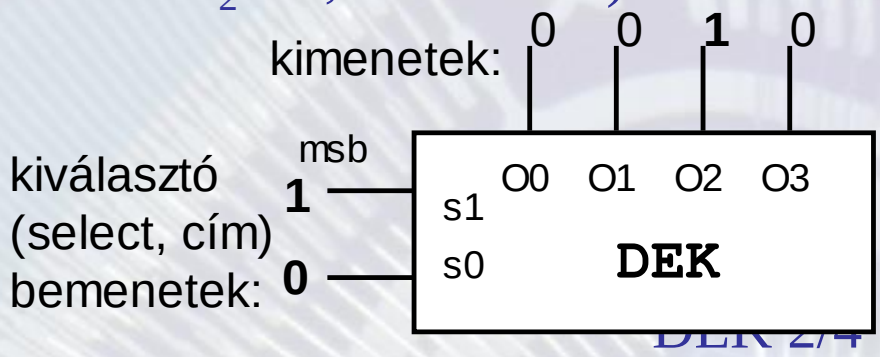


  - Az 1- hozzáadásához az összeadó cin bemenetére 1-et kapcsolunk (S = a+b+cin)
  - ***Az összeadó co kimenete itt fordítva működik***, mint a hagyományos kivonónál, ha zavaró, ***meginvertáljuk***.

Funkcionális egységek: Dekóder

- A dekóder funkciója, hogy **a bemenetére érkező n bites bináris számnak megfelelő sorszámú kimenetét aktivizálja a maximálisan 2^n darab kimente közül** (tehát a kimenete N-ből 1 kódolású). Másképp fogalmazva **d*ekódolja*** a select (kiválasztó) bementén lévő számot. (A rajzon $s=10_2 = 2$, csak O2=1)

s1	s0	O0	O1	O2	O3
0	0	**1**	0	0	0
0	1	0	**1**	0	0
1	0	0	0	**1**	0
1	1	0	0	0	**1**



Elnevezése: DEK bemenet szám/kimenet szám.
A példa dekódere: DEK2/4

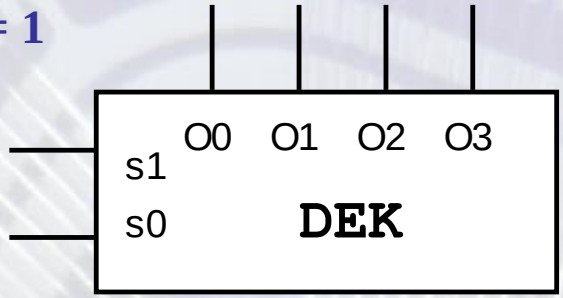
- A dekóder a binárisan értelmezett bemeneti jelekből az n bementű általános ***logikai függvény összes mintermjét előállítja, minden kimenetén egyet:***

O0 = /s1/s0 O1=/s1s0 O2=s1/s0 O3=s1s0

• Funkcionális egységek: Dekóder

- Mivel **a dekóder Oi kimenete akkor 1, ha az s bementén levő számra s = i** ezért az egyes kimeneti függvényeket **konstans komparátor**oknak (konstans összehasonlítók) is tekinthetjük.
- DEK2/4 kimeneteinek szorzat alakú leírása **Verilogban**:

```
module dek2_4( input s0,s1, output O0,O1,O2,O3);
        assign O0 = ~s1&~s0; // ha s[1:0] = 2'b00, akkor O0 = 1
        assign O1 = ~s1&s0;   // ha s[1:0] = 2'b01, akkor O1 = 1
        assign O2 =  s1&~s0;
        assign O3 =  s1&s0;
endmodule
```



Ugyanez az == operátorral és vektor változókkal:

```
module dek2_4(input [1:0] s, output [3:0] O)
        assign O[0] = (s == 2'b00);   // ~s1&~s0;  Az AND kapu konstans komparátor
        assign O[1] = (s == 2'b01);   // ~s1&s0;
        assign O[2] = (s == 2'b10);   //  s1&~s0;
        assign O[3] = (s == 2'b11);   //   s1&s0;
endmodule
```

