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

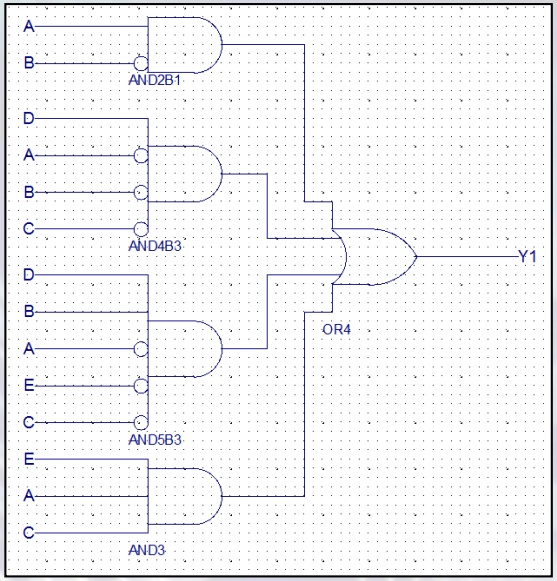
Vetítős előadások

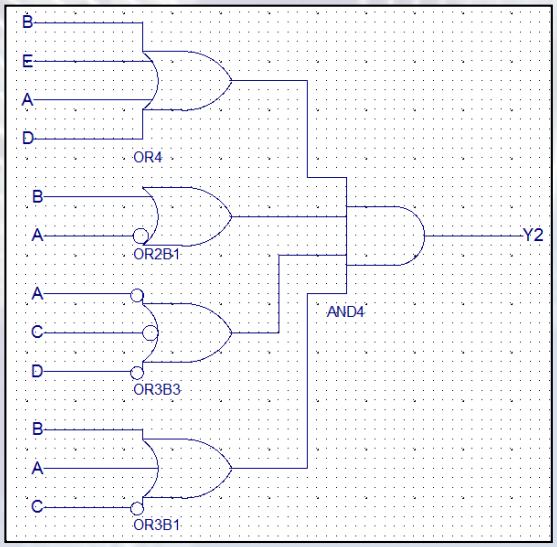
- **Ebben az évben az előadások további két teremben is, kivetítőről követhetők, a kapcsolat így nem elég közvetlen**
- **Ennek javítására beiktatunk visszajelzési lehetőséget**
- **Bármelyik teremből kérdések, megjegyzések emailben küldhetők a benes@mit.bme.hu címre, de csak ... @edu.bme.hu címről,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

Minimalizálási algoritmusok

- **Eddig csak kétszintű realizációról volt szó** (***A jel maximum 2 kapun át jut a kimenetre)***
- ***Ezeknél a minimalizálás a szorzat tagokból (SOP) ill. Az összeg tagokból (POS) változókat tud eltüntetni és a felesleges szorzat ill. összeg tagokat. Tehát csökkenhet az első szint kapuinak és kapu bemeneteinek száma, továbbá a második szint kapu bemeneteinek száma.***
  - Sum-of-Products (SOP, ÉS-ek VAGY-a)
  - Pl: Y1 =A/B+/A/B/CD+ /AB/CD/E+ACE

  illetve
  - Product-of-Sums (POS, VAGY-ok ÉS-e)
  - (INV) – OR – AND
  - Pl: (A+B+D+E)*(/A+B)*(/A+/C+/D)* (A+B+/C)





Minimalizálási algoritmusok

- **További minimalizálási lehetőségek**
  - **Többszintű megvalósítás** (***2-nél több*** kapun terjed a jel a kimenetig)
    - Alapötlet: ***közös szorzatkifejezések kiemelése***
    - Előny: esetleg kevesebb erőforrás (kapu) is elég
    - Hátrány: nagyobb jelterjedési idő (lassabb működés)
  - **Több kimenetű hálózat egyszerűsítése**
    - Közös szorzatkifejezések kiemelése és ***megosztása*** (odavezetése a megfelelő kimenetekhez)
    - Előny: esetleg kevesebb erőforrás (kapu) is elég
- **Speciális optimalizáló programok léteznek, pl. MIS-II**

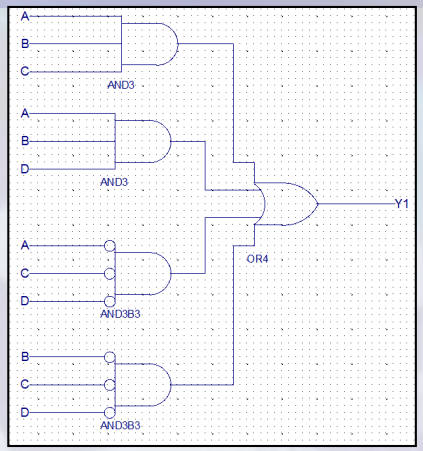
Minimalizálási algoritmusok

- **Többszintű realizáció**

  A 2 szintű SOP minimalizálás eredménye:

  F = **AB**C+**AB**D+/A**/C/D**+/B**/C/D**

  4 db AND3 + 1db OR4 kapu



- Kiemelhető szorzatok keresése

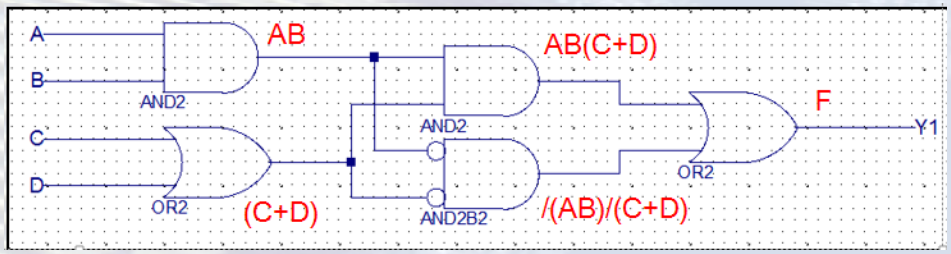
  F = **AB**(C+D) + (/A+/B) **/C/D**

  /A+/B =/(AB), **/C/D**=/(C+D)

  I. Eredmény: F = (AB)(C+D) +/(AB)/(C+D)

  AB és /AB-hez egyetlen AND kapu és egy inverter elég. Hasonlóan (C+D) és /(C+D)-hez egy OR és egy inverter elég.

  3 db AND2 + 2 db OR2



Minimalizálási algoritmusok

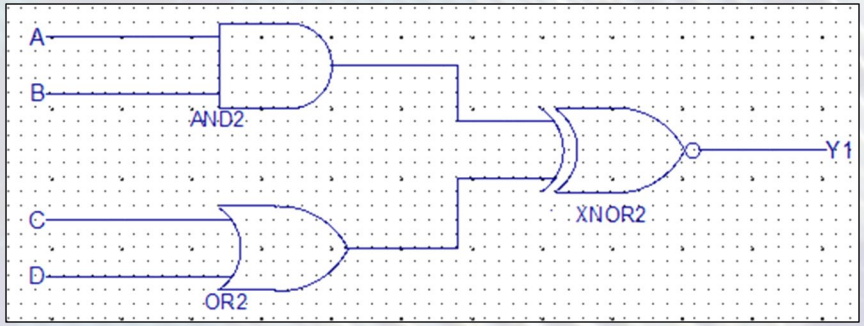
Most további egyszerűsítés lehetséges:

X * Y + /X * /Y = X XNOR Y

F=(AB)(C+D) +/(AB)/(C+D) = **(AB) XNOR (C+D)**

II. eredmény (EXNOR kaput is használva)**:**

F = (AB) XNOR (C+D)



1db AND2, 1db OR2, 1db XNOR

Minimalizálási algoritmusok

Több kimenetű minimalizálás

Példa: Egy logika bemenetére 4 bites számot adunk (a bitjei:
8 4 2 1
ABCD). ***Ha a szám érvényes BCD kód*** (szám <= 9), ***akkor a 4 bites kimenet a számot adja***:
A_OUT=A, B_OUT=B, C_OUT=C. D_OUT=D
ha szám nem érvényes BCD kód (szám > 9***), akkor 1111-et:***
A_OUT=1, B_OUT=1, C_OUT=1. D_OUT=1

Külön előállítjuk a hibás kódot jelző logikát:
Az igazságtábla kitöltése, DNF felírása és egyszerűsítése után:

ERROR = A(B+C) Mert A(8)+B(4) vagy A(8)+C(2) bitek 1 értéke esetén a szám > 9

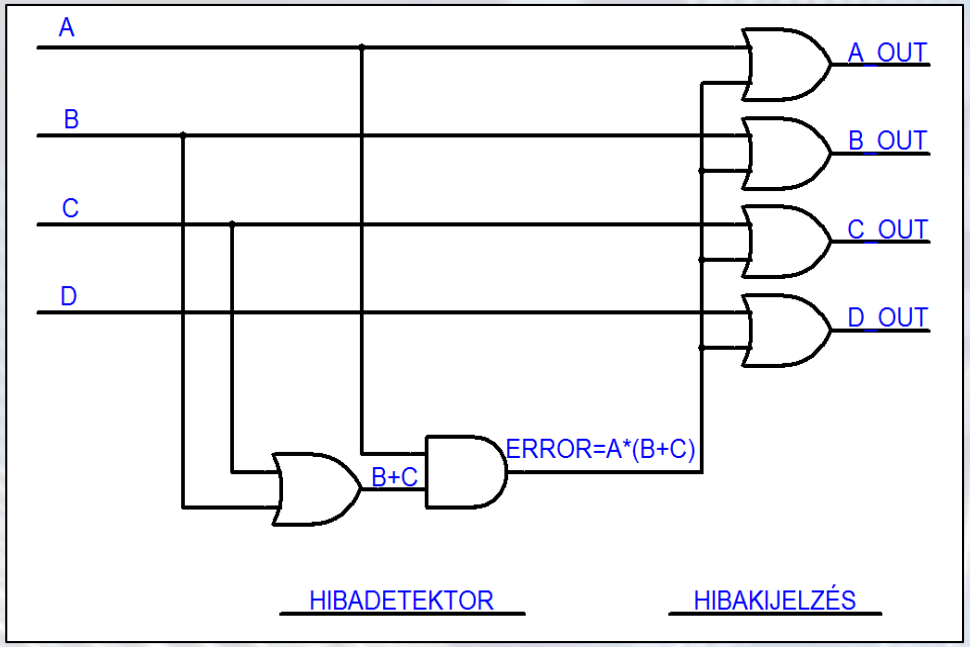
X_OUT= X + ERROR

Minimalizálási algoritmusok

- **Több kimenetű minimalizálás**

  - ERROR = A*(B + C)
    A_OUT = A+**A(B+C)**
    B_OUT = B+**A(B+C)**
    C_OUT = C+**A(B+C)**
    D_OUT = D+**A(B+C)**

    Az **A(B+C)-t egyszer valósítjuk meg** és mind a **4 kimenetnél felhasználjuk**.



