



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

# **Digitális technika**

## **VIMIAA01**

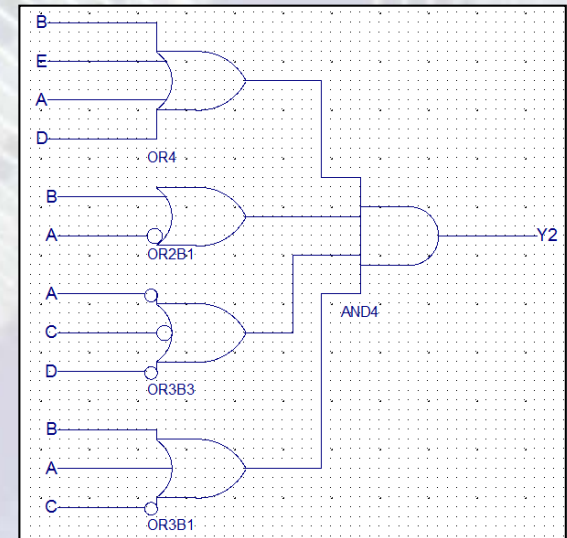
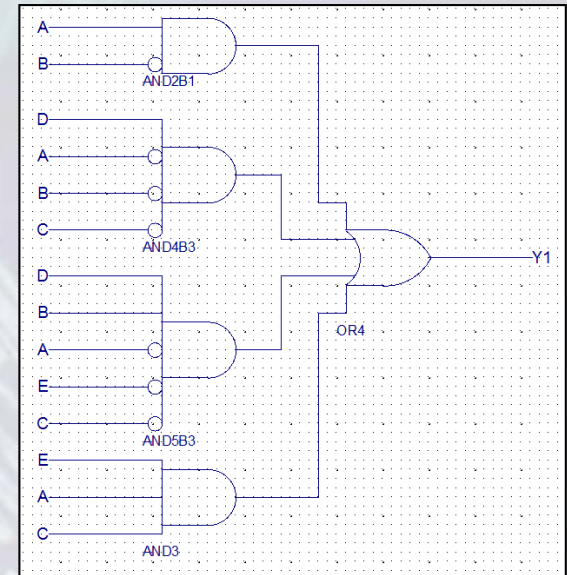
**Fehér Béla**  
**BME MIT**

# Minimalizálási algoritmusok

- Eddig csak kétszintű realizáció
  - Sum-of-Products (SOP)
    - ÉS-ek VAGY-a
  - (INV) – AND – OR

illetve

- Product-of-Sums (POS)
  - VAGY-ok ÉS-e
- (INV) – OR – AND



# Minimalizálási algoritmusok

- **További minimalizálási lehetőségek**
  - Többszintű realizáció
    - Közös szorzatkifejezések kiemelése
  - Több kimenetű realizációk
    - Közös szorzatkifejezések kiemelése és megosztása
- **Speciális optimalizáló program, MIS-II**
  - Kifejezetten erre a speciális területre hangolva
  - Redundáns logika megosztása
  - Közös feltételek kihasználása

# Minimalizálási algoritmusok

- **Többszintű realizáció**

- $F = A*B*C + A*B*D + /A*/C*/D + /B*/C*/D$

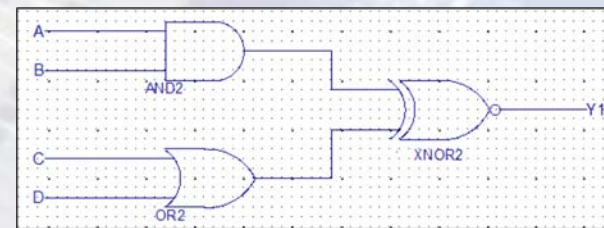
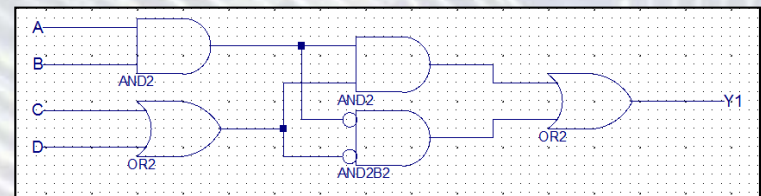
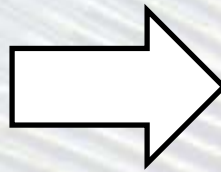
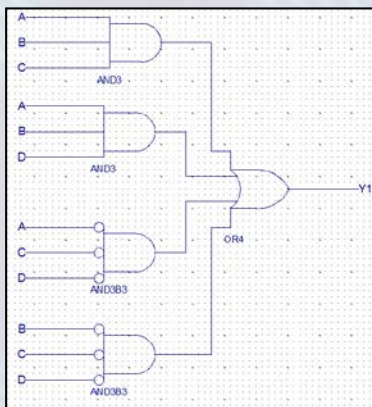
- A közvetlen realizáció 4 db AND3 + 1db OR4 kapu

- **Többszintű realizáció**

- Az  $X = A*B$  ill.  $Y = C+D$  helyettesítéssel

- $F = X*Y + /X*/Y = (A*B)*(C+D) + /A*B*/(C+D)$

- 3 db AND2 + 2 db OR2



# Minimalizálási algoritmusok

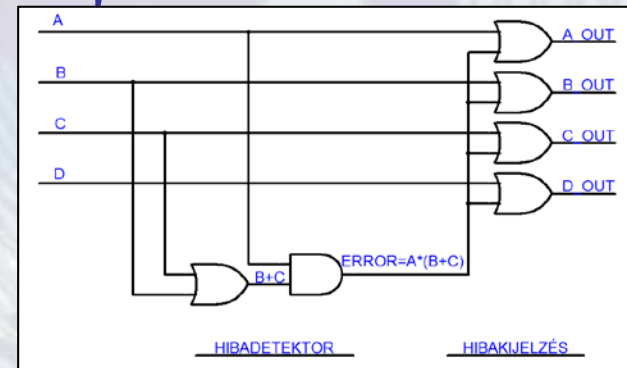
- **Többkimenetű minimalizálás**

- Példa: GYAK2. BCD kód hibajelzés

- $ERROR = A*(B + C)$

- Közös realizáció:

- 1 db AND2 + 5 db OR2



- Bitenkénti optimalizálás, szorzatok újrafelhasználása

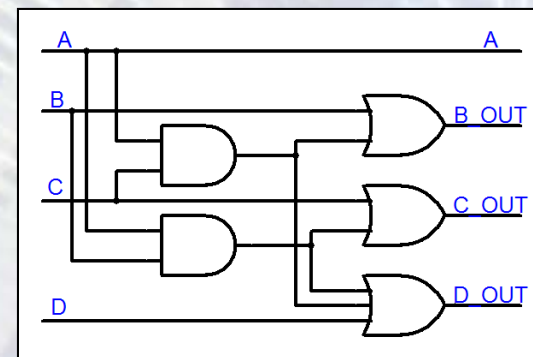
- $A\_OUT = A$

- $B\_OUT = B + A*C$

- $C\_OUT = C + A*B$

- $D\_OUT = D + A*B + A*C$

- 2 db AND2 + 3 db OR2



# 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 program
    - (UC Berkeley alapján)
  - Elérhetőség:  
<http://www.sontrak.com>

The screenshot displays the Logic Friday software interface. It features a menu bar (File, Operation, Truthtable, Equation, Gates, View, Help) and a toolbar. The main window is divided into several sections:

- Function List:** A table listing functions with their input and output counts, and minimization status.
- Truth Table:** A grid for entering input-output pairs (A, B, C, D to F0, F1).
- Equation:** A text area for entering logic equations, showing both the entered and minimized forms.
- Gate Diagram:** A schematic diagram showing logic gates (NOT, OR, AND) connected to inputs A, B, C, D and outputs F0, F1.

Function	Inputs	Outputs	True	False	DC	PI	Gates
F0-F1	4	2	2, 4	14, 12	0, 0	3	5
F0-F1(2)	4	2	2, 4	14, 12	0, 0	3	5
F6-F7	4	2	1, 1	15, 15	0, 0	Unminimized	Not mapped

Entered by truthtable:  
 $F0 = A' B' C' D + A' B C D'$   
 $F1 = A' B' C' D' + A' B' C' D + A' B' C D' + A' B' C D$

Minimized:  
 $F0 = A' B C D' + A' B C' D$   
 $F1 = A' B'$

Gate Diagram components:  
- NOT gates: (13) on A, (11) on C  
- OR gate: (15) on A and B  
- AND gates: (12) on C and D, (14) on the OR gate output and D  
- Outputs: F0, F1

# Funkcionális egységek

- Az eddigi látott módszerekkel az alapkapukból építkezve tetszőleges („random”) kombinációs logikai áramköröket felépíthetünk (elsősorban kis- és közepes komplexitásig).
- Kiindulunk az adott formában megadott specifikációból, elvégezzük a lehetséges minimalizációs műveleteket, (itt figyelembe vehetünk optimalizációs célokat, azaz legnagyobb sebesség, legkevesebb alkatrész) és kapusinten realizáljuk a függvényt
- A legtöbb esetben a Verilog specifikáció alapján a minimalizálást/optimalizálást a fejlesztőrendszerre hagyjuk.