• Funkcionális egységek: Dekóder

- A dekódernek lehet **engedélyező (en, e) bemenete** is. Ha nincs engedélyezve, akkor az összes kimenete 0. Ha engedélyezve van, akkor a megszokott módon működik.

```
module dek_2_4e(input e, input  [1:0] s  output [3:0] O)
    assign O[0] = en & (s == 2'b00);
    assign O[1] = en & (s == 2'b01);
    assign O[2] = en & (s == 2'b10);
    assign O[3] = en & (s == 2'b11);
endmodule
```

Kombinációs funkcionális egységek Verilog viselkedési leírása

- A Verilog-ban viselkedési leírással is megadhatók a logikák.
- A Verilog nyelvi elemeknél csak a Digitális technika tárgyban használt lehetőségeket részletezzük, már a szintaxis megadásánál is. Pontos (általános) leírás a Verilog bevezetőben található. Azonban csak az előadás és gyakorlat-labor anyagban található részt kérjük vissza.
- A viselkedési leírás szintaxisa: **always@(**érzékenyégi lista**)**
  feltételes vagy
  feltétel nélküli **értékadások**
- Az **always** blokkban **csak reg típusú változónak szabad értéket adni**
- **Az érzékenységi listában** fel kell sorolni a **bemeneti jelek**et.
- **Az értékadások akkor értékelődnek ki, ha az érzékenységi lista bármely jele megváltozik.** (Olyan logikát generál a CAD rendszer, amely mindig az aktuális kiértékelődésnek megfelelően viselkedik.)
- **Kombinációs hálózatok** viselkedési leírásában az **érzékenységi lista helyett elég egy *-ot írni**:

  always@(*)
- Ha az always blokk belső leírása nem egyértelmű kezdettel és véggel rendelkező nyelvi konstrukció, akkor **begin** és **end** közé kell tenni. (Egyébként sem árt.)

Kombinációs funkcionális egységek Verilog viselkedési leírása

Az always blokkban használható **feltételes** szerkezet
if else szerkezet

```
if(kifejezés)
   értékadás1;
else
   értékadás2;
```

Ha a kifejezés nem 0, akkor az **if else közötti rész** hajtódik végre, egyébként az **else utáni**. Az **else** ág **elhagyható**.

Több utasítás esetén az utasításokat **begin end** közé kell tenni:

```
if(kifejezés)
   begin
   értékadás1;
   értékadás2;
   end
else
   begin
   értékadás3;
   értékadás4;
   end
```

Kombinációs funkcionális egységek Verilog viselkedési leírása

Az always blokkban használható **többutas elágazás a case szerkezet.**

case (kifejezés**)**
  konstans1: értékadás(ok)1;
  konstans2: értékadás(ok)2**;**
  ...
    default: default_értékadás(ok);
endcase

Minden konstansnak különbözőnek kell lenni. Ha a **kifejezés megegyezik valamely konstanssal**, akkor az **ahhoz a részhez tartozó értékadások hajtódnak végre**. Ha több értékadás tartozik egy konstanshoz, azokat **begin end** közé kell tenni.
Ha a case kifejezése **egyik konstanssal sem egyezik**, akkor a **default ág** él.