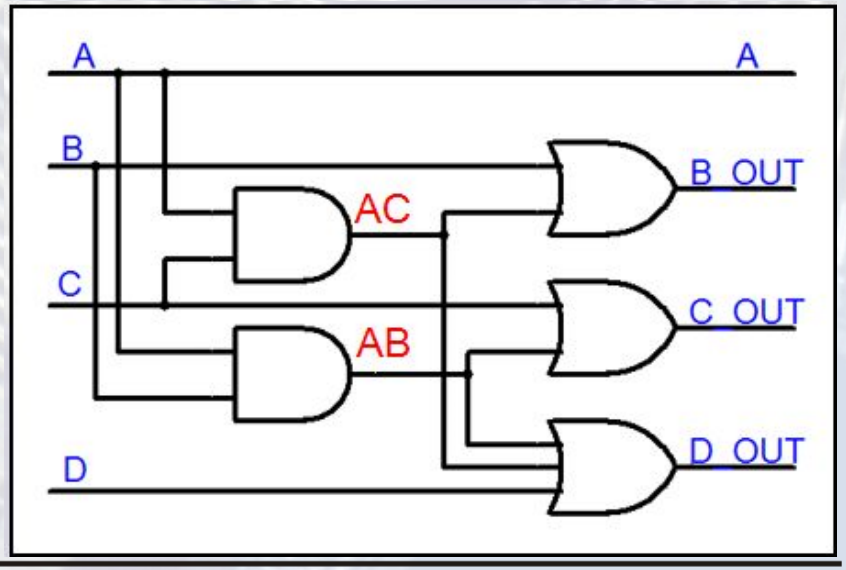
6 db 2 bemenetű kaput használunk

Minimalizálási algoritmusok

Kimenetenkénti egyszerűsítés. Csak ponált változók, az elnyelési tulajdonságot használjuk, Utána a több kimenetben szereplő ***szorzatok (ÉS kapuk) újrafelhasználása:***

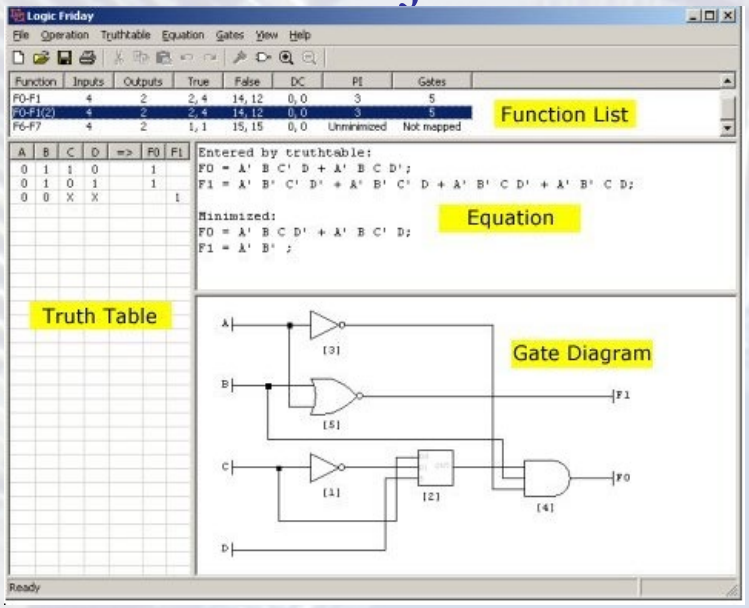
- A_OUT = A + AB + AC = A
- B_OUT = B + AB + AC = B + **AC**
- C_OUT = C + AB + AC = C + **AB**
- D_OUT = D + **AB** +**AC**
- Az **AC**-t és **AB**-t előállító ÉS kaput 2 kimenenél is újra felhasználjuk.

4 db 2 bemenetű és 1 db 3 bemenetű kaput használunk



Logic Friday

- **Oktatási célú demonstrációs eszköz, érdeklődőknek**
  - Általános logikák tervezésére, Max. 16 be-, 16 kimenet
  - Független, vagy több kimenetű minimalizálás
  - Gyors futásidő vagy pontos minimalizálás opció
  - Specifikáció: Egyenlet, igazságtábla, kapcsolási rajz
  - Espresso és MIS II algoritmusokat használja
  - Elérhető az interneten

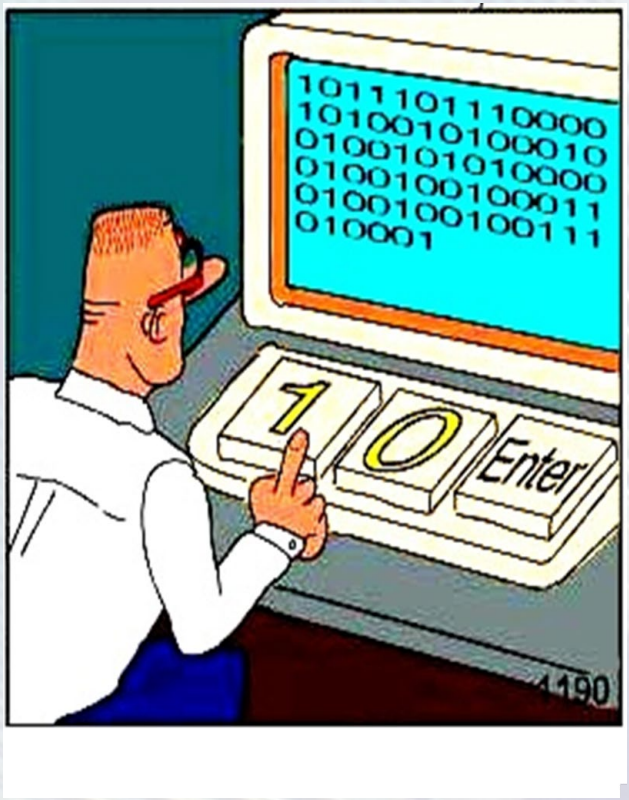


Összefoglalás

- **Az eddig látott módszerekkel az alapkapukból építkezve tetszőleges („random”) kombinációs logikai áramköröket felépíthetünk (elsősorban kis- és közepes komplexitásig).**
- Kiindulunk az adott formában megadott **specifikáció**-ból, értelmezzük, elvégezzük a lehetséges **minimalizáció**s műveleteket, (itt figyelembe vehetünk **optimalizáció**s célokat, azaz sebesség, alkatrész) és **kapuszinten realizáljuk** a függvényt
- A legtöbb esetben a Verilog specifikáció alapján a minimalizálást/optimalizálást a fejlesztőrendszerre hagyjuk.

Kérdések, észrevételek

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



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 megoldási lehetőséget.***
- A ***logikai alapelemek*** (kapuk stb.) ***helyett*** akár azoknál sokkal komplexebb ***funkcionális elemekből*** (modulokból) ***építkezünk***.

Funkcionális egységek

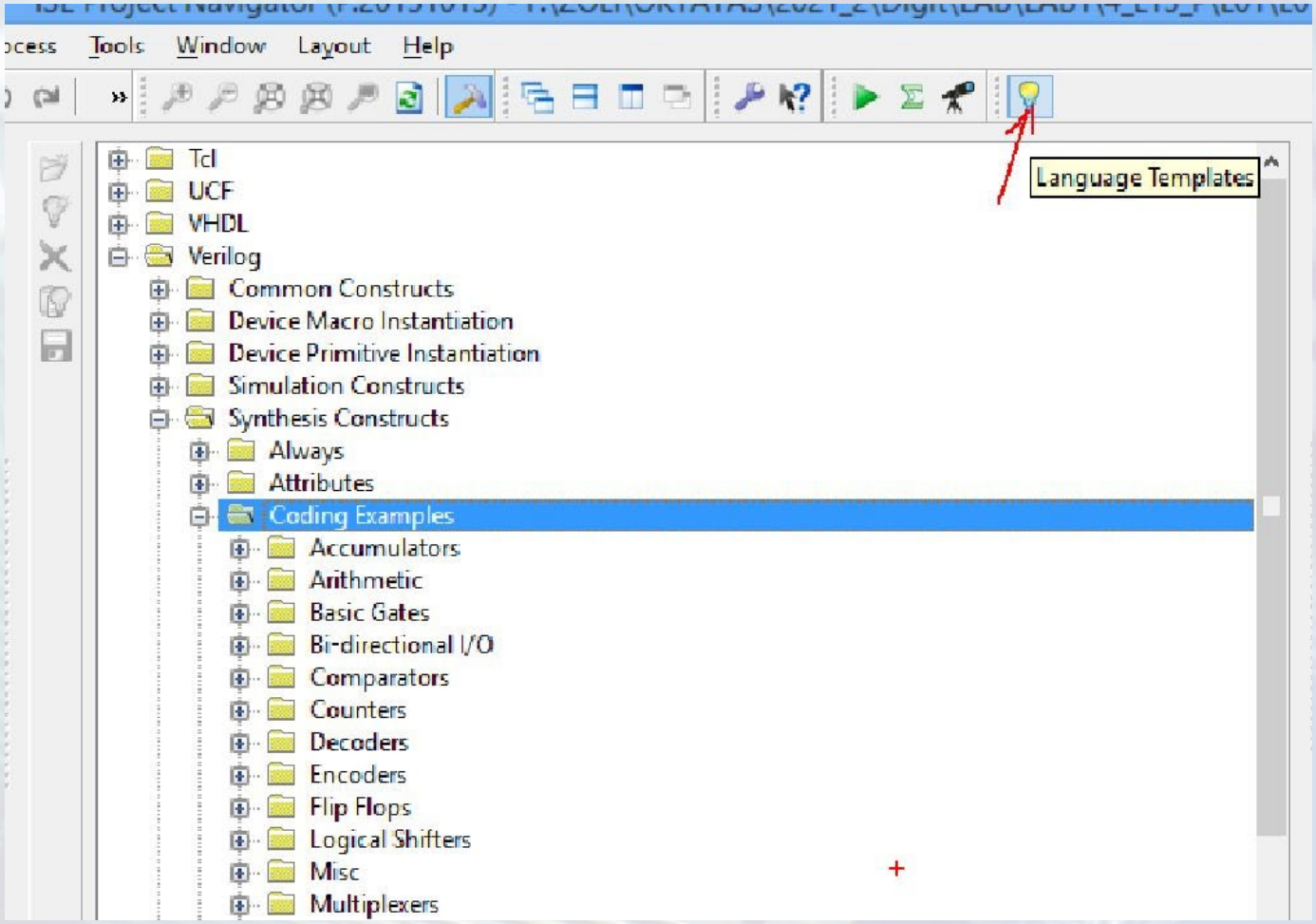
- **Általános célú kombinációs logikai funkciók**
  - Logikai funkciók: dekóder (**DEC**), enkóder (ENC), prioritás enkóder (PRI), **multiplexer (MUX)**, demultiplexer (**DEMUX**)
  - Aritmetikai funkciók: öszeadó (**ADD**), **kivonó (SUB)**, **komparátor (CMP)**, paritás generátor (PAR)
  - Egyéb, esetleg szükséges funkciók

Funkcionális egységek

- Hagyományos technológiában (TTL MSI IC-k) ***minden funkcióra önálló elem/alkatrész létezett***, különböző méretekben, tokozásban, lábszámban.
- FPGA-ban, könyvtári alapú, modulszintű építkezésnél paraméterezhető alapfunkciók használhatók, az aktuális terv igényei szerint beállítva.
- **HDL alapú tervezésnél a tanult *tervezési mintákat használunk*.** Ilyeneket a CAD rendszerek is tartalmaznak.
  Pl. ISE Language templates/Verilog/Synthesys Constructs/**Coding Examples**

Funkcionális egységek

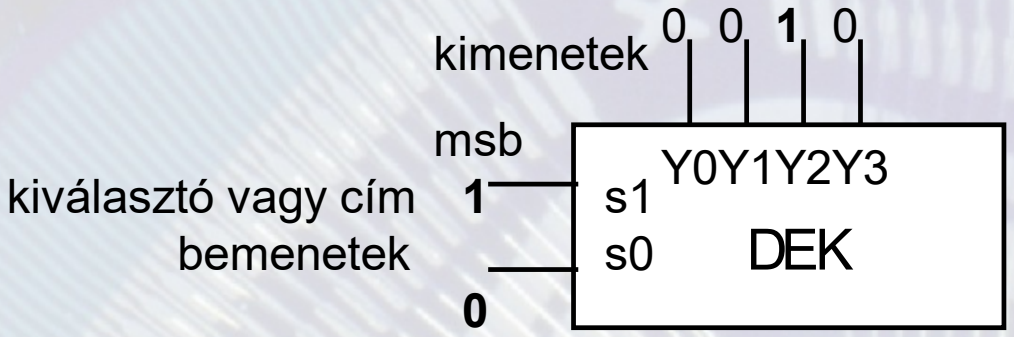
Az ISE Verilog tervezési mintái itt érhetők el:



• 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.**

s1	s0	Y0	Y1	Y2	Y3
0	0	**1**	0	0	0
0	1	0	**1**	0	0
1	0	0	0	**1**	0
1	1	0	0	0	**1**



DEK 2:4

Y0 = /s1/s0 Y1=/s1s0 Y2=s1/s0 Y3=s1s0

- A dekóder a binárisan értelmezett bemeneti jelekből az általános ***logikai függvény összes mintermjét előállítja. Minden kimenethez 1 minterm tartozik. Ezeket*** **konstans komparátoroknak *is tekinthetjük.***