# Funkcionális egységek

- **Az egyedi „kapuszintű” logikai függvények tervezésénél sokszor egyszerűbb/célravezetőbb szabványos modulokból építkezni és ezekből felépíteni a rendszert (ha nincsenek extrém optimalizációs előírások)**
- **A szabványos egységek célja, hogy a leggyakoribb, tipikus feladatokra biztosítsanak egyszerűen használható, megbízhatóan működő, skálázható megoldási lehetőséget**



# Funkcionális egységek

- **Általános célú kombinációs logikai funkciók**
  - Logikai funkciók (**DEC, ENC, PRI, MUX, DEMUX**)
  - Aritmetikai funkciók (**ADD, SUB, COMP, PAR**)
  - Egyéb, esetleg szükséges funkciók
- **Hagyományos technológiában (TTL MSI IC-k)** minden funkcióra önálló elem/alkatrész létezett, különböző méretekben, tokozásban, lábszámban
- **FPGA-ban, könyvtári alapú, modulszintű** építkezésnél paraméterezhető alapfunkciók, az aktuális terv igényei szerint beállítva
- **HDL alapú tervezésnél akár „instant”** specifikáció, természetesen felhasználva tervezési mintákat

# Funkcionális egységek: Dekóder

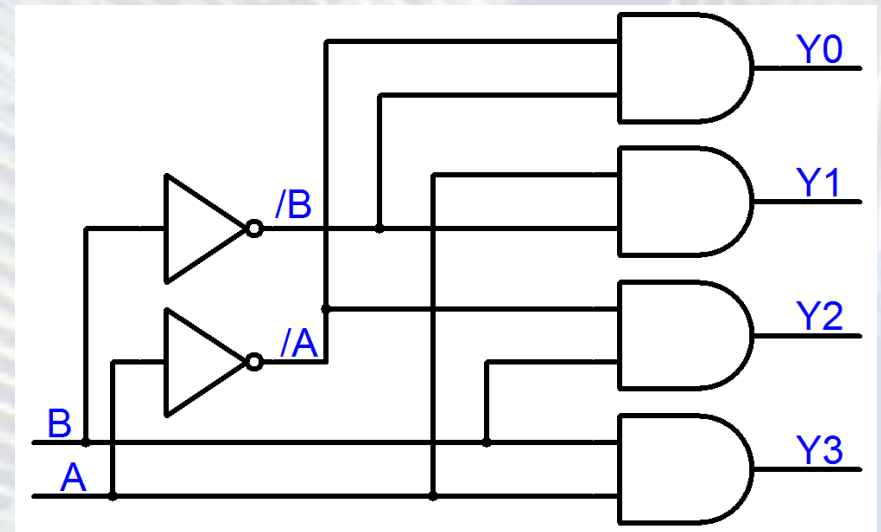
- A dekóder a binárisan értelmezett bemeneti jelekből az általános logikai függvény összes mintermjét előállítja (az igazságtábla minden sorához van „1”)

$$\text{KIM} = f(\text{BEM})$$

- Ha a BEM egy  $n$  bites binárisan kódolt bemenet, akkor KIM egy  $2^n$  méretű,  $1$ -a- $2^n$ -ből kódolású bitvektor, ezért hívjuk DE-KÓDOLÁSNAK
- A tipikus méretek: 1:2, 2:4, 3:8, 4:16. (de lehet 4:10 is)
- Lehet nagyobb méretben is, de esetleg érdekesebb az egyszintű dekóder helyett a többszintű sor-oszlop vagy hierarchikus fa struktúrájú felépítést használni.

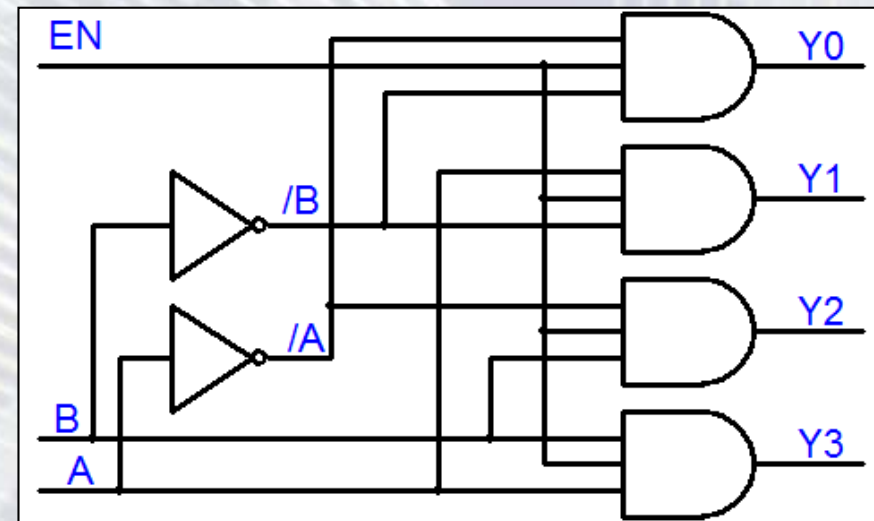
# Funkcionális egységek: Dekóder

- Egyszintű, közvetlen dekóderek felépítése
- A dekóder kimeneti bitjei egyenként megfelelnek az  $n$  bemenetű általános logikai függvények egy-egy mintermjének, azaz minden változó szerepel benne, 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)



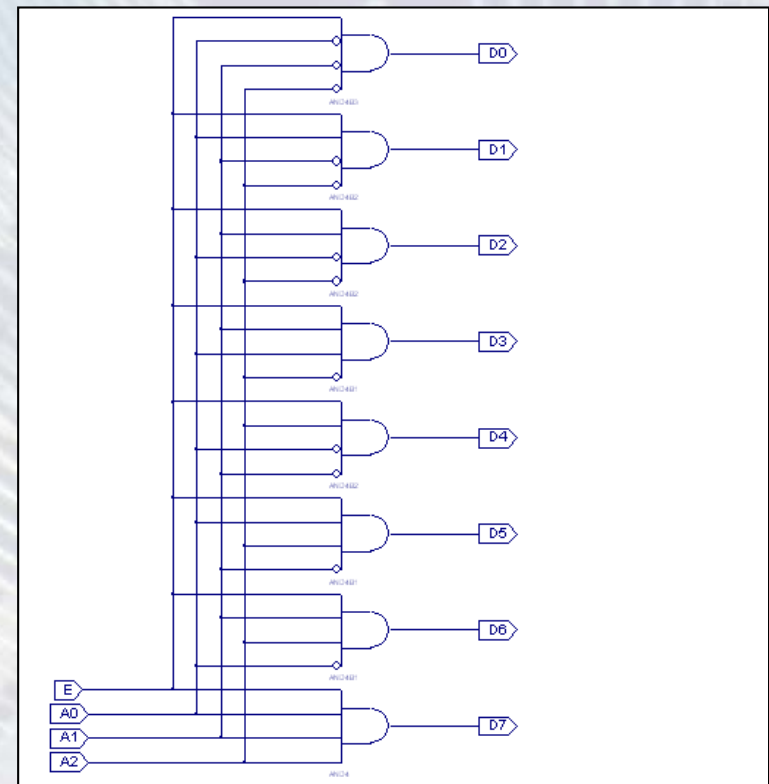
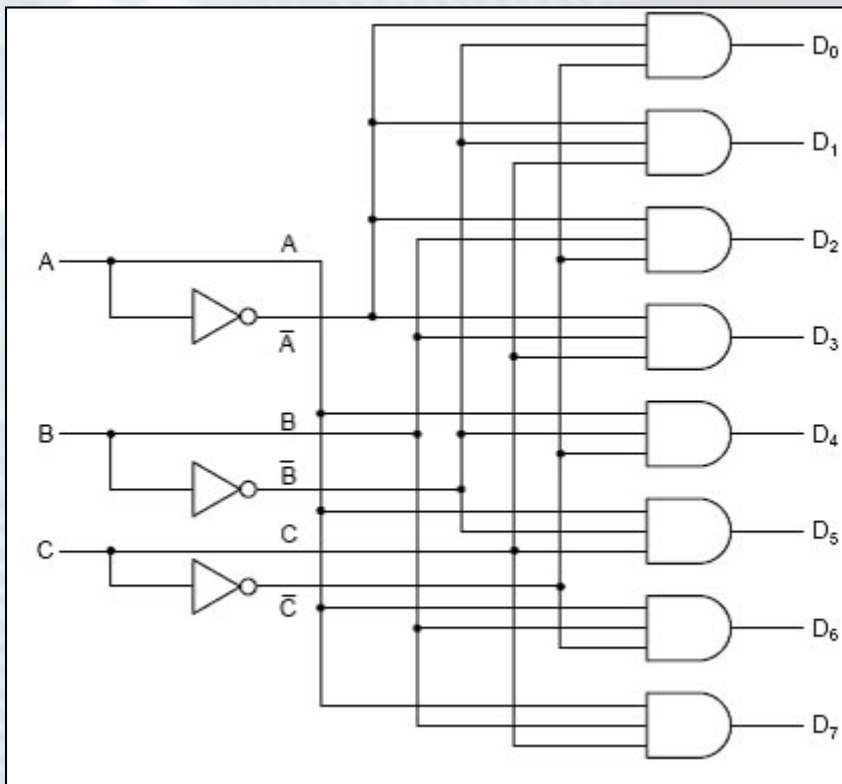
# Funkcionális egységek: Dekóder