Funkcionális egységek: Dekóder

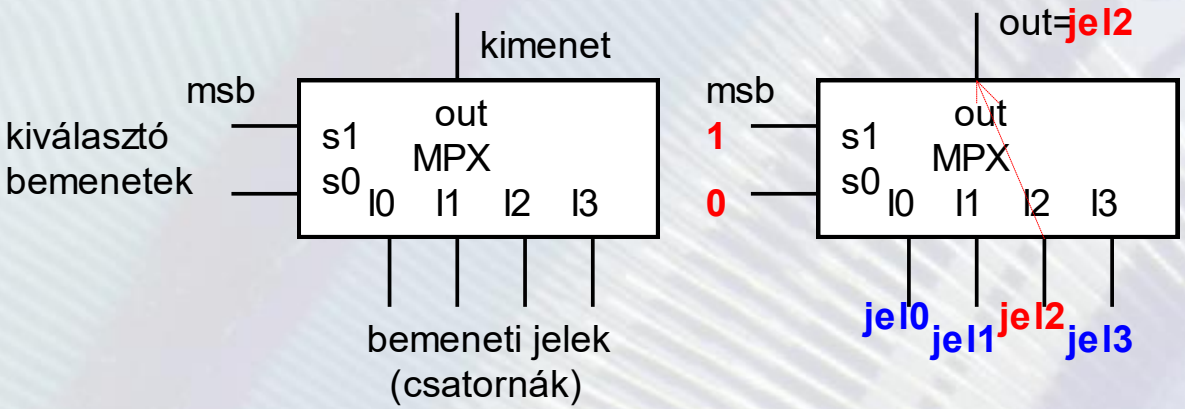
Engedélyezhető DEK2/4 modul viselkedési leírás always blokkal

Az **aways** blokkban **aminek értéket adunk** az **csak reg típusú lehet!**

```
module dek2_4e(input e, input [1:0] s, output reg [3:0] out)
always @ (*)
begin
    if (en)                          // ha en =1
        case (s)
        2'b01: out = 4'b0010;        // ha sel = 01, akkor az 1. kimenet lesz 1
        2'b10: out = 4'b0100;
        2'b11: out = 4'b1000;        // mindem lehetőséget felsoroltunk,
       default: out = 4'b0001;       // ha sel = 00, akkor a 0. kimenet lesz 1
        endcase                      // a default helyett 2’b00-t is írhatnánk
    else                             // mert úgy is minden lehetőséget lefedünk
        out = 4'b0000;               // ha en = 0, minden kimenete 0.
end
```

Funkcionális egységek: Multiplexer

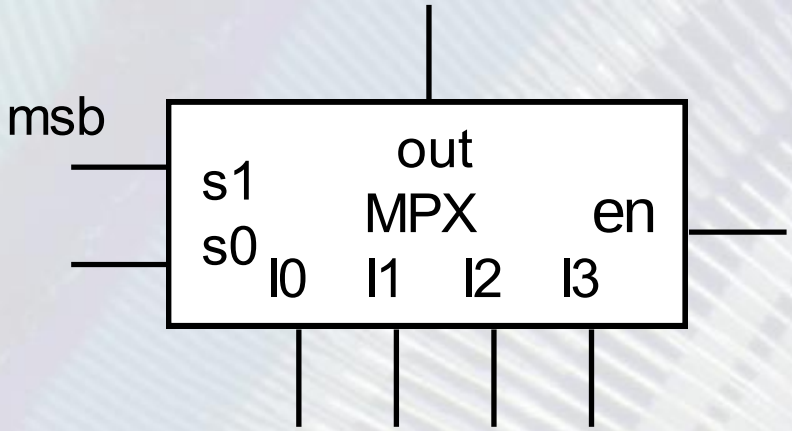
- A multiplexer feladata a több bemeneti adata (csatornája) közül a kiválasztó (select, s) bementére adott bináris számnak megfelelő sorszámúnak a kimenetre (out, o) kapcsolása
- Ezt ***adatút választásnak*** is szokás nevezni
  (Pl. a rajzon $s=10_2 = 2$, csak out = I2, jel2-őt választotta ki.)



- Elnevezése MPX(vagy MUX) bemeneti csatorna szám/1
  A példában MPX4/1
- Jellemző méretek: 2/1, 4/1, 8/1, 16/1

Funkcionális egységek: Multiplexer

- A multiplexer lehet engedélyezhető (en, e).
- Ha nincs engedélyezve, a kimenete 0 értékű. Ha engedélyezve van, a megszokott módon viselkedik.