Funkcionális egységek: Dekóder

KIM = f (BEM)

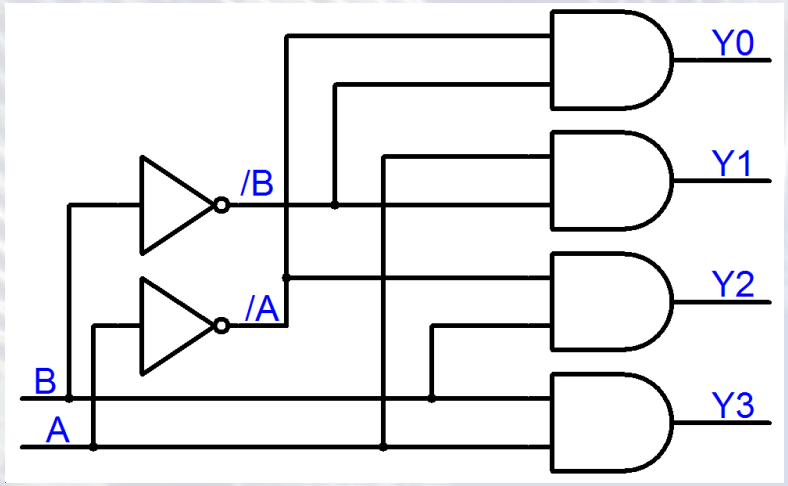
- Ha a **BEMENET** egy ***n bites binárisan kódolt bemenet***, akkor a **KIMENET** egy ***2^n méretű, 1-a-2^n -ből kódolású bitvektor***, ezért hívjuk **DE-KÓDOLÁS**NAK

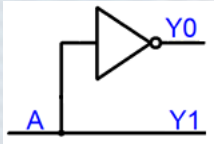
- A tipikus méretek (bementek száma: kimenet száma): 1:2, 2:4, 3:8, 4:16. (de lehet 4:10 is)

- Lehet nagyobb méretben is, de esetleg érdemesebb az egyszintű dekóder helyett a többszintű sor-oszlop vagy hierarchikus fa struktúrájú felépítést használni.

Funkcionális egységek: Dekóder

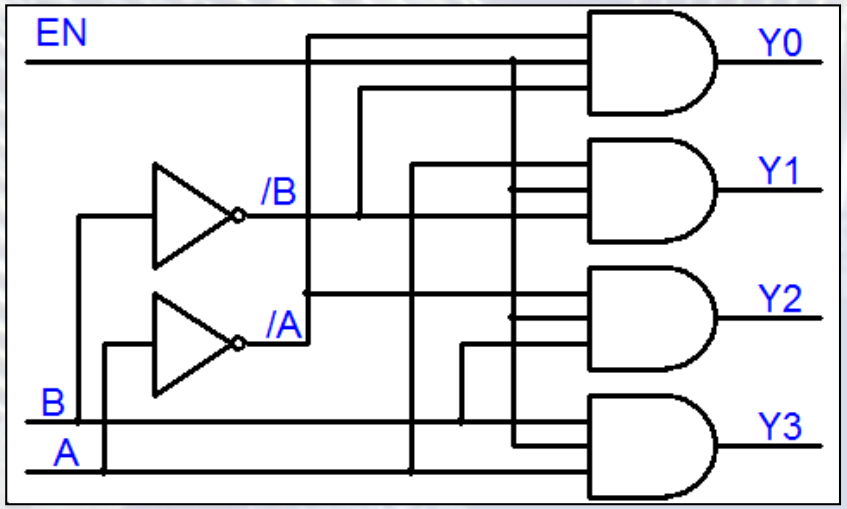
- **Egyszintű, közvetlen dekóderek felépítése**
- A ***dekóder kimeneti bitjei egyenként megfelelnek az n bemenetű általános logikai függvények egy-egy mintermjének***, azaz minden változó szerepel benne, normál (ponált) vagy negált értelemben.
- **Minden kimenet egy n bemenetű ÉS kapu kimenete**
- **Pl. 2 bemenetre:**
  - n bitre n INV és 2^n db n bemenetű ÉS
  - (Egy bitre csak 1 INV)





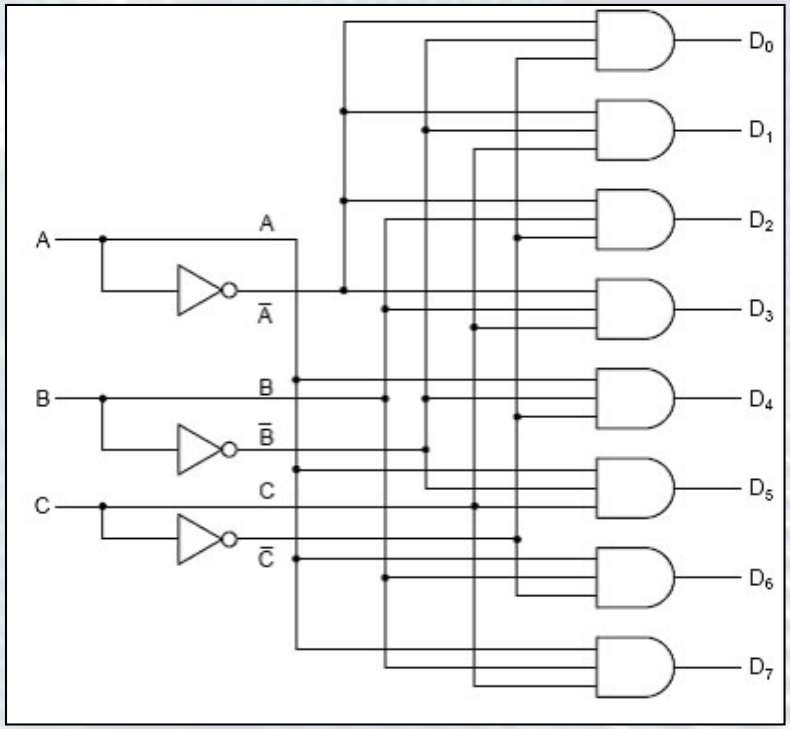
Funkcionális egységek: Dekóder

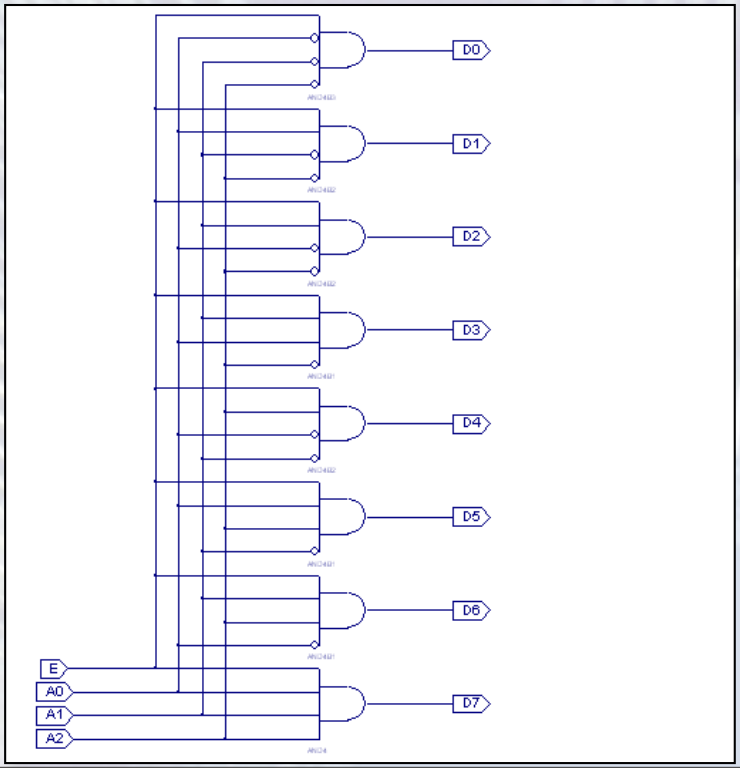
- ***Az előbbi dekóder egyik kimenete mindig aktív.*** (A bemeneten mindig van egy kombináció-)
- Ezért érdemes bevezetni egy ezt felülbíráló jelet: **ENGEDÉLYEZÉS.**
  - ***Ha EN = 0, minden kimenet inaktív*** (Y0=Y1..=Yn=0)
  - Így könnyen tiltható/engedélyezhető a kimenetek által vezérelt egységek működése
  - Egyébként is ez a DEK legfontosabb funkciója
  - Könnyű a többszintű bővíthetőség



Funkcionális egységek: Dekóder

- **Egyszintű, közvetlen dekóderek felépítése**
  - Ábrázolása inverterekkel vagy a bemeneteken jelölve a negált aktív szintet. (rajzolhatóság, olvashatóság)

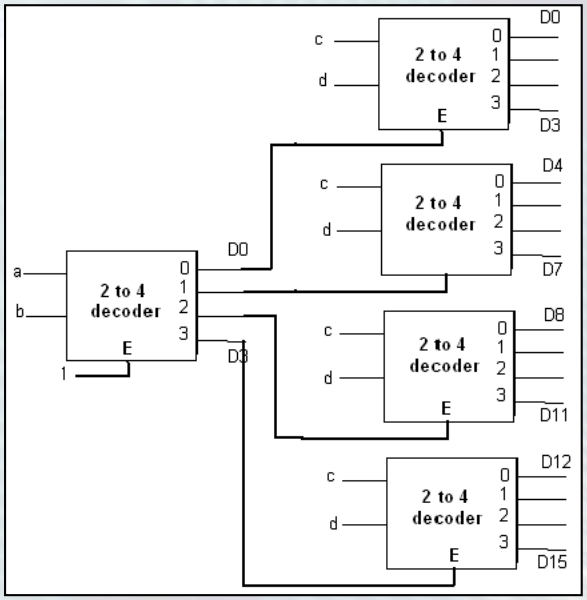




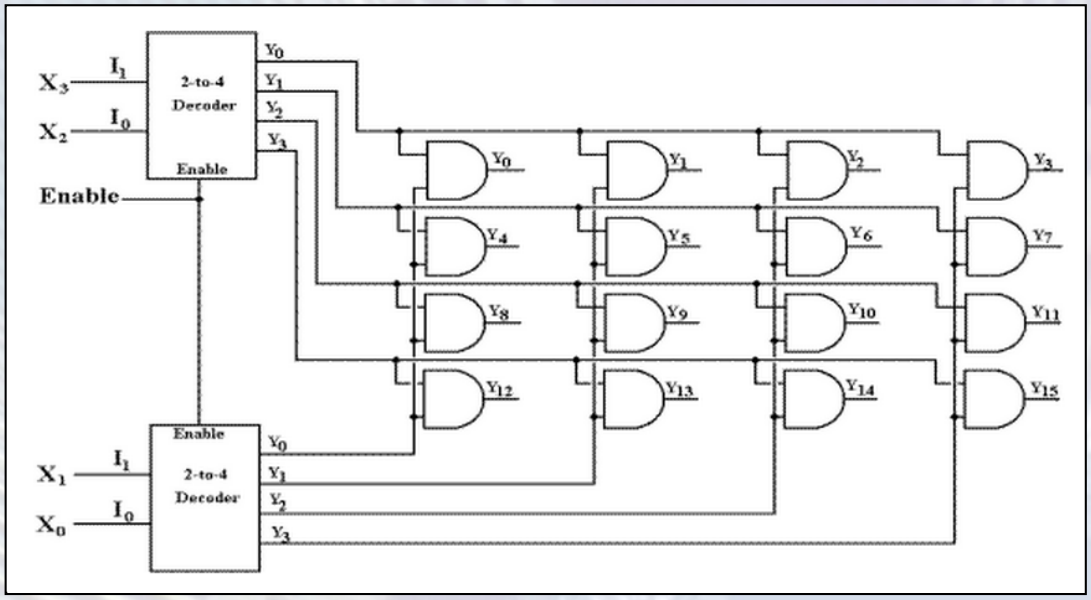
Funkcionális egységek: Dekóder

- **Nagyméretű dekóderek (moderált méretben mutatva)**
  - 4:16 dekóder, engedélyező bemenettel, közvetlenül
  - 2:4 méretű, engedélyezhető dekóderből felépítve