- A dekóder egyik kimenete értelemszerűen mindig aktív (A bemeneten mindig van egy kombináció)
- Ezért érdemes bevezetni egy ezt felülbíró jelet: **ENGEDÉLYEZÉS.**
  - Ha  $EN = 0$ , minden kimenet inaktív
  - Így könnyen szabályozható a kimenetek által vezérelt egységek működése
  - Egyébként is ez a DEK legfontosabb funkciója
  - Könnyű a többszintű bővíthetőség



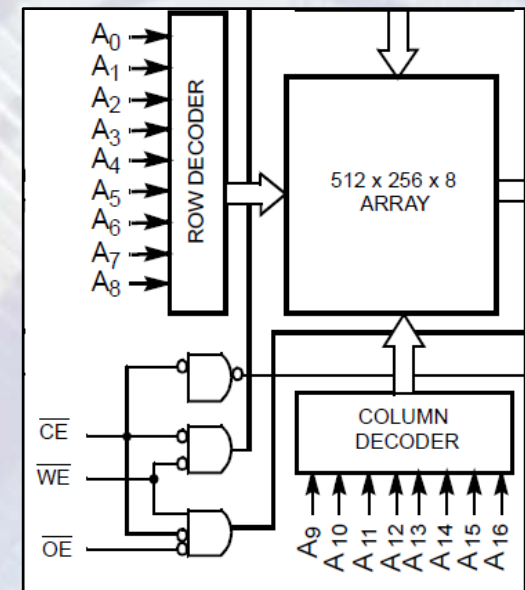
# Funkcionális egységek: Dekóder

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



# Funkcionális egységek: Dekóder

- **Nagyméretű dekóderek**
  - 60 kimenetre 60 db 6 bites ÉS kapu
  - Egy memóriában pl.  $2^{17}$  kimenetre 131072 db 17 bemenetű ÉS kapu
- **Más stratégia kell**
  - Memória: sor-oszlop dekóder, mátrixszerű lokális engedélyezés, 2 bemenetű ÉS kapukkal
  - A korábban bevezetett engedélyező bemenet használatával többszintű, hierarchikus felépítés

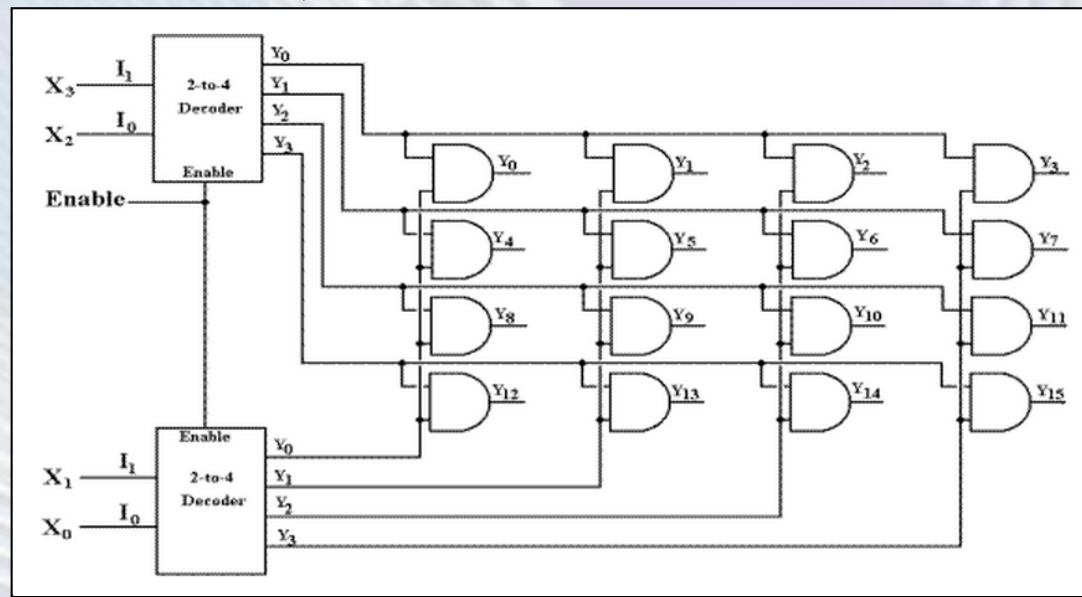
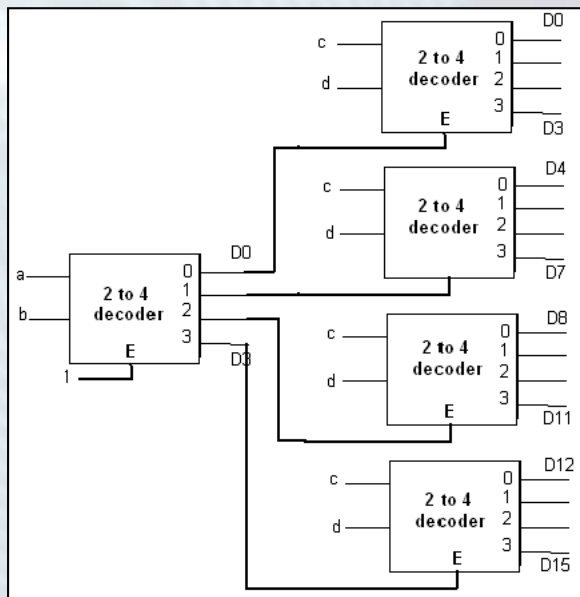


# Funkcionális egységek: Dekóder

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

HIERARCHIKUSAN

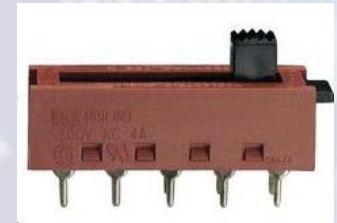
2D, SOR-OSZLOP módban



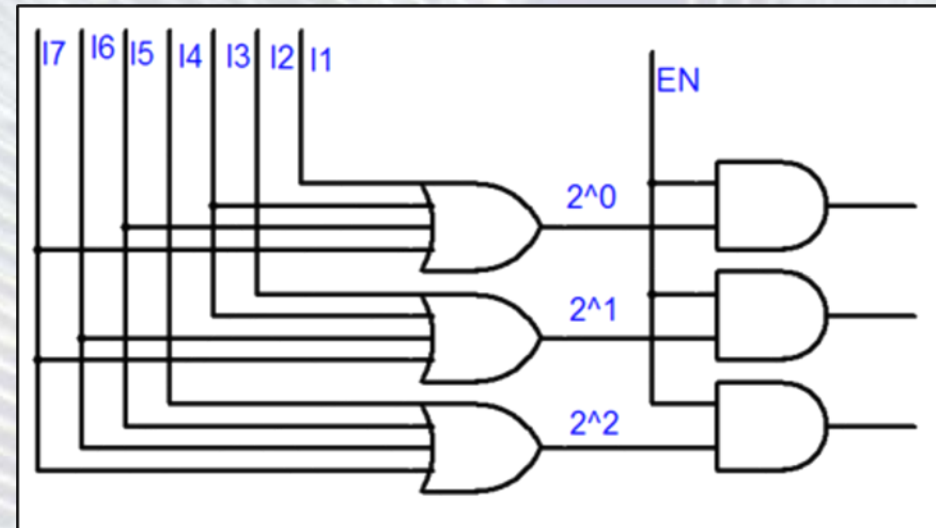
Verilog HDL kódolási minták a Labor 3 diákon

# Funkcionális egységek: Enkóder

- Az ENC enkóder a dekóder funkció inverzét kínálja
- Valójában nem annyira általánosan használt elem
- Közvetlenül csak akkor használható, ha a bemenet igazi 1-az-N-ből kódolású
- Bizonyos értelemben adattömörítést végez,  $N$  bitből  $\lceil \log_2 N \rceil$  bitre



- Mi legyen az I0 bittel?
- Nem 1-az-N bemenet?
- Külön kimenet az érvénytelenítésre: INVALID vezérlés





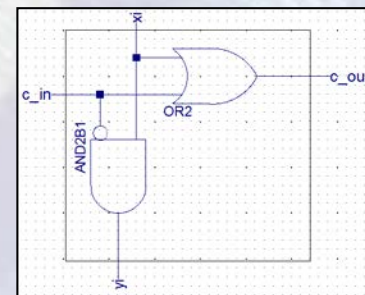
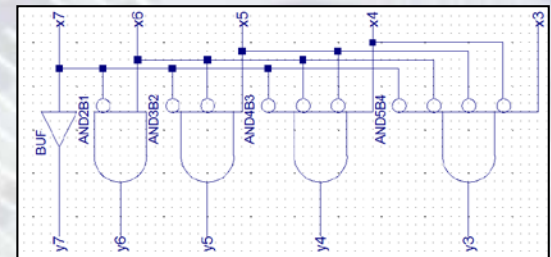
# Funkcionális egységek: Prioritás Enk.