Funkcionális egységek: Multiplexer

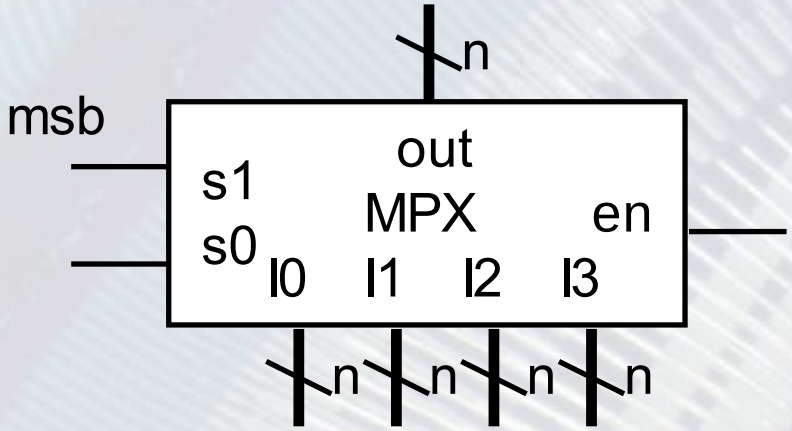
Engedélyezhető MUX4/1 modul viselkedési leírás always blokkal

Aminek értéket adunk az aways blokkban az **csak reg típusú lehet!**

```
module MUX4_1(input [1:0] s, input [3:0] I, input  en, output reg out)
always @ (*)
begin
    if (en)                 // ha en =1
        case (s)
        2'b01: out = I[1];          // ha sel = 01, akkor az I1 bemenetet választja ki
        2'b10: out = I[2];          //
        2'b11: out = I[3];          // minden lehetőséget felsoroltunk,
      default: out = I[0];          // ha sel = 00, akkor az I0 bemenetet választja ki
        endcase                     // minden lehetséges s éréket lefedtünk
    else
        out = 1'b0;                 // ha en = 0,  kimenete 0.
end
endmodule
```

Funkcionális egységek: Multiplexer

- A multiplexer adat bemenetei és kimenete lehet több bit széles.
- A **több bitből álló jelcsoport** neve **busz**
- Az ilyen multiplexer neve **busz multiplexer**



Funkcionális egységek: Multiplexer

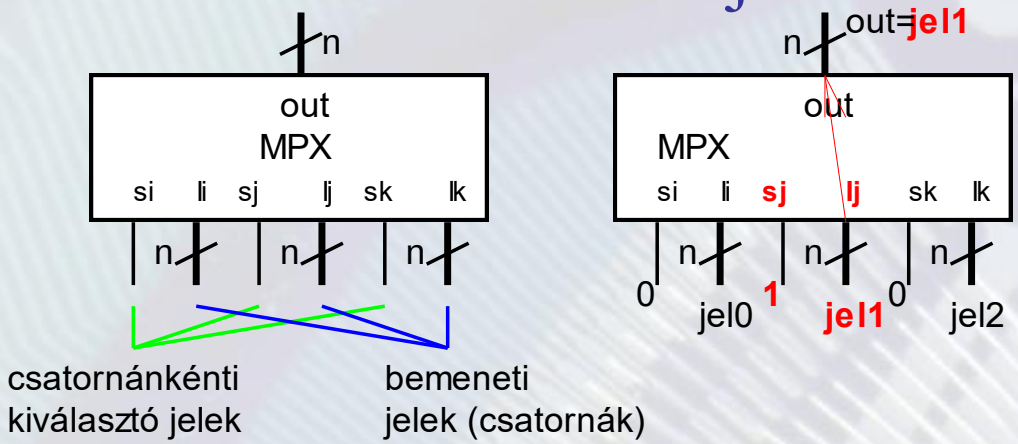
Engedélyezhető MUX8_4/1 viselkedési leírás always blokkal:

```
module MUX4_1(input [1:0] s, input [7:0] I0,I1,I2,I3, input en, output reg [7:0] out)
always @ (*)
begin
    if (en)                 // ha en =1
        case (s)
        2'b01: out = I1;            // ha sel = 01, akkor az I1 bemenetet választja ki
        2'b10: out = I2;            //
        2'b11: out = I3;            // minden lehetőséget felsoroltunk,
      default: out = I0;            // ha sel = 00, akkor az I0 bemenetet választja ki
        endcase                     // minden lehetséges s éréket lefedtünk
    else
        out = 8'b0;                 // ha en = 0,  kimenete 0.
end
endmodule
```

Funkcionális egységek: Multiplexer

Multiplexer csatornához rendelt kiválasztó jelekkel

- Időnként hasznos, ha a multiplexer csatorna kiválasztása a csatornához tartozó kiválasztó jellel történik.