HIERARCHIKUSAN

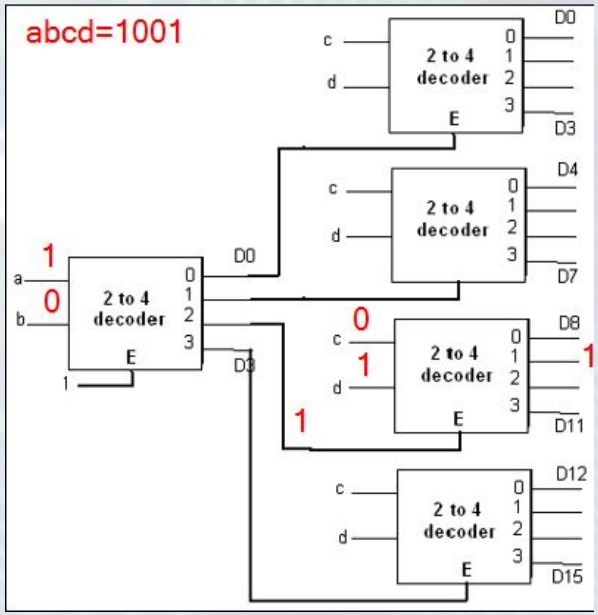


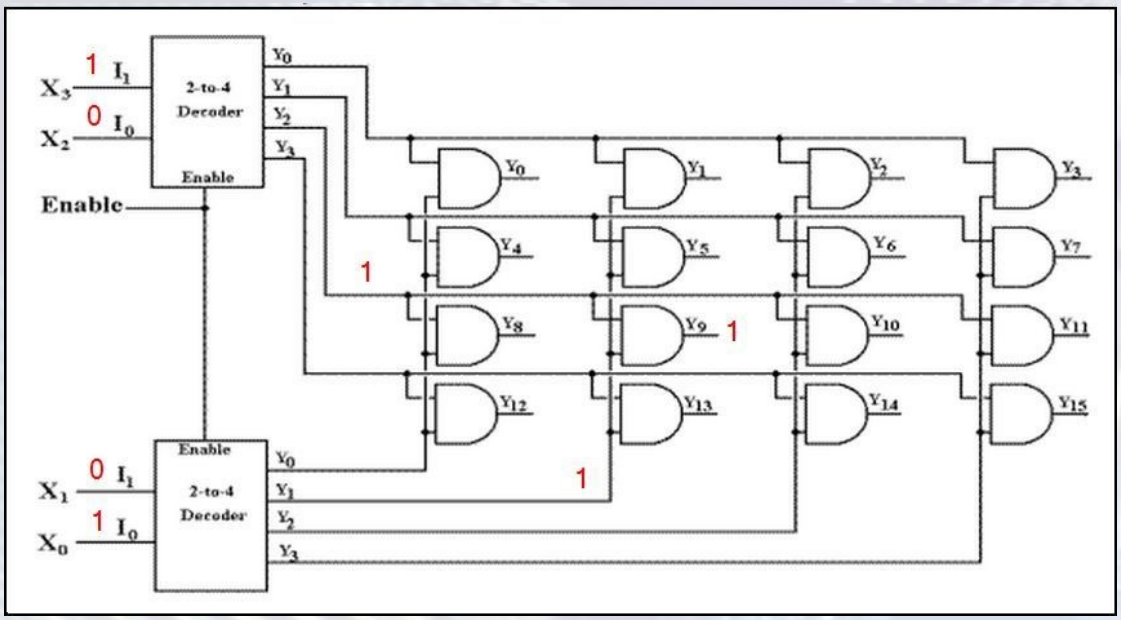
2D, SOR-OSZLOP módban



Funkcionális egységek: Dekóder

Működésük:





Funkcionális egységek: Dekóder

Verilog HDL kódolási minták 2:4 dekóderekre

A mintermek megadásával:

```
assign out[0] = en &  ~sel[1] & ~sel[0];    // sel = 0
assign out[1] = en &  ~sel[1] &  sel[0];    // sel = 1
assign out[2] = en &   sel[1] & ~sel[0];    // sel = 2
assign out[3] = en &   sel[1] &  sel[0];    // sel = 3
```

Komparálás reláció (konstans komparátor) használatával még egyszerűbb:

```
assign out[0] = en & (sel == 2'b00);  //~sel[1] & ~sel[0] helyett (sel == 3'h0)
assign out[1] = en & (sel == 2'b01);
assign out[2] = en & (sel == 2'b10);
assign out[3] = en & (sel == 2'b11);
```

A Verilog konkatenálás { } és shiftelés << műveleteivel nagyon tömör leírás adható:

```
assign out = {3'b000, en} << sel;  (Nem tananyag)
```

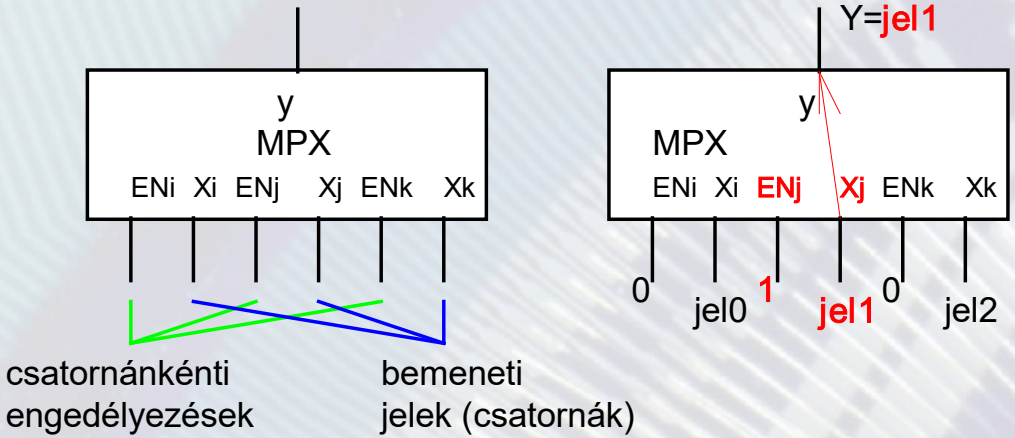
Funkcionális egységek: Dekóder

Viselkedési leírás always blokkal:

```
reg [3:0] dec;
always @ (*)
begin
    if (en)
        case (sel)
        2'b00: dec = 4'b0001;
        2'b01: dec = 4'b0010;
        2'b10: dec = 4'b0100;
        2'b11: dec = 4'b1000;
        endcase
    else
        dec = 4'b0000;
end
assign out = dec;
```

Funkcionális egységek: Multiplexer

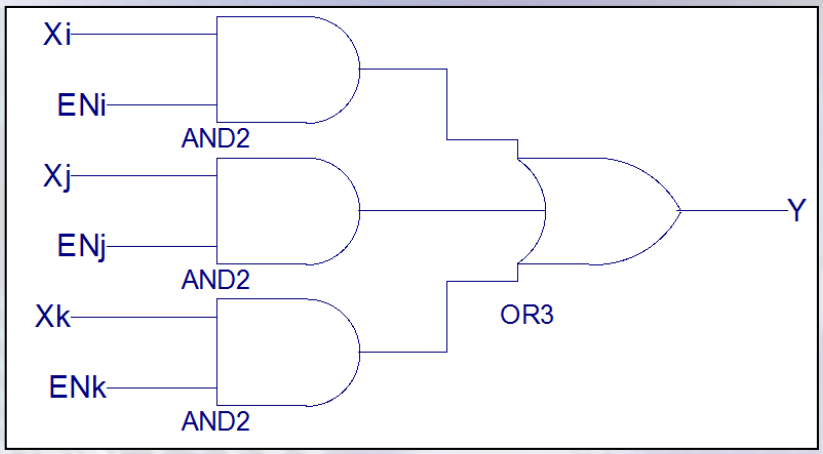
- Az egyik legfontosabb funkcionális elem
- Alapvető feladata az adatforrás, ***adatút választás*** megvalósítása
- **Multiplexer csatornánkénti engedélyezéssel**



- Az i-edik bemenet (Xi, i-edik csatorna) Eni (Seli) jele engedélyezi, hogy Xi bemenet megjelenjen a kimeneten. Az En (Sel) jelek közül csak 1 db lehet aktív.

Funkcionális egységek: Multiplexer

- Megvalósitás SOP alakban:
- Xi, Xj, Xk az adatbemenetek
- ENi, ENj, ENk engedélyezés (kiválasztás)

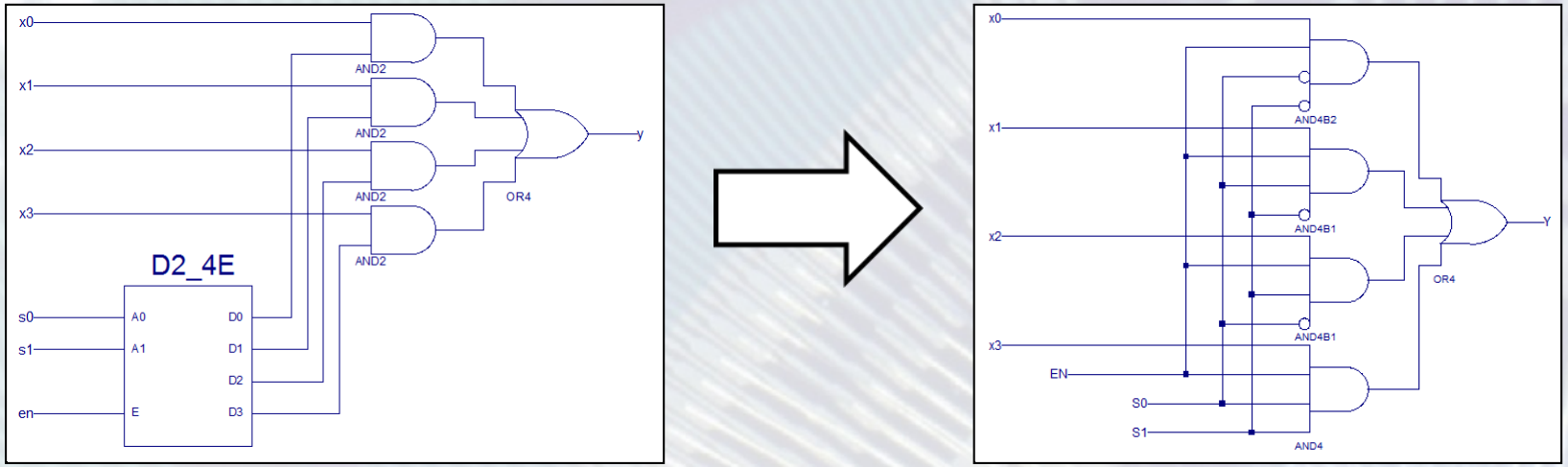


Y = Xi*ENi + Xj*ENj + Xk*ENk

- Tetszőleges számú bemenettel bővíthető
- **Helyes működés feltétele:** az ***EN bitvektor legyen 1-az-N-ből kódolású*** (egyszerre csak egy db EN lehet 1) → Egy dekóder kimenete állítja elő

Funkcionális egységek: Multiplexer

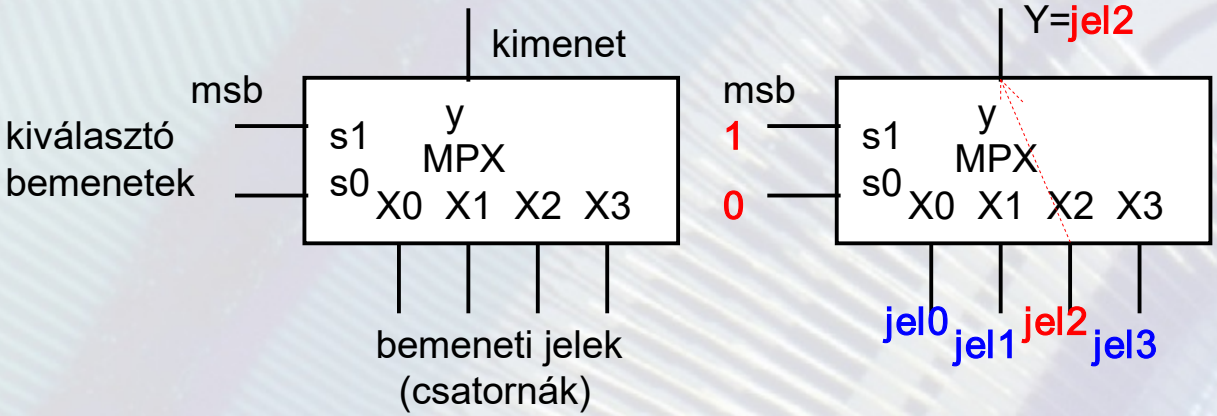
- **A MUX tehát lényegében egy DEK+AND-OR**
- Azonban a MUX jelentősége miatt a beépített DEK funkciót külön nem szoktuk azonosítani



- A multiplexer ***engedélyező bemenettel is rendelkezhet***. Ha en=1, akkor a kimeneten a kiválasztott bemenet jelenik meg, egyébként 0.