- A normál enkóder funkcionalitásának javítása
- Tetszőleges bemeneti kódot feldolgoz. A kimeneten a legnagyobb súlyú aktív bit bináris indexét adja ki.
- Itt is van tömörítés  $N$  bitből  $\lceil \log_2 N \rceil$  bitre
- Sajnos itt is van érvénytelen bemenet, a csupa 0.
- Realizáció: 2 lépésben
  - Prioritás feloldás
    - Párhuzamos
    - Iteratív
  - Bináris enkóder

BEMENETEK									KIMENETEK			
EI	I7	I6	I5	I4	I3	I2	I1	I0	O2	O1	O0	INV
0	x	x	x	x	x	x	x	x	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	x	0	0	1	0
1	0	0	0	0	0	1	x	x	0	1	0	0
1	0	0	0	0	1	x	x	x	0	1	1	0
1	0	0	0	1	x	x	x	x	1	0	0	0
1	0	0	1	x	x	x	x	x	1	0	1	0
1	0	1	x	x	x	x	x	x	1	1	0	0
1	1	x	x	x	x	x	x	x	1	1	1	0

# Funkcionális egységek: Prioritás Enk.

- **Prioritás feloldás: A legnagyobb helyiértékű aktív bit jelzése, összes többi 0**
  - Az előző táblázat x (Don't Care) bejegyzéseinek kezelése 001xxxxx → 00100000
- **Tetszőleges kód leképezése 1-az-N-ből kódra**
  - Amint egy bit aktív, nullázza a kisebb helyiértékeket
    - Párhuzamos megoldás
      - Gyors, de túl sok kapubemenet
    - Iteratív megoldás, minden bitre
      - $y_i = /c\_in * x_i$
      - $c\_out = c\_in + x_i$
      - Letiltás terjesztése balról jobbra
      - 2 bemenetű kapuk, lassú jelterjedés



# Funkcionális egységek: Prioritás Enk.

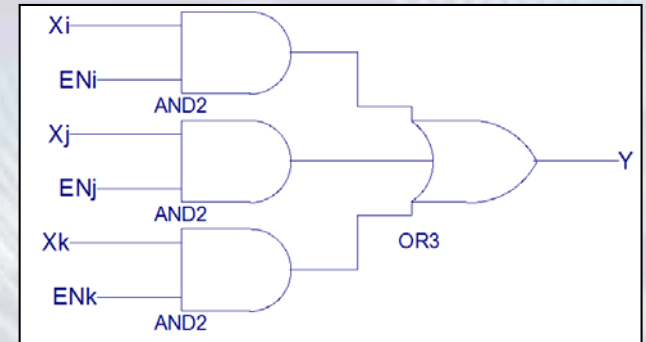
- Verilog HDL környezetben speciális nyelvi eszközök
- **always @ (\*)** blokk szerkezet
  - A viselkedési leírás támogatására
- **casex** kulcsszó
  - A Don't Care bejegyzések egyszerű kezelésére

```
////////////////////////////////////  
// Specifikáció az igazságtáblázat alapján, kihasználva a don't care jelölést  
// Ebben az esetben a casex kulcsszó használatával tömör, kifejező formában  
// írhatjuk fel a bemenet - kimenet közötti kapcsolatot  
////////////////////////////////////  
  
reg [2:0] rout;  
  
always @ (*)  
begin  
    if (en)  
        casex (dinp)  
            8'b00000001 : rout = 3'b000;  
            8'b0000001x : rout = 3'b001;  
            8'b000001xx : rout = 3'b010;  
            8'b00001xxx : rout = 3'b011;  
            8'b0001xxxx : rout = 3'b100;  
            8'b001xxxxx : rout = 3'b101;  
            8'b01xxxxxx : rout = 3'b110;  
            8'b1xxxxxxx : rout = 3'b111;  
            default:    rout = 3'b000;  
        endcase  
    else  
        rout = 3'b000;  
    end  
  
assign pout = rout;  
  
assign inval = ~en | ~|dinp;
```

# Funkcionális egységek: Multiplexer

- A legfontosabb funkcionális elem
- Alapvető feladata az adatforrás, adatút választás megvalósítása

- $X_i, X_j, X_k$  az adatbemenetek
- $EN_i, EN_j, EN_k$  engedélyezés

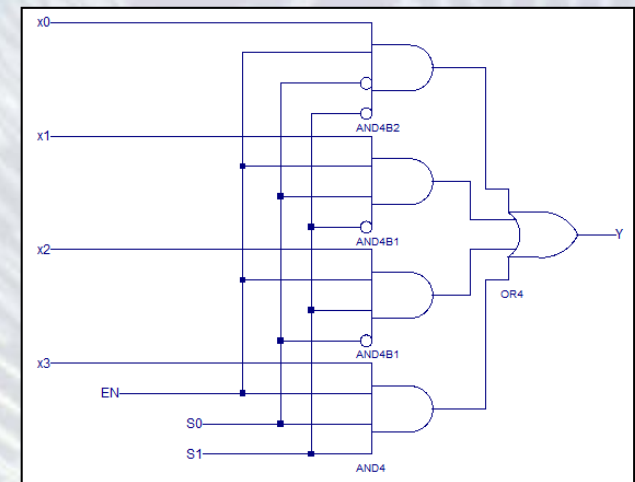
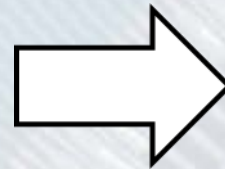
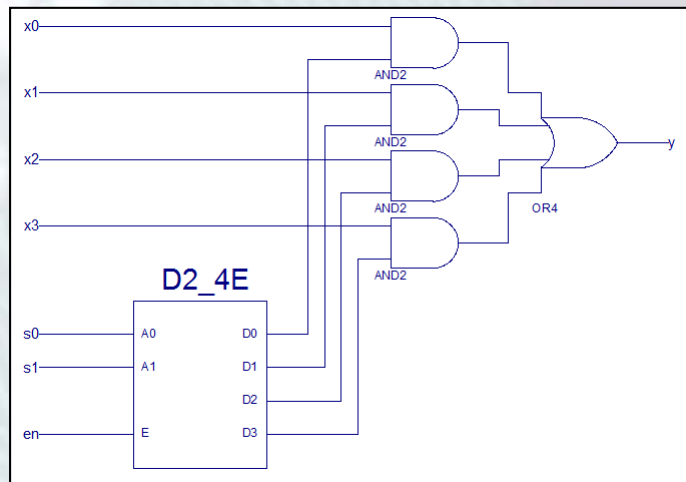


$$Y = X_i * EN_i + X_j * EN_j + X_k * EN_k$$

- Természetesen tetszőleges számú bemenetre
- Helyes működés feltétele: az EN bitvektor legyen 1-az-N-ből kódolású → Vagyis egy dekóder kimenete
- Megjegyzés: Az AND-OR hálózat helyettesíthető 3 állapotú Hi-Z kimenetű meghajtókkal is (később...)