- Az i-edik bemenet (Xi, i-edik csatorna) Eni (Seli) jele engedélyezi, hogy Xi bemenet megjelenjen a kimeneten.
- **Az s jelek közül csak 1 db lehet aktív!** Ezt az s jeleket előállító logikával biztosítjuk. A dekóder pontosan ilyen tulajdonságú kimenetekkel rendelkezik.

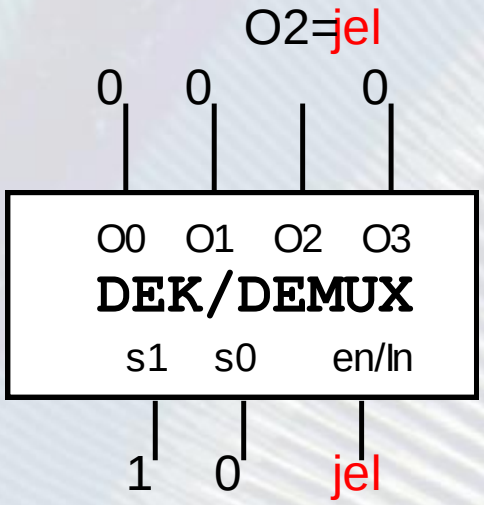
Funkcionális egységek: Multiplexer

Csatornánkénti kiválasztású buszmultiplexer viselkedési leírása always blokkal:

```
reg [7:0]out; //aminek értéket adunk az aways blokkban az csak reg típusú lehet!
wire [3:0] s;
wire [7:0] I0,I1,I2,I3;  // az out és az in azonos méretű bitvektorok
wire en;
always @ (*)
begin
    if (en)                 // ha en =1
        case (s)
        4'b0001: out = I0;          // ha sel = 0001, akkor az I0 bemenetet választja ki
        4'b0010: out = I1;          // ha sel = 0010, akkor az I1 bemenetet választja ki
        4'b0100: out = I2;          //
        4'b1000: out = I3;          //
         default: out = 8'h00;    //  Ha nem 1 a 4-ből kódú az s akkor a kimenet 0!
        endcase                     // most 12 ilyen eset van!
    else
        out = 8'b0;         // ha en = 0,  kimenete 0.
end
```

Funkcionális egységek: DEMUX

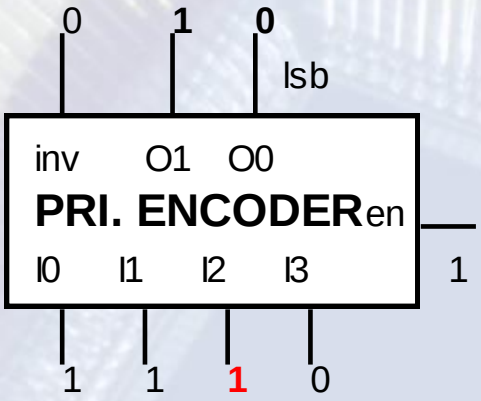
- A **DEMUX** demultiplexer **a multiplexer funkció inverze**
- **Az egyetlen adat bemenet értéke a kiválasztó jelekkek megadott sorszámú kimeneten jelenik meg, a többi kimenet 0.** Lényegében egy ***engedélyezhető dekóderrel azonos felépítésű, csak az en bemenet szerepe itt adat bemenet*** (In).