Funkcionális egységek: Multiplexer

- ***Beépített dekóderrel (kódolt kiválasztó bemenetekkel) rendelkező multiplexer (hagyományos megvalósítás) esetén a kiválasztó bemenetekre a kiválasztandó csatorna bináris sorszámát kell ráadni.***



- Jellemző méretek: 2:1, 4:1, 8:1, 16:1

Funkcionális egységek: Multiplexer

A multiplexer Verilog specifikációja sokféle lehet

Klaszikus kapu szintű leírás

```
assign out = en & ((dinp[0] & ~sel[1] & ~sel[0])
                 | (dinp[1] & ~sel[1] &  sel[0])
                 | (dinp[2] &   sel[1] & ~sel[0])
                 | (dinp[3]  &  sel[1] &  sel[0]));
```

Komparálás reláció használatával

```
assign out = en & ((dinp[0]  & (sel == 2'b00))
                 | (dinp[1] & (sel == 2'b01))
                 | (dinp[2] & (sel == 2'b10))
                 | (dinp[3] & (sel == 2'b11)));
```

Konkatenálás és shiftelés: dekóder, utána bitenkénti ÉS az adatvektorral, majd bitek közötti VAGY a bitredukciós | operátorral

```
assign out = |(({3'b000, en} << sel) & dinp); (Nem tananyag)
```

Funkcionális egységek: Multiplexer

Viselkedési leírás always blokkal:

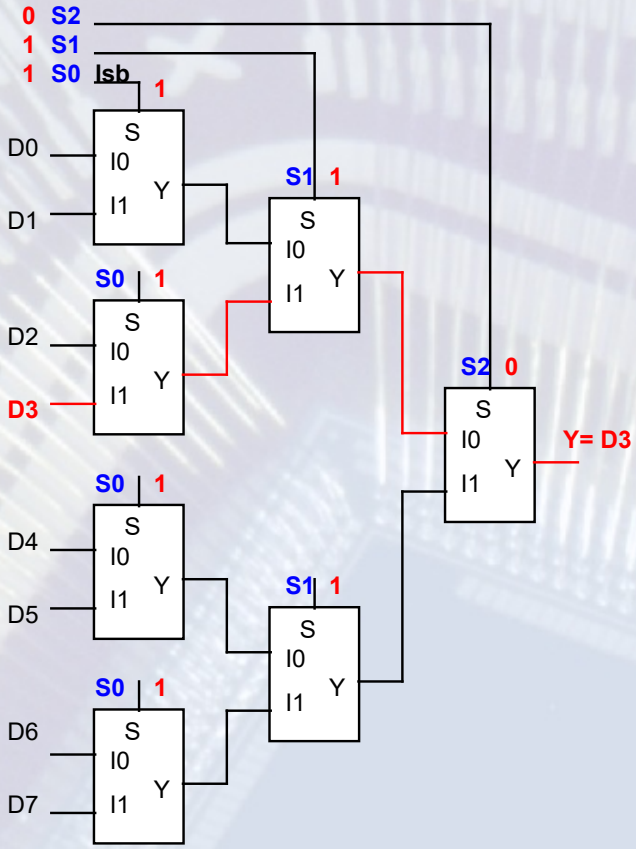
```
reg mpx;
always @ (*)
begin
    if (en)
      case (sel)
      2’b00:  mpx = dinp[0];
      2’b01:  mpx = dinp[1];
      2’b10:  mpx = dinp[2];
      2’b11:  mpx = dinp[3];
      endcase
    else
          mpx = 1'b0;      // Nincs engedélyezve
end
assign out = mpx;
```

Funkcionális egységek: Multiplexer

- **Multiplexer bővítése (hogyan csinálunk kevesebb bemenet számúakból nagyobbat)**

  - A MUX is felépíthető hierarchikusan, fa struktúrában

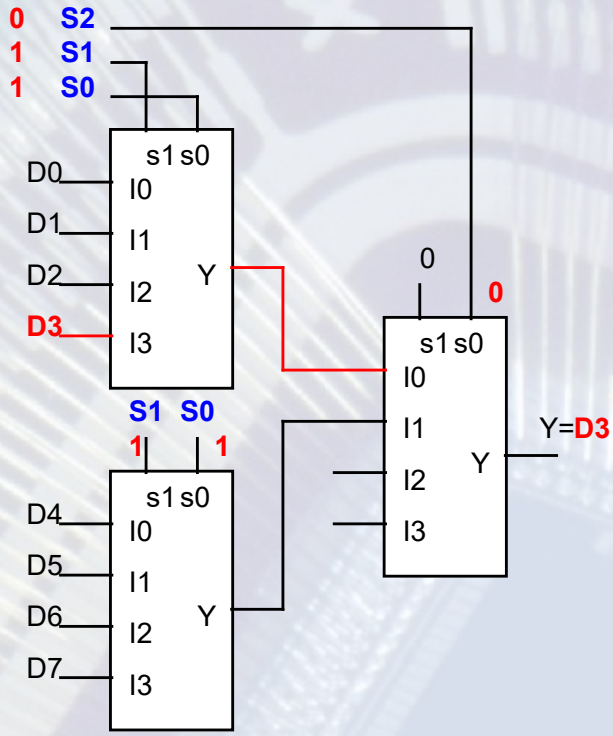
  - LSB-vel kezdve első lépés a páros-páratlan bitek közti választás, majd így haladunk tovább, mindig az aktuális szomszédok között választva



• Funkcionális egységek: Multiplexer

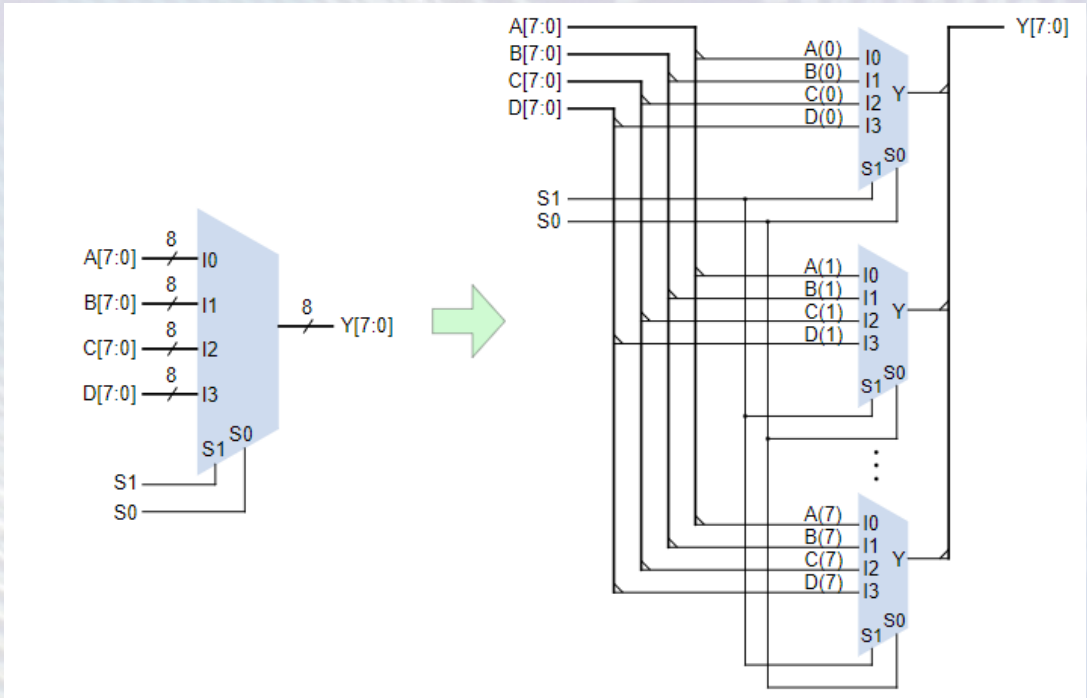
A fa felépíthető 4:1, 8:1vagy nagyobb lépésekben is, ekkor kevesebb szint kell, de több bemenetűek a kapuk.

Itt 4:1-es MPX-ekből építünk 8:1-est. Most 2 szint elég. A második szinten a 4:1-es MPX-ert 2:1-esként használ-juk. (Az s1 bemenetére 0-át kötve, s0 az I0, I1 bemenetei közül választ, I2, I3 nem használt.)



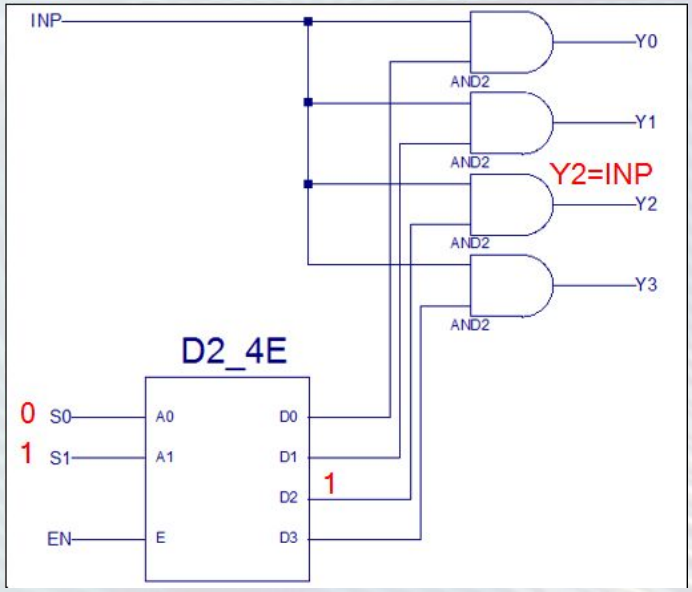
Funkcionális egységek: BUS MUX

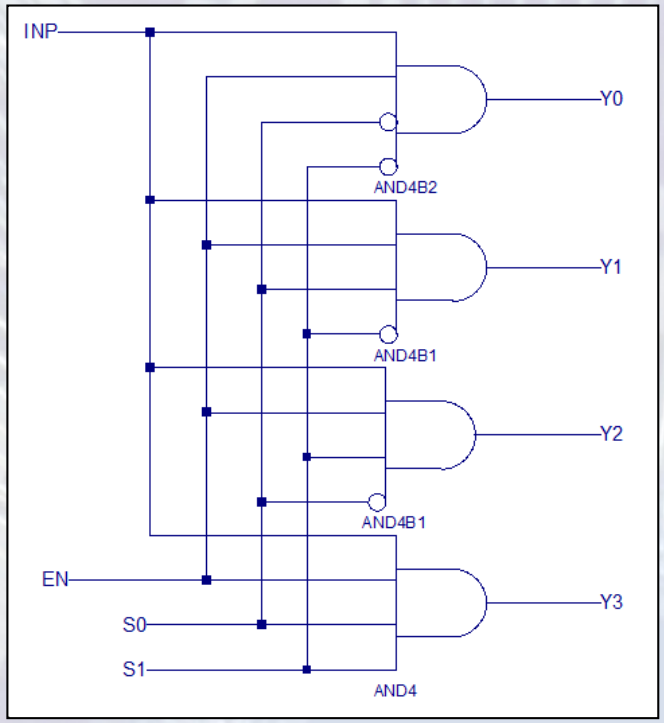
- **A MUX felépíthető bitvektorokra is, ez a BUS MUX**
- A bemeneti/kimeneti adatok bitmérete azonos
- A kiválasztó jelek száma s =⌈$\log_2$ Bem._csat._száma⌉
- Az ábrán az adatméret 8 bit, a csatornák száma 4, így s=2
- A BUS MUX mind a 8 bitre egy-egy 4 adat bemenetű normál MUX, közösített s1,s0 kiválasztó jelekkel