# Funkcionális egységek: Multiplexer

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



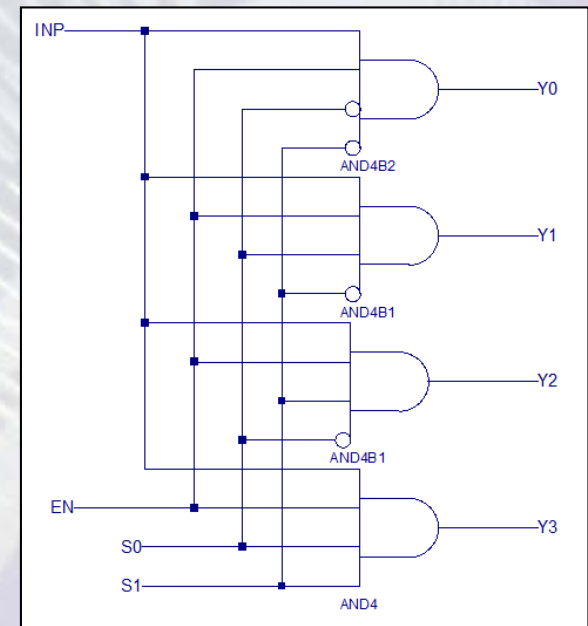
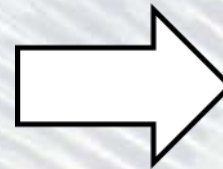
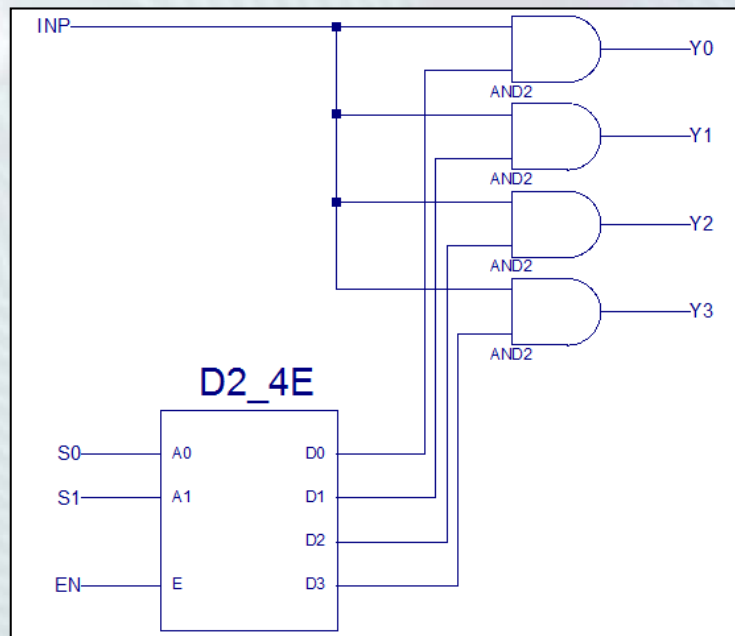
- Jellemző méretek: 2:1, 4:1, 8:1, 16:1
- Verilog specifikációja sokféle lehet, tervezési stílus kérdése → Példák a Labor 3 diákon

# Funkcionális egységek: Multiplexer

- **A MUX is felépíthető hierarchikusan, fa struktúrában**
- **A struktúra generálható az LSb vagy az MSb felől**
  - LSb-vel kezdve első lépés a páros-páratlan bitek közti választás, majd így haladunk tovább, mindig az aktuális szomszédok között választva
  - MSb-vel kezdve első lépés az adatbemenetek felezése (első-második fele), majd tovább a maradék felezése (negyedelés) és í.t. az utolsóig
  - A fa felépíthető 4:1, 8:1 vagy nagyobb lépésekben is, ekkor kevesebb szint kell, de több bemenetűek a kapuk
  - A választott megoldás sok más tényező függvénye is lehet, pl. egyszerűen a huzalozás megoldhatósága

# Funkcionális egységek: DEMUX

- A DEMUX demultiplexer a multiplexer funkció inverze
- Egyetlen adatforrást több kimenetre képes szétosztani
  - Lényegében ez is DEK + AND adatkapu felépítésű
  - Szokásos méretei 1:2, 1:4, 1:8

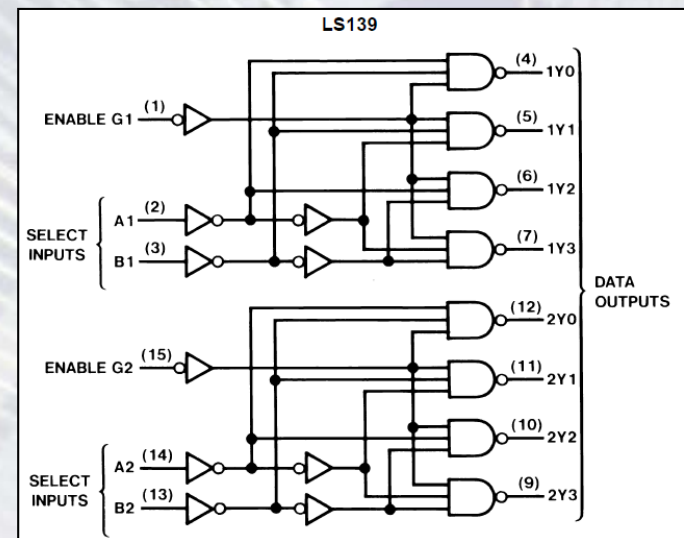
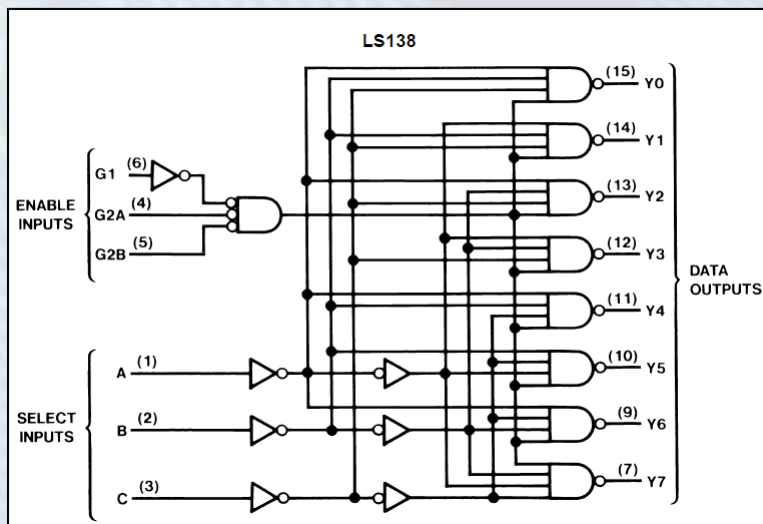


# Funkcionális egységek: DEMUX

- A DEMUX demultiplexer a multiplexer funkció inverze
  - Bár szakértőszemmel inkább csak egy dekóder
  - Szokásos katalógusneve is ezt tükrözi
  - A lényeg az EN jelek használati módjában van

## DM74LS138, DM74LS139 Decoders/Demultiplexers

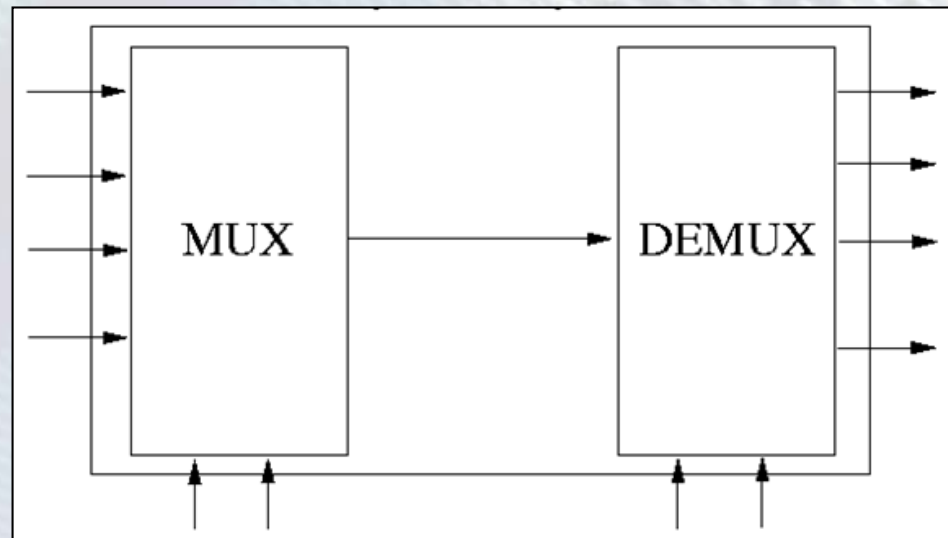
The LS139 comprises two separate two-line-to-four-line decoders in a single package. The active-low enable input can be used as a data line in demultiplexing applications.