- Szokásos méretei 1/2, 1/4, 1/8

Funkcionális egységek: Prioritás enkóder

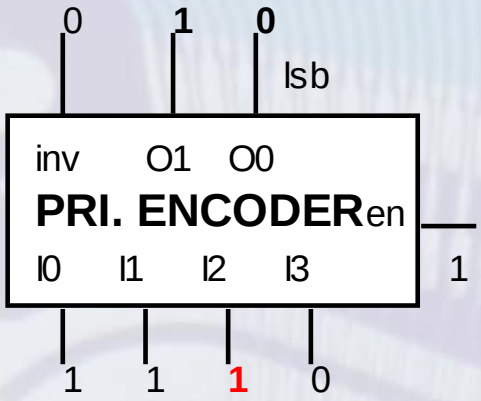
- A **prioriás enkóder** az N bites **bemenetére kapcsolt kód alapján a** kimenetén **bináris számként kiadja, azon *legnagyobb sorszámú* bemenetének sorszámát, ahol 1-es van**. (Függetlenül attól, hogy a többi bemenetén milyen érték van.)
- Ha minden bemenetén 0 van, hibás a kimenete. Ezért invalid (inv, érvénytelen) jelzés kimenete és **engedélyező (en)** bemenete is van.
- Az **inv** kimenete akkor 1, ha nincs engedélyezve vagy csupa 0 a bemenete.



Funkcionális egységek: Prioritás enkóder

- A **prioriás enkóder** igazságtáblája.

en	I3	I2	I1	I0	O1	O0	inv
0	x	x	x	x	0	0	1
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	x	0	1	0
1	0	1	x	x	1	0	0
1	1	x	x	x	1	1	0



- Az x-el a don't care (közömbös, tetszőleges) értéket jelöljük. Ilyen a Verilogban is van.
  Pl. Az utolsó sorban az I2I1I0 értéke közömbös. Ez $2^3 = 8$ igazságtábla sort helyettesít.
- A Verilog **casex** szerkezetében a case konstansai x értékekeket is tartalmazhatnak.

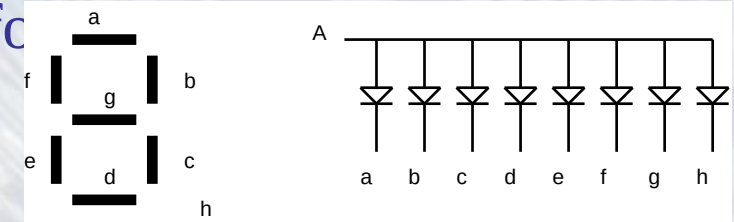
Funkcionális egységek: Prioritás enkóder

Prioritás enkóder viselkedési leírása always blokkal:

```
reg [1:0] out;   // aminek értéket adunk az aways blokkban az csak reg típusú lehet!
wire [3:0] I;    // bemenetek
wire en, inv;    // engedéylezés és invalid kimenet jelzés
always @ (*)
begin
    if (en)                  // ha en =1
        casex (I)
        4'b0001: out = 2’b00;  // a 0. bemenet 1, ezért out =0
        4'b001x: out = 2’b01;  // az 1. bemenet 1, előtte 0-ák,  out =1
        4'b01xx: out = 2’b10;  // a 2. bemenet 1, előtte 0-ák, out =2
        4'b1xxx: out = 2’b11; // a 3. (legnagyobb) bemenet 1, out =3
         default:  out = 2’b00;  //  Ha I=0, akkor a kimenet 0!
        endcase                          //
    else
        out = 2'b0;                  // ha en = 0,  kimenete 0.
end
assign inv = ~en | (I == 4’b0000); // a kimenet nem érvényes, ha en=0 vagy a
                                   // bemenet csupa 0.
```

Funkcionális egységek: Hex/7szegmenses dekóder

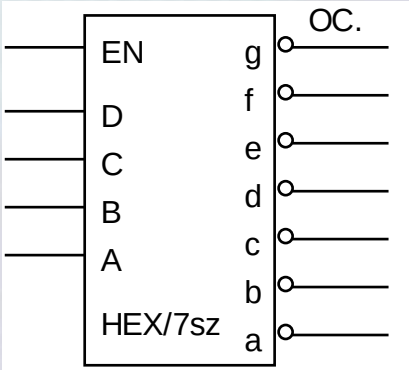
- A 7 szegmenese kijelzőn számokat és korlátozottan betűket lehet megjeleníteni.
- A LED kijelző szegmenseinek elhelyezkedése és bekötésük. A ú.n. *közös anódos* kijelző LED-jeinek pozitív lába közösítve van, erre kell a tápot adni. Azon LED-ek fo nak világítani, amelyek katódja *ellenálláson keresztül* a 0V-ra kap-csolódik. (Az ellenállásokat nem rajzoltuk be.)