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***)

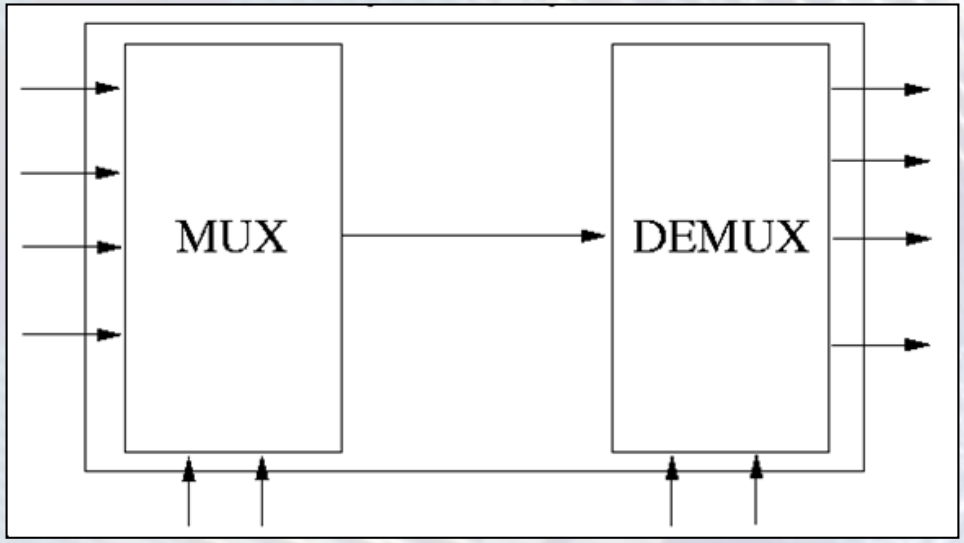




- Ez is DEK + AND adatkapu felépítésű
- Szokásos méretei 1:2, 1:4, 1:8

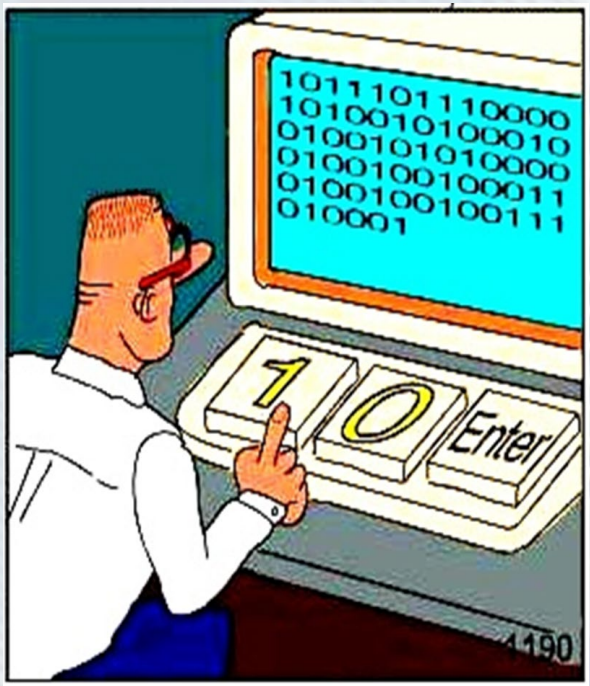
Funkcionális egységek: DEMUX

- **A MUX-DEMUX egységek használata**
  - Több adat átvitele egyetlen vonalon
    - Forrásoldalon MUX, vételi oldalon DEMUX
    - Az átviteli vonal csak egyetlen vezeték, de sokkal nagyobb sávszélességű
    - Időosztásos adatátvitel, a két oldalon azonosan ütemezett kiválasztó jelekkel



Kérdések, észrevételek

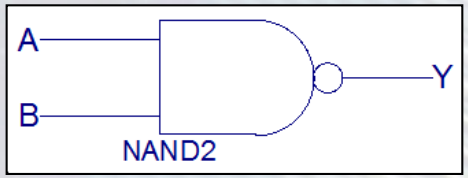
- **A kérdéseket a benes@mit.bme.hu címre várjuk de csak ... @edu.bme.hu címről, tárgy: DIGIT**

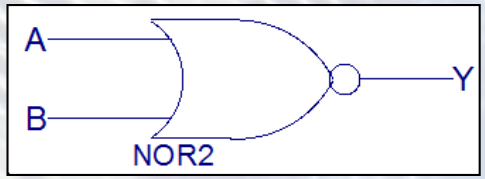


Univerzális logikai elemek

- **A Boole algebra definíciója alapján a 3 alapművelettel (AND, OR, INV) minden logikai függvény előállítható**

- **<u>Univerzális logikai elem</u>:**
  Egyetlen alapelemmel minden logikai függvény előállítható.
- A NAND, NOR kapuk ilyenek

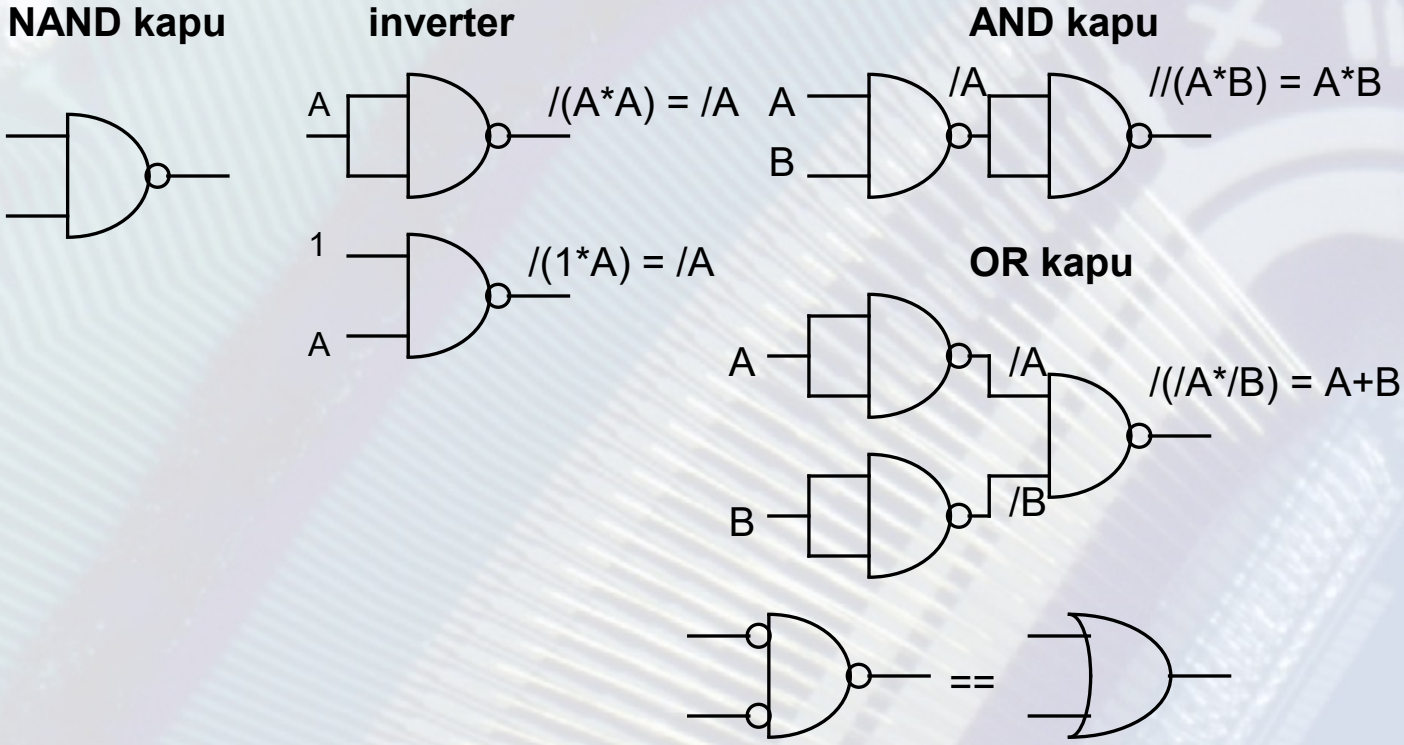




Univerzális logikai elemek

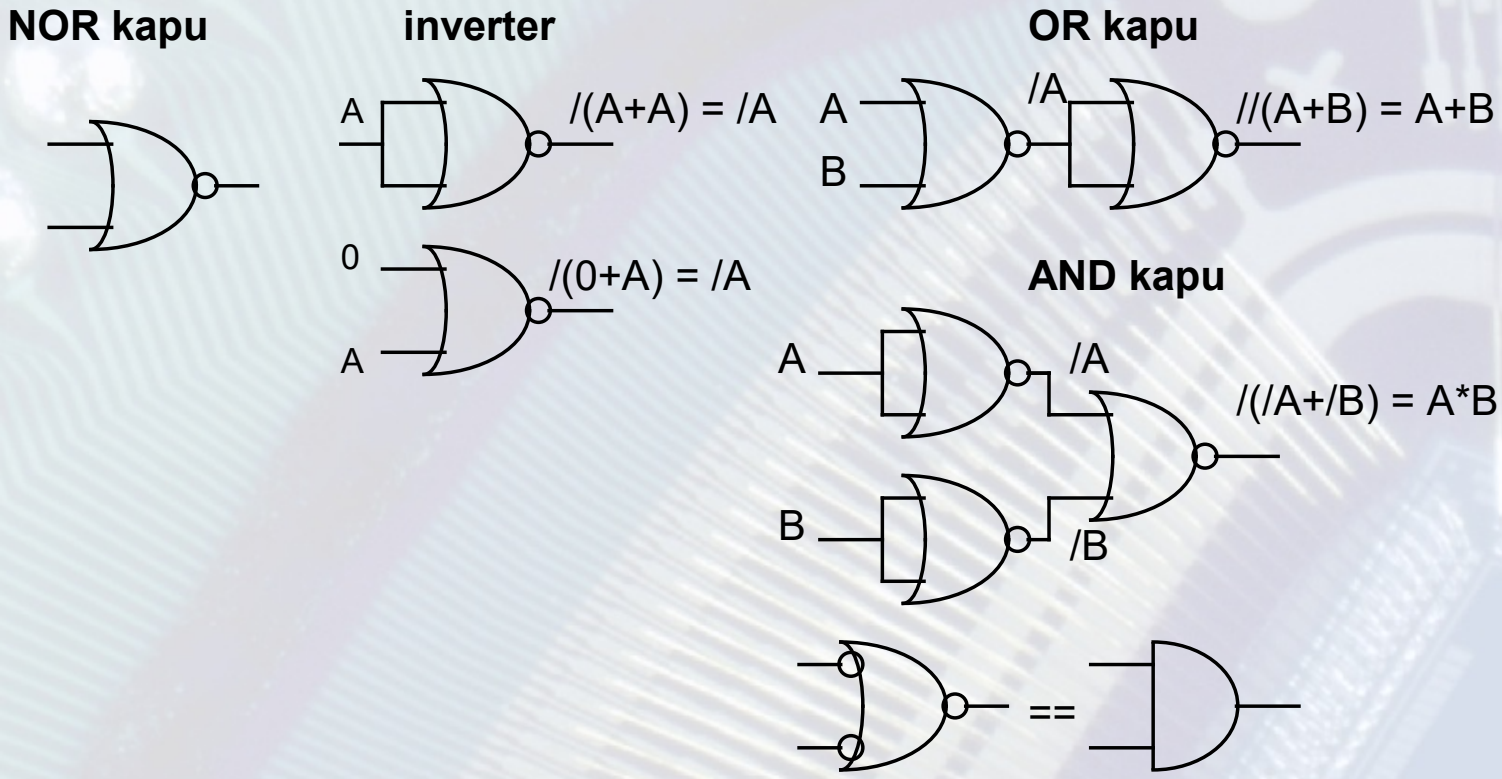
Bizonyítás: Ha a három alapfüggvény realizálható velük, akkor beláttuk.