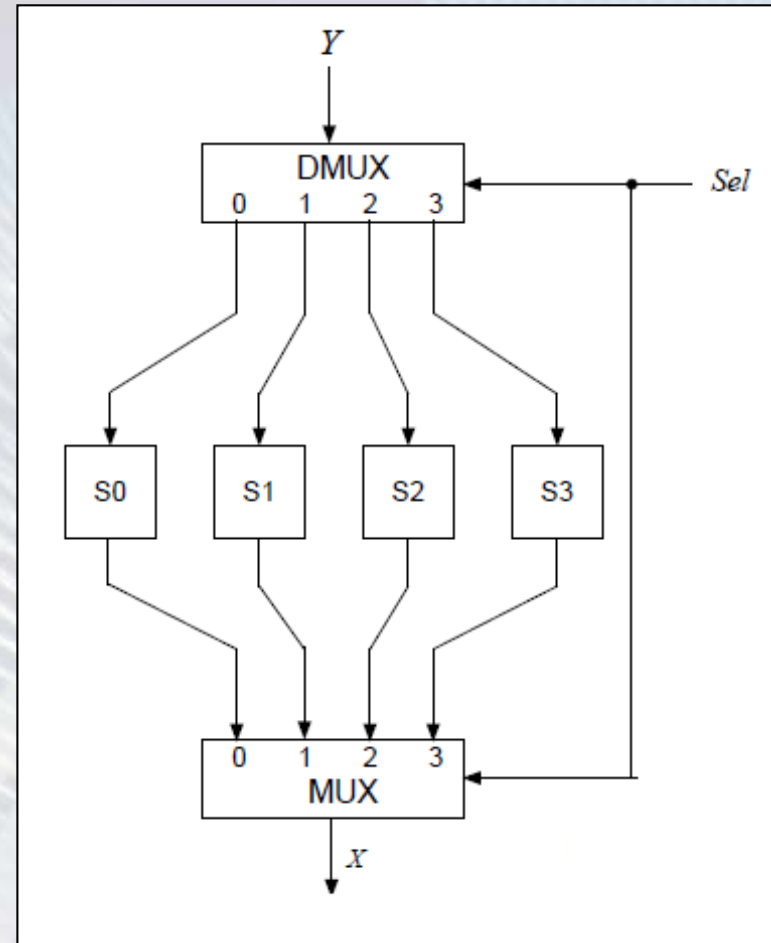
# Funkcionális egységek: DEMUX

- **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

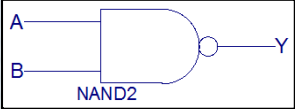
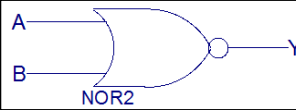


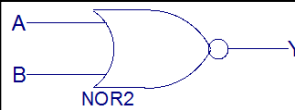
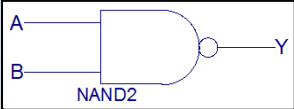
# Funkcionális egységek: DEMUX

- **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 ( $\Delta t = 10\text{ns}$ ) 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



# Univerzális logikai elemek

- A Boole algebra definíciója alapján a 3 alpművelettel AND, OR, INV minden logikai függvény előállítható
- Ez 3 fajta alapelemet jelent. Nem sok, de ....
- Univerzális logikai elem: Egyetlen alapelemmel minden logikai függvény előállítható. Létezik ilyen?
- Igen, 2 is: NAND, NOR kapuk ilyenek
- Bizonyítás: Ha a három alapfüggvény realizálható velük, akkor beláttuk. De Morgan tétel alapján
- NAND  $\neg(A * B)$   NOR  $\neg(A + B)$    
INV:  $\neg(A * A)$   $\neg(A + A)$   
AND:  $\neg(\neg(A * B) * \neg(A * B))$   $\neg(\neg(A + A) + \neg(B + B))$   
OR:  $\neg(\neg(A * A) * \neg(B * B))$   $\neg(\neg(A + B) + \neg(A + B))$



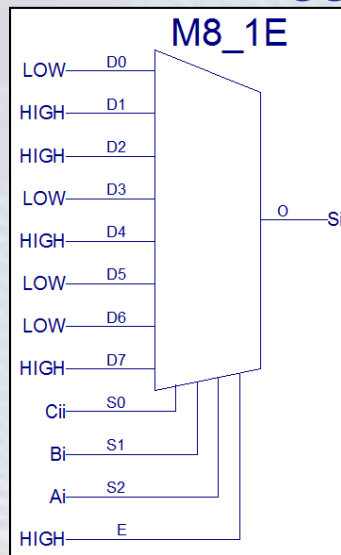
# Univerzális logikai elemek

- **Tehát a legegyszerűbb univerzális elemek:**
  - A NAND és NOR kapuk → Homogén hálózatok
- **Lehetséges más univerzális elemkészletet is használni:**
  - Multiplexer
  - Memória táblázat (LUT, Look Up Table)
  - Első közelítésben ez bonyolultabbnak látszik, de
    - A szabvány funkciók nagyon hatékonyan és gazdaságosan realizálhatók a CMOS VLSI félvezető IC technológiában, de ez csak a tranzisztorszintű kapcsolásban látszik
    - Az egységek közvetlenül többváltozós függvényeket képesek megvalósítani, ezért a tervezés egyszerűbb, kapu szintű minimalizálás, ill. optimalizálás szinte nem is szükséges

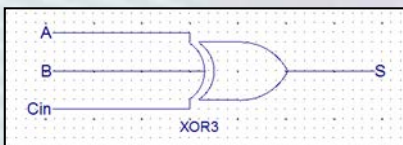
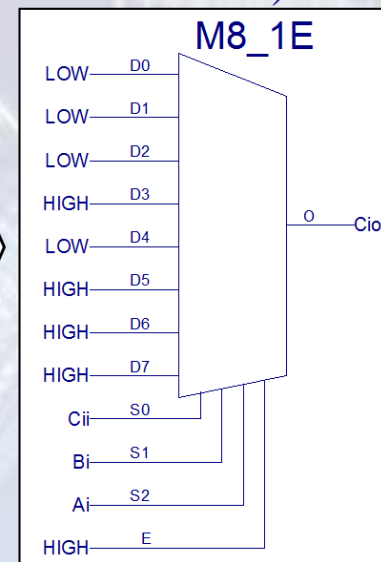
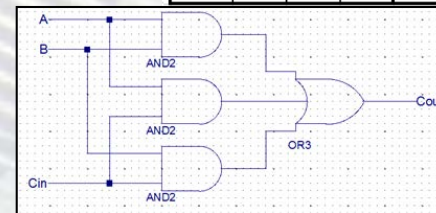
# A Multiplexer, mint UNIV elem

- **Multiplexer = Univerzális logika**
- **Bemeneti változók: A kiválasztó (SEL) bemenetek**
- **Függvény megadása: Az igazságtábla kimeneti jel oszlopának értékeit konstans jelként a MUX adatbemeneteire 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



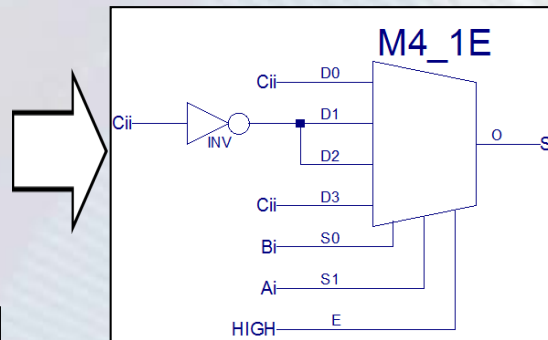
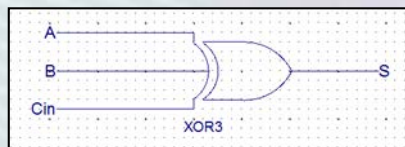
BEMENETEK				KIM
INDX	A	B	C	F2
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



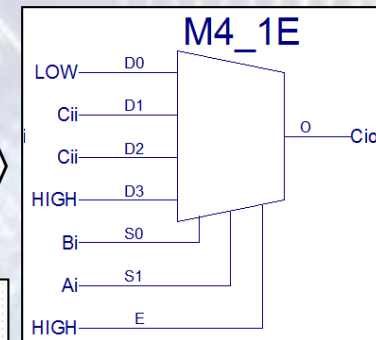
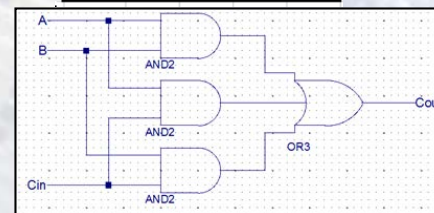
# A Multiplexer, mint UNIV elem

- **Bemeneti változók:** A kiválasztó (SEL) bemenetek, de egy változó (ponált/negált értékei) az adatbemenetekre kapcsolhatók (MUX méret felére csökken)
- **Függvény megadása:** Az igazságtábla kimeneti jel oszlopának értékeit a kiválasztott jellel adjuk meg
- **Az előző példa alapján (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



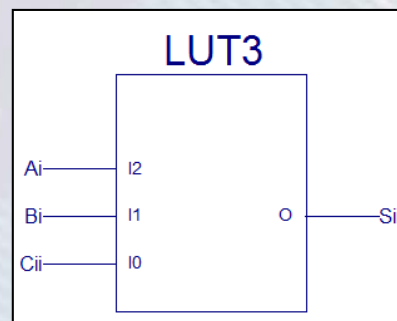
BEMENETEK				KIM
INDX	A	B	C	F2
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