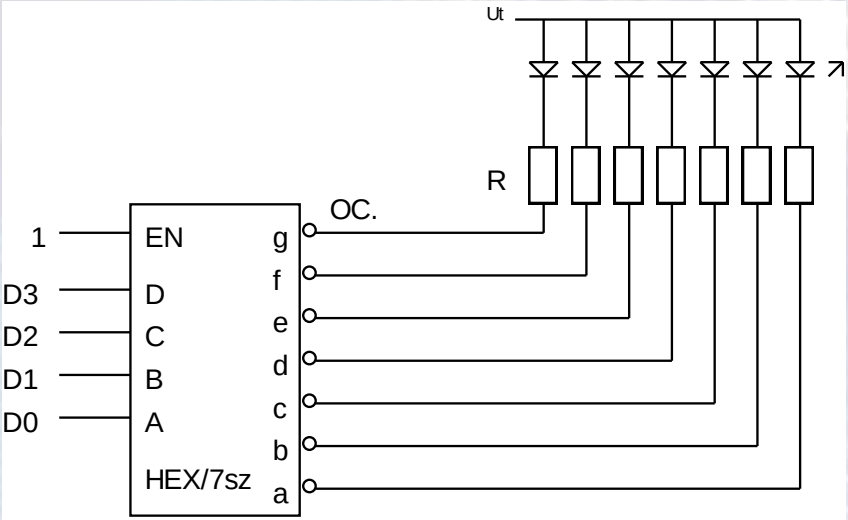
- Például az 1 kijelzéséhez a b és c LED-eket kell kigyújtani.
- A HEX/7 szegmenses dekóder olyan kombinációs hálózat, amely **a bementére kapcsolt 4 bites bináris számhoz a kimenetén az bináris szám hexadecimális megjelenítéséhez szükséges 7 bites kódot adja**. (Általában az *a* szegmenshez rendelik az LSB-t.)

Funkcionális egységek: Hex/7szegmenses dekóder

- A rajzjele alacsony aktív kimenet esetén:



- Példa a bekötésére:



Funkcionális egységek: Hex/7szegmenses dekóder

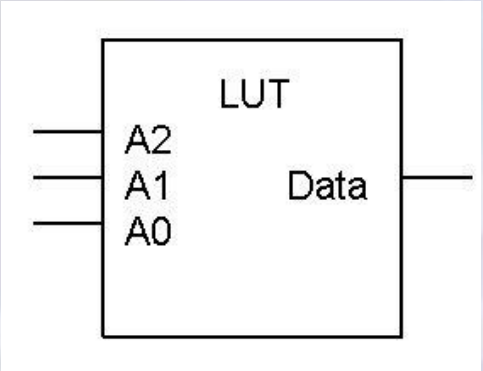
Verilog kód: Magas aktív kimenet esetén. (Itt azon szegmensek értéke 1, amelyeknek világítani kell. A dec_out kimenet meginvertálásával megkapjuk az alacsony aktív kimenetű verziót.

```
module dec_7seg(
   input  wire [3:0] dec_in,
   output reg  [6:0] dec_out
);

//A hétszegmenses dekóder igazságtábláját könnyen leírhatjuk Verilog nyelven
//always blokkban case utasítást használva. A kimenet esetén at '1' érték
//jelentse az aktív (bekapcsolt) szegmenst. A dekóder kimenetének értelmezése:
//dec_out[0]: A, dec_out[1]: B, dec_out[2]: C, ..., dec_out[6]: F
always @(*)
begin
   case (dec_in)
      4'h1   : dec_out <= 7'b0000110;
      4'h2   : dec_out <= 7'b1011011;
      4'h3   : dec_out <= 7'b1001111;
      4'h4   : dec_out <= 7'b1100110;
      4'h5   : dec_out <= 7'b1101101;
      4'h6   : dec_out <= 7'b1111101;
      4'h7   : dec_out <= 7'b0000111;
      4'h8   : dec_out <= 7'b1111111;
      4'h9   : dec_out <= 7'b1101111;
      4'ha   : dec_out <= 7'b1110111;
      4'hb   : dec_out <= 7'b1111100;
      4'hc   : dec_out <= 7'b0111001;
      4'hd   : dec_out <= 7'b1011110;
      4'he   : dec_out <= 7'b1111001;
      4'hf   : dec_out <= 7'b1110001;
      default: dec_out <= 7'b0111111;
   endcase
end
endmodule
```