- **Bizonyítás NAND kapura**



Univerzális logikai elemek

- **Bizonyítás NOR kapura**



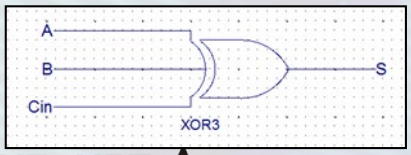
Univerzális logikai elemek

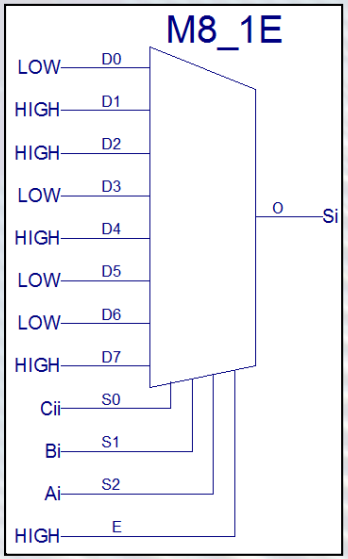
- **Tehát a legegyszerűbb univerzális elemek:**
  - A NAND és NOR kapuk ezeket homogén hálózatoknak nevezzük
- **Lehetséges más univerzális elemkészletet is használni:**
  - Multiplexer
  - Memória táblázat (LUT, Look Up Table)
  - Ezek az egységek közvetlenül többváltozós függvényeket képesek megvalósítani, ezért a tervezés egyszerűbb, kapu szintű minimalizálás, ill. optimalizálás szinte nem is szükséges

A Multiplexer, mint UNIV elem

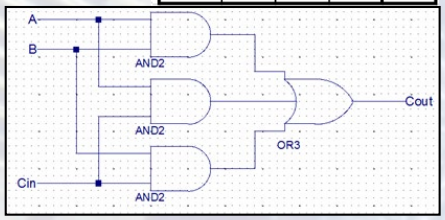
- ***Multiplexer*** = Univerzális logika
- ***Bemeneti változók:*** A MUX kiválasztó (SEL) bemeneteire kötött változók, LSB-re az igazságtábla jobb szélső változóját kell kötni.
- ***Függvény megadása:*** Az igazságtábla i-edik sorának függvény értékét konstans jelként a MUX i-edik adatbemenetére kapcsoljuk
  - A korábbi F1 és F2 függvényeink (1 bites FADD Si és Co)

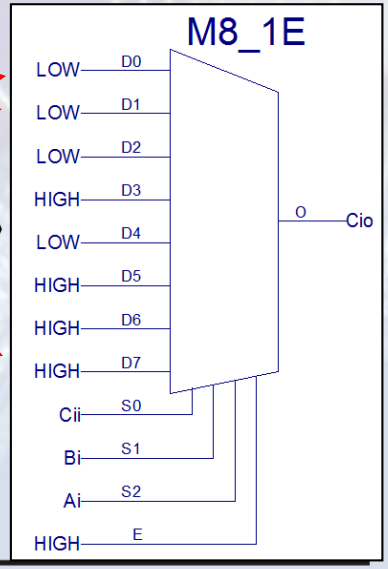
BEMENETEK				KIM
INDX	A	B	C	F1
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1





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





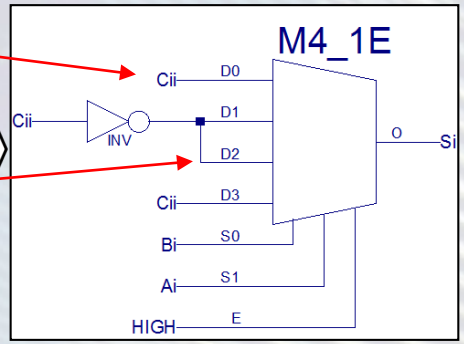
A Multiplexer, mint UNIV elem

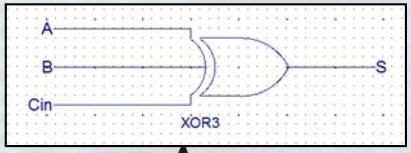
- **Az előbbi MUX mérete felére csökkenthető, ha:**
- **Bemeneti változók:** A kiválasztó (SEL) bemenetére mennek egy kivétellel (MUX méret felére csökken)
- Ez utóbbi változó (normál és/vagy negált értékei) az adatbemenetekre kapcsolhatók
- **Függvény megadása:** Az igazságtábla kimeneti jel oszlopának értékeit a SEL-el kiválasztott sorszámú bemeneteken adjuk meg (0 vagy 1 konstansok és a külön változó ponált és/vagy negált értéke).

A Multiplexer, mint UNIV elem

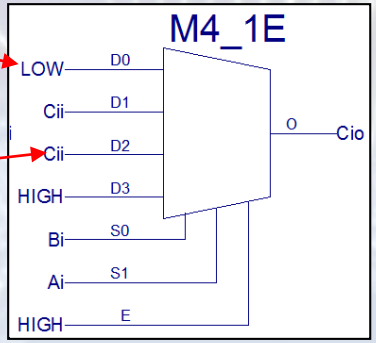
- **Az előző példa alapján (1 bites FADD Si és Co)**
- **A módszert Co-n magyarázzuk (Si hasonló)**
- **AB=00 sorokban F2 = 0 ezért D0 = 0**
- **AB=01 sorokban F2 = C ezért D1 = C**
- **AB=10 sorokban F2 = C ezért D2 = C**
- **AB =11 sorokban F2 = 1 ezért D3 = 1**

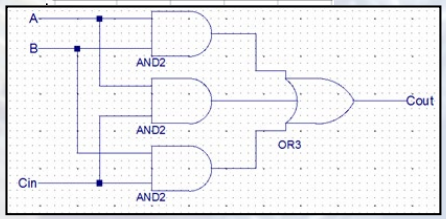
BEMENETEK				KIM
INDX	A	B	C	F1
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1





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



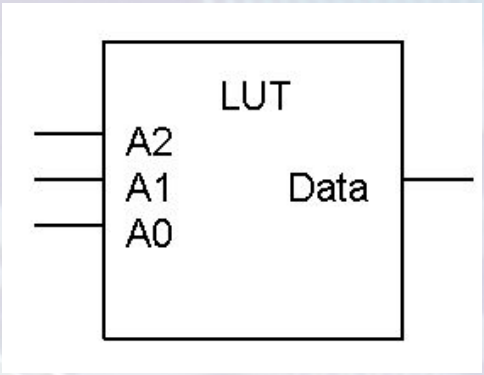


A Multiplexer, mint UNIV elem

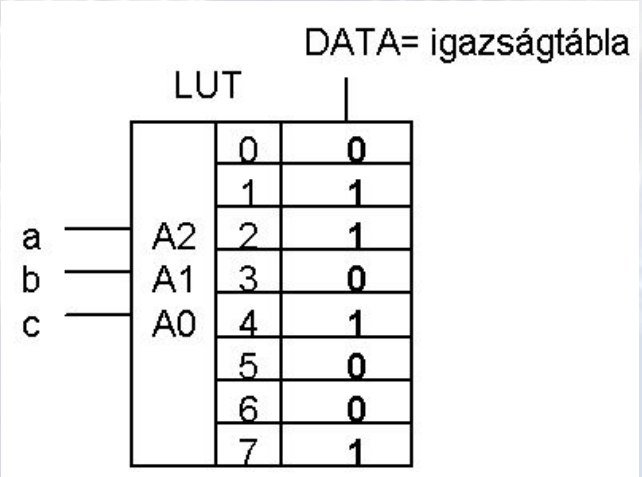
- A függvény DNF alakjából is könnyen megtervezhető:
- Co = /abc+a/bc+ab/c +abc
- s1-et a-ra, s2-őt b-re kötjük, így a és b választják ki a MPX bemeneteit
- A szorzatoknál az s1s0-ra kötött ab változók alá írjuk azt a bemeneti kombinációt amelynél a szorzat ab változós része 1 értékű
- Co = /abc+a/bc+ab/c +abc
  s1s0: 01 10 11 11
- Ahol 2-szer szerepe ugyanaz az érték, abból az ab-t kiemeljük és egyszerűsítünk:
  Co = /abc+a/bc+ab*1
  s1s0: 01 10 11
- A MUX i-edik bemenetére bekötjük az i-edik szám feletti szorzat harmadik változóját (itt c) az aktuális formájában (ponálva vagy negálva) vagy 1-et (ha a változó kiesett). A MUX megmaradó bemeneteire 0-át kötünk. I0:0, I1:c, I2:c, I3:1

A memória táblázat, mint UNIV elem

- LUT memória táblázat (FPGA logikai elem)
- A címbitekkel (A[n-1:0]) megcímzett rekeszének tartalmát adja ki az adat (Data) kimenetén.
- Pl:8 rekesszel rendelkező (3 címbit) LUT



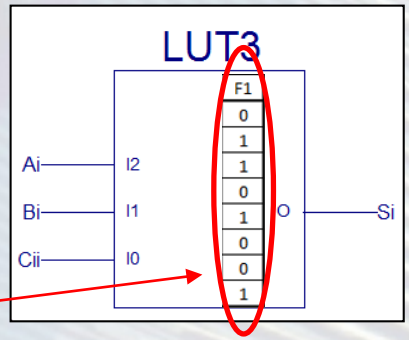
- A LUT-ot úgy használhatjuk univerzális kombinációs hálózatként, hogy a változókat a cím bemenetekre kötjük és tartalomként az igazságtáblázatot töltjük bele. Figyelni kell a bemeneti áltozók sorendjére!



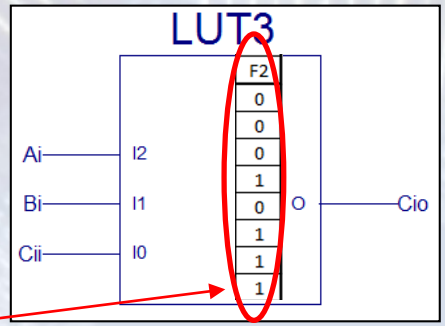
A memória táblázat, mint UNIV elem

- **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,B,C-> I2,I1,I0)**
  - A korábbi F1 és F2 függvényeink (FADD Si és 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

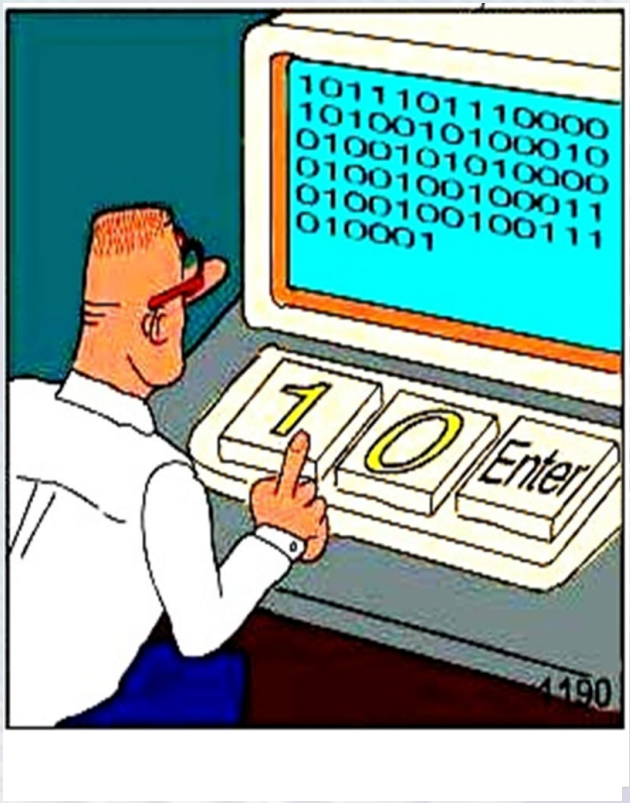


BEMENETEK				KIM
INDX	A	B	C	F2
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