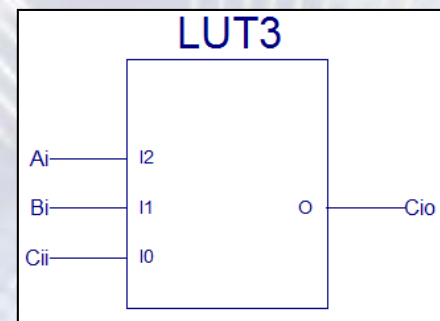
# A memória táblázat, mint UNIV elem

- **Memória = Univerzális logika**
- **Bemeneti változók: A memória címbitjei**
- **Függvény megadása: Az igazságtábla kimeneti jel oszlopának értékeit konstans jelként beírjuk a memória sorindex szerinti címeire**
  - 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

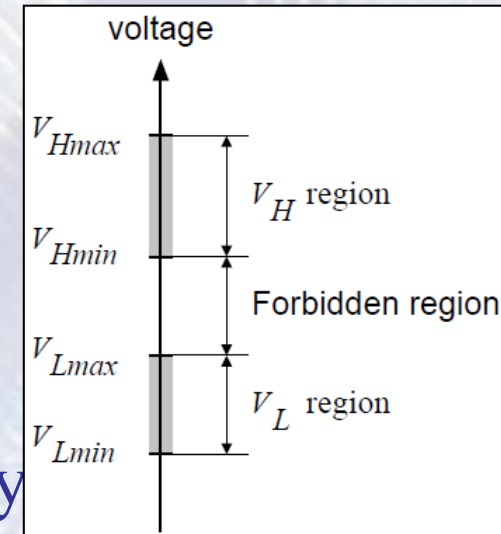


BEMENETEK				KIM
INDX	A	B	C	F2
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



# Digitális technika áramköri alapok

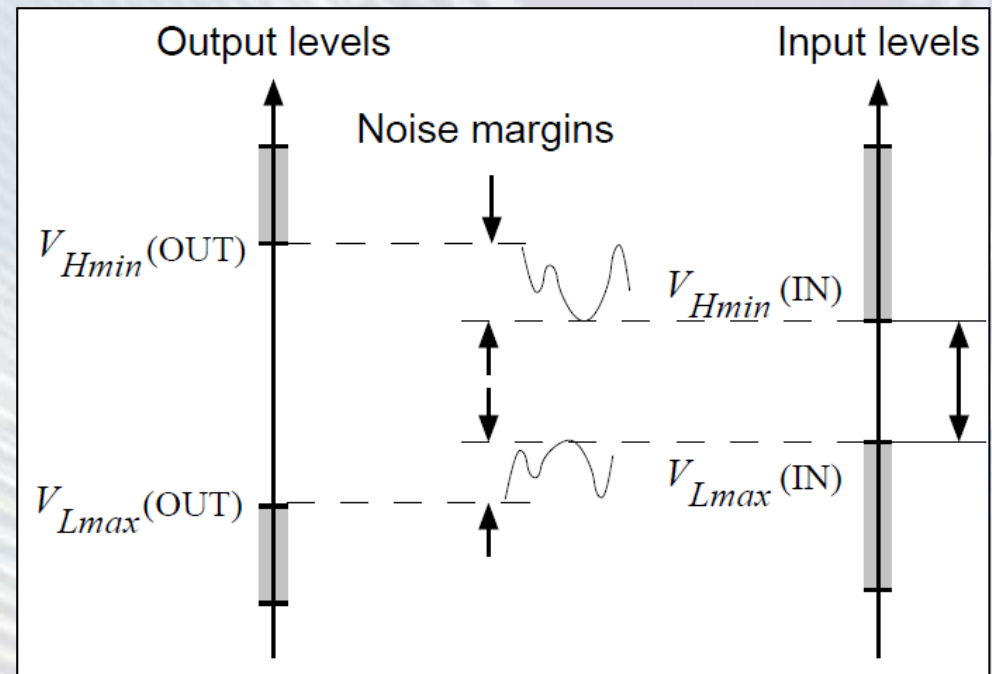
- **Boole algebra, bináris aritmetika: két érték**
- **Logikai értelmezés: 0, 1, (L, H, LOW, HIGH)**
- **Fizikai reprezentáció: Feszültség szint GND = 0V, a  $+V_{cc}$  lehet +5V, +3,3V, +2,5V, 1,8V, +1,2V, ...**
- **$P = k * V_{cc}^2 * f_{clk}$ , azaz  $V_{cc}$  csökkentése jelentősen javítja a fogyasztást.**
- **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)



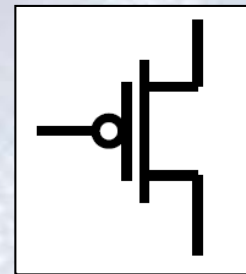
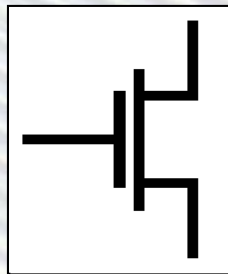
# Digitális technika áramköri alapok

- **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

- Történelmileg különböző áramköri verziók
- Ma a CMOS technológia domináns (NMOS, PMOS)
- Komplementer működésű tranzisztoros kapcsolók
- Az áramköri alkalmazás következtében

• $V_{IN}$ Vezérlőjel	NMOS tranzisztor	PMOS tranzisztor
$V_L$	nem vezet	vezet
$V_H$	vezet	nem vezet
• $V_L$ vezérlésnél	KIKAPCSOL	BEKAPCSOL
• $V_H$ vezérlésnél	BEKAPCSOL	KIKAPCSOL