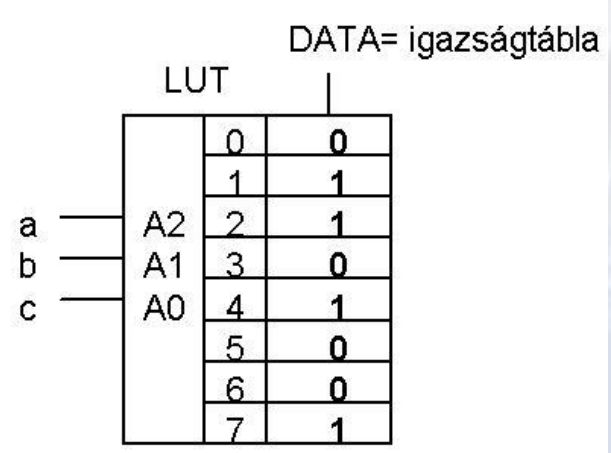
A memória táblázat, mint univerzális logikai elem

- LUT memória táblázat (FPGA logikai elem)
- A címbitekkel (A[n-1:0]) megcímzett rekeszének tartalmát adja ki az adat (Data) kimenetén.
- N darab címbit esetén 2^N rekesze van.



Az ábrák 8 rekesszel rendelkező (3 címbit) LUT-okat mutatnak.

- A LUT-ot úgy használhatjuk univerzális kombinációs hálózatként, hogy a változó-kat a cím bemenetekre kötjük és tartalom-ként az igazságtáblázatot töltjük bele. Figyelni kell a bemeneti áltozók sorrendjére!
- A laborban használt SPARTAN3 FPGA LUT-jainak 4 címbemenete (16 ekesze) van.



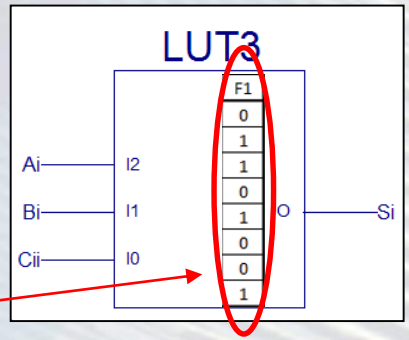
A memória táblázat, mint univerzális logikai elem

- **Függvény megadása:** Az igazságtábla kimeneti jel oszlopának értékeit konstans jelként beírjuk a memória sorindex szerinti címeire
- **A bemeneti változókat az azonos helyiértékű címbitekre kell kötni. (A példa rajzain A,B,C-> I2,I1,I0)**

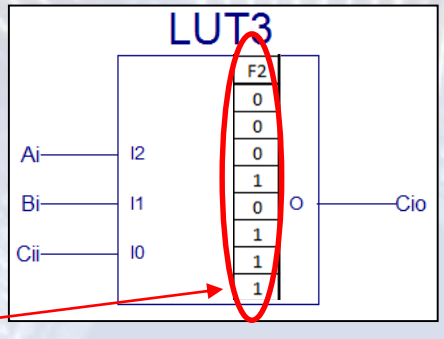
  - A korábbi teljes összeadó függvényei

    F1: S és F2: Co

BEMENETEK				KIM
INDX	A	B	C	F1
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1



BEMENETEK				KIM
INDX	A	B	C	F2
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



Digitális technika 3. EA vége