Kérdések, észrevételek

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



Verilog

Vreilog 24-37 diák

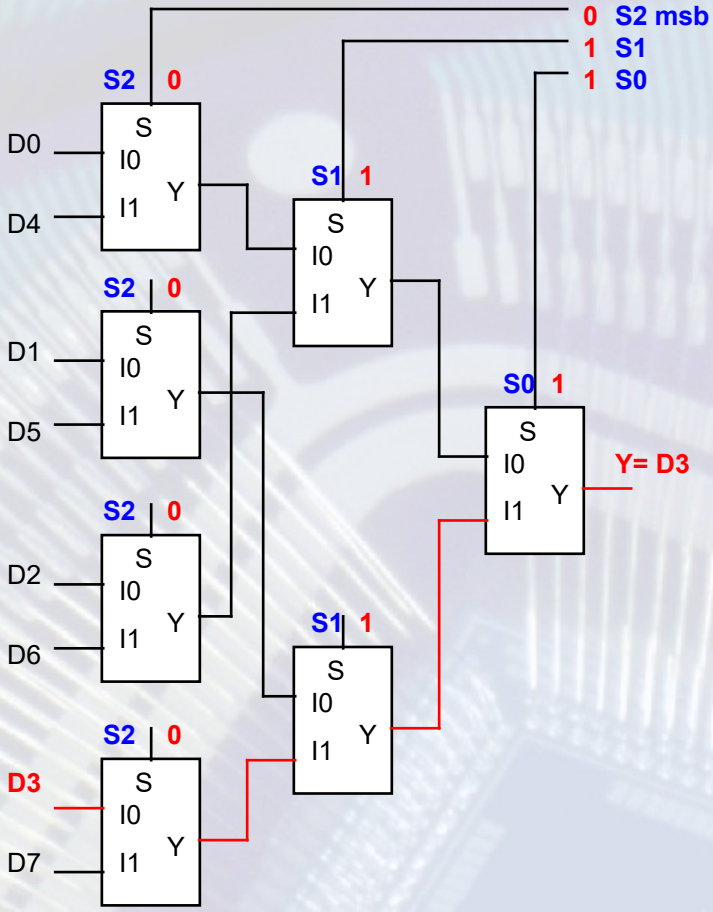
Digitális technika 3. EA vége

Az alábbi anyag nem szerepel a vizsgakövetelményekben

Funkcionális egységek: Multiplexer

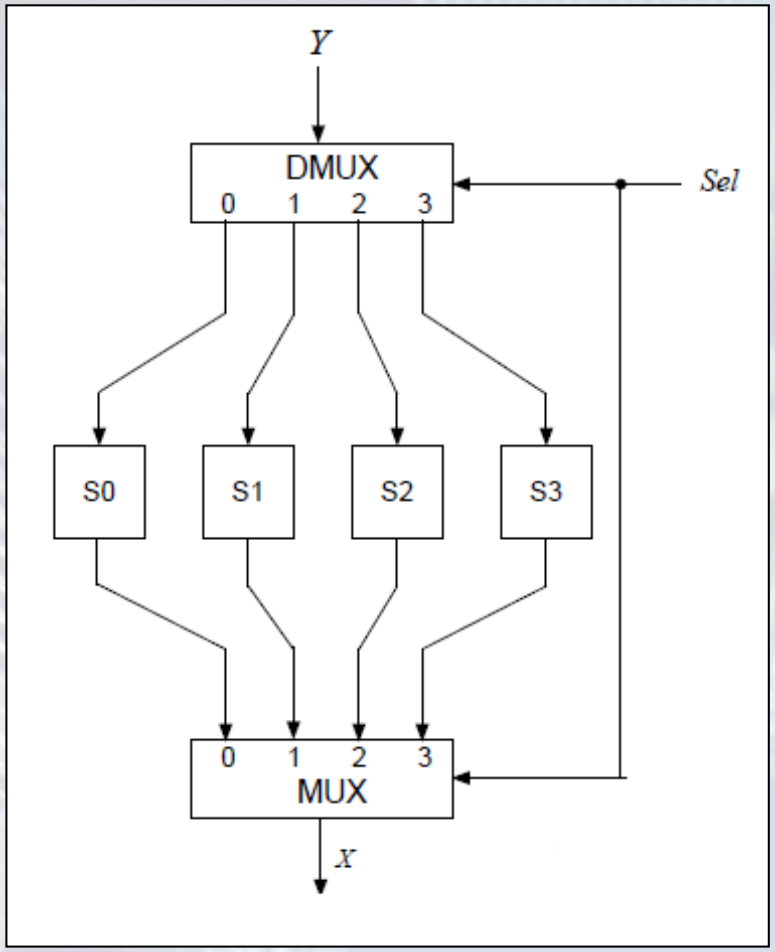
Multiplexerek bővítése

MSB-vel kezdve első lépés az adatbemenetek felezése (első-második fele), majd tovább a maradék felezése (negyedelés) és í.t. az utolsóig.



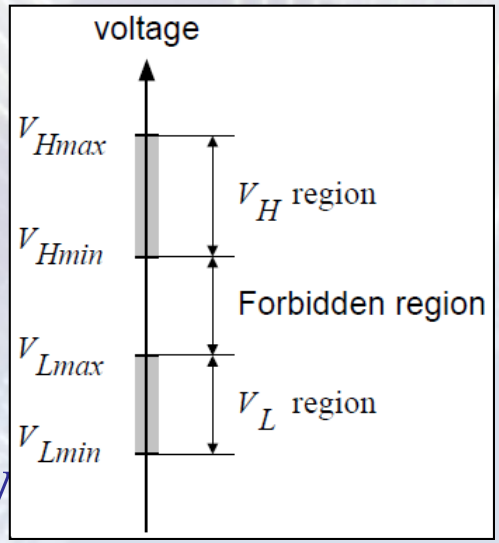
Funkcionális egységek: DEMUX

- **A DEMUX-MUX egységek használata**
  - Lassú adatfeldolgozók párhuzamosítása
    - Forrásoldalon 1:4 DEMUX,
    - Feladatok szétosztása, S0, S1, S2, S3 egységek között
    - Pl. A 100MHz mintavételezésű adatokat ($\Delta t = 10ns$) 4 db 40ns végrehajtási idejű egységgel lehet kiszolgálni
    - A kimeneten minden 10ns-ban van egy új eredmény valamelyik egységtől
    - Vételi oldalon 4:1 MUX, az elkészült feladatok összegyűjtésére



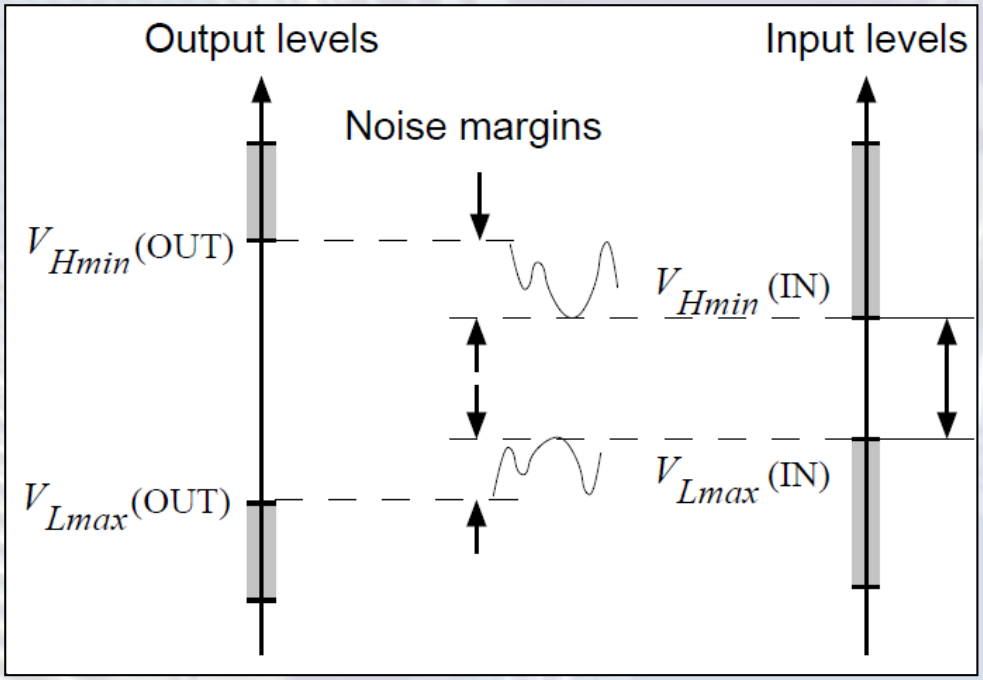
Digitális technika áramköri alapok

- **Boole algebra, bináris aritmetika: két érték**
- **Logikai értelmezés: 0, 1, (L, H, LOW, HIGH)**
- **Fizikai reprezentáció: Feszültségszint GND = 0V, a $+V_{cc}$ lehet +5V, +3,3V, +2,5V, +1,8V, +1,2V, …**
  - **Energia fogyasztás $P_D = C_L * V_{cc}^2 * f_{clk}$,** azaz V_{cc} csökkentése jelentősen javítja a fogyasztást.
  - Az érvényes logikai feszültség szintek határai pl. $+V_{cc}$ =3,3V esetén
  - V_{Hmax} = 3,3V, V_{Hmin} = 2,0V (60%)
  - V_{Lmax} = 0,8V, V_{Lmin} = 0,0V (25%)
  - A kettő között a tiltott, hibás tartomány



Digitális technika áramköri alapok

- **Megbízható működés érdekében: Zavarjelekkel szembeni védelem, érzéketlenség**
- **Megoldás: Kimeneti és bemeneti jelszintek szélső értékei közötti kiterjesztett átfedés ($+V_{cc}$ =3,3V)**
  - $V_{Hmin(OUT)} = 2,4V$ $V_{Hmin(INP)} = 2,0V$ (0,4V zajtartalék)
  - $V_{Lmax(OUT)} = 0,4V$, $V_{Lmax(INP)} = 0,8V$ (0,4V zajtartalék)



Digitális technika áramköri alapok

- **Boole algebra, bináris aritmetika: két érték**
- **Logikai értelmezés: 0, 1, (L, H, LOW, HIGH)**
- **Szokásos értelemben:**
  - $V_H \leftrightarrow 1$, igaz, aktív, bekapcsolt
    $V_L \leftrightarrow 0$, hamis, inaktív, kikapcsolt
  - Negatív logikánál éppen fordított
  - $V_L \leftrightarrow 1$, igaz, aktív, bekapcsolt
    $V_H \leftrightarrow 0$, hamis, inaktív, kikapcsolt
    - Negatív logika használata jellemzően áramköri okokból
  - A dualitás következtében pozitív logikában az ÉS a negatív logikában a VAGY kapcsolatnak felel meg.

Digitális technika áramköri alapok

- **Logikai kapuk kimeneti meghajtása**
- **Általános tulajdonság:**
  - Kétállapotú kimenet,
    - 1, aktív felhúzás, (a felső PMOS tranzisztorok vezetnek)
    - 0, aktív lehúzás, (az alsó NMOS tranzisztorok vezetnek)
  - Következmény:

NORMÁL KIMENETEK NEM KAPCSOLHATÓK ÖSSZE KÖZVETLENÜL !!!!!

  - **Megoldási lehetőségek:**
    - **Logikai megoldás: MUX, adatút választás**
    - **Áramköri megoldás1: Tri-sate, 3 állapotú, Hi-Z kimenet**
    - **Áramköri megoldás2: Nyitott kollektoros kimenet**