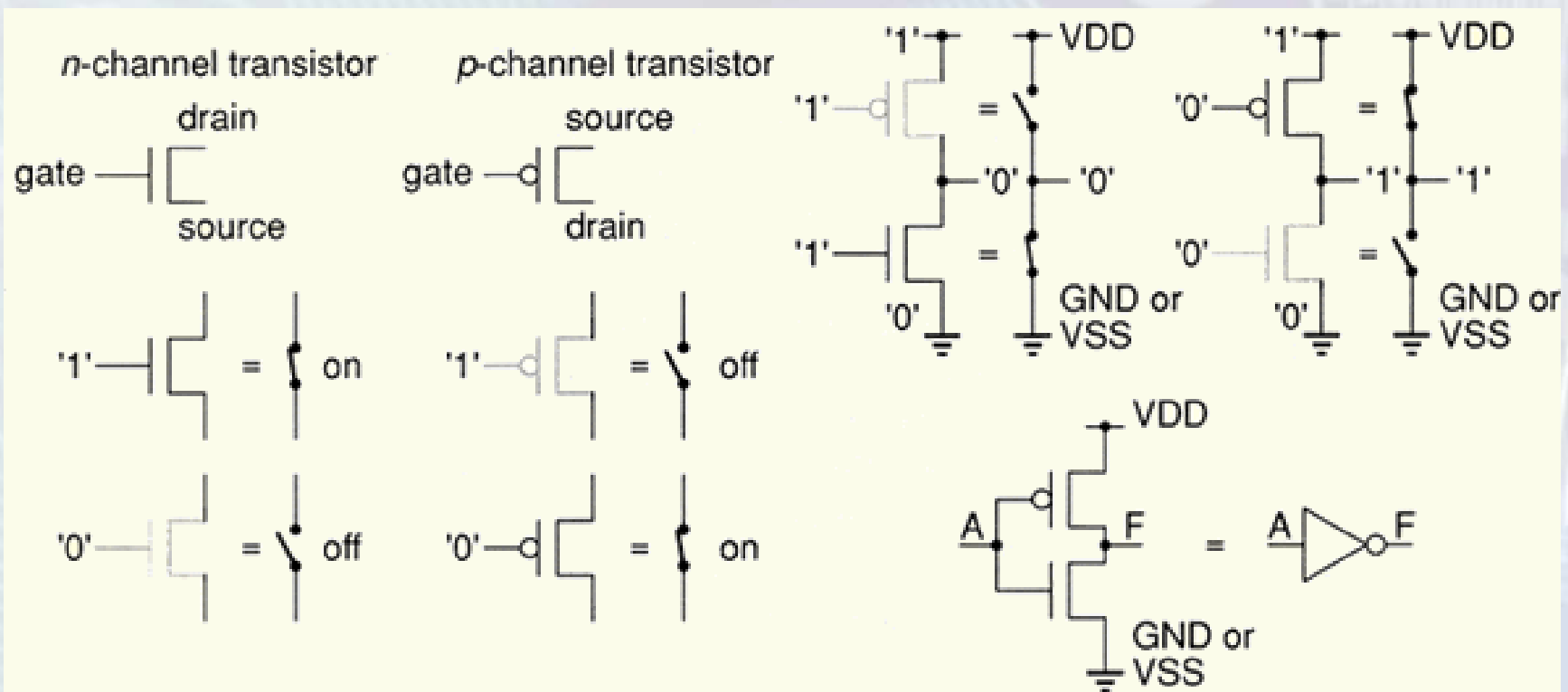
# Digitális technika áramköri alapok

- Összefoglaló a tranzisztorok fontosabb jellemzőiről

NMOS

PMOS

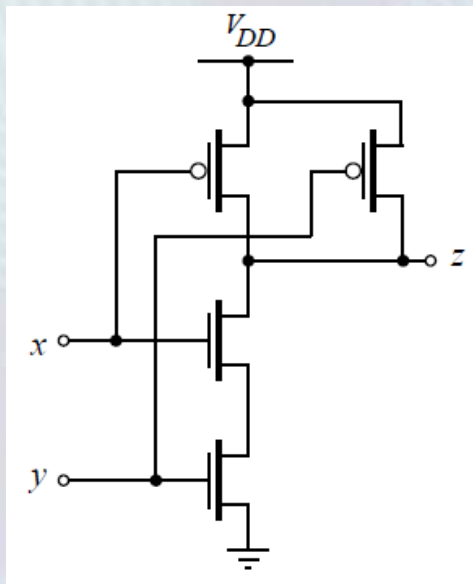
CMOS INV



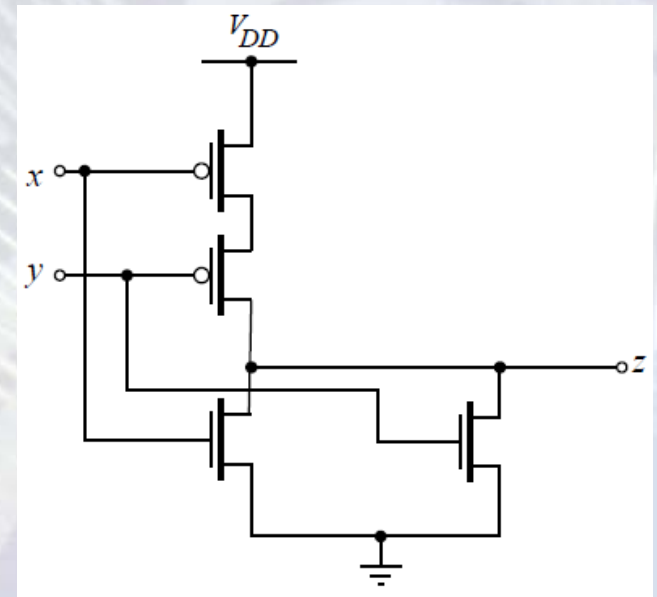
# Digitális technika áramköri alapok

- **Elemi kapuk : NAND és NOR**
- **Logikai kapcsolat: A kapcsoló tranzisztorok soros és párhuzamos kapcsolása (annyi, amennyi bemeneti jel)**

## NAND

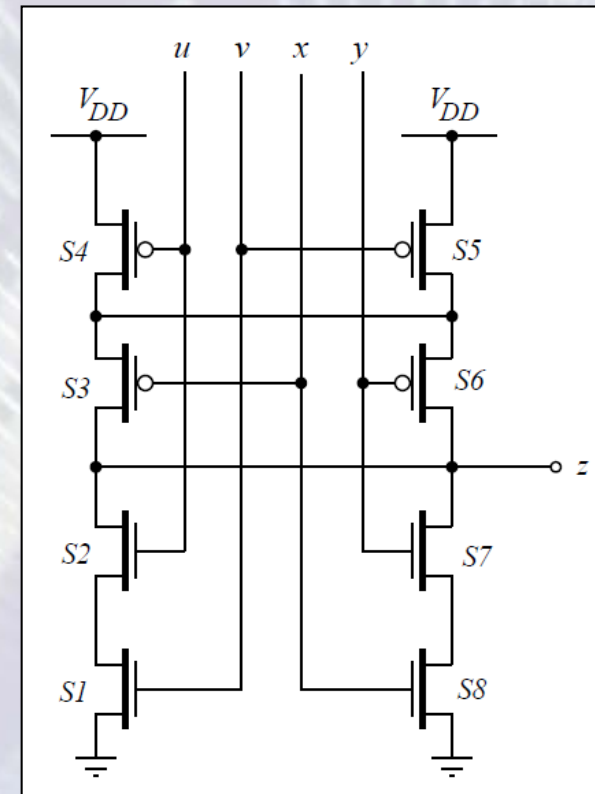
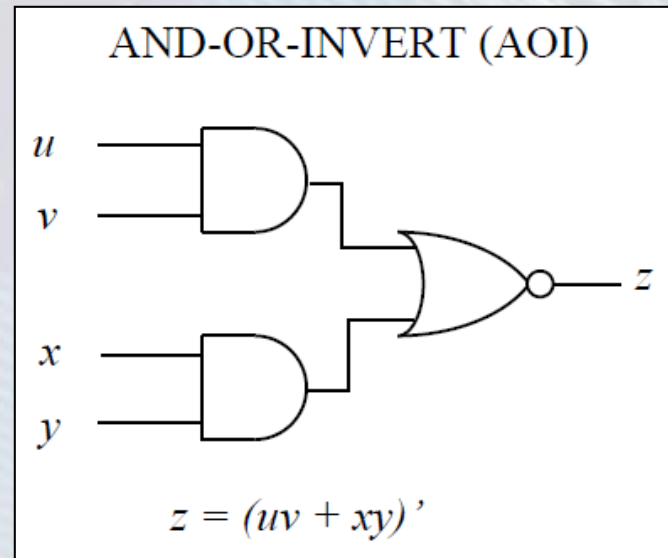


## NOR



# Digitális technika áramköri alapok

- Egy érdekes tranzisztorszintű elrendezés
  - Nem vizsgaanyag
- Csak demonstráció, hogy a logikai áramkörök nem feltétlenül kapukból épülnek fel, és mindaz amit kapuszenten értelmezünk, esetleg teljesen értelmetlen.



# Digitális technika áramköri alapok

- Egy másik példa (Ez sem vizsgaanyag)

United States Patent [19]

Veenstra

[54] UNIVERSAL LOGIC MODULE WITH ARITHMETIC CAPABILITIES

[75] Inventor: Kerry S. Veenstra, San Jose, Calif.

[73] Assignee: Altera Corporation, San Jose, Calif.

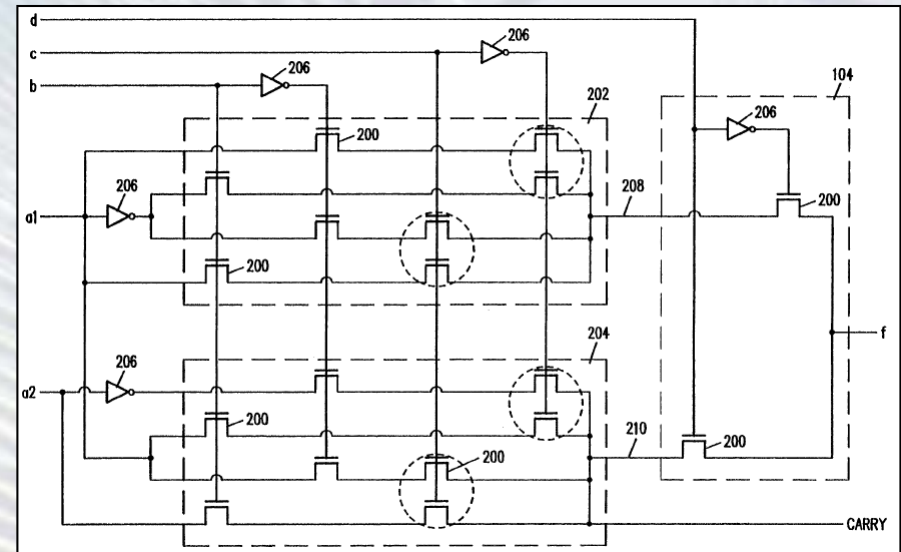
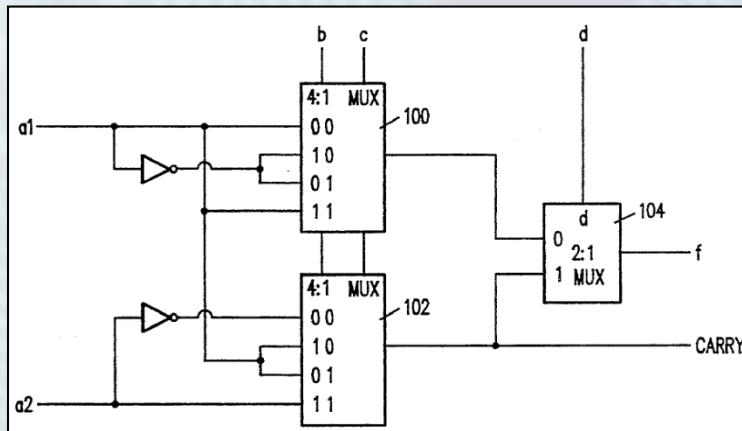
[21] Appl. No.: 153,321

[22] Filed: Nov. 12, 1993

[57]

ABSTRACT

A universal logic module for use in a programmable logic device, capable of generating all logical functions of three variables or less. The universal logic module also implements a full adder with carry propagation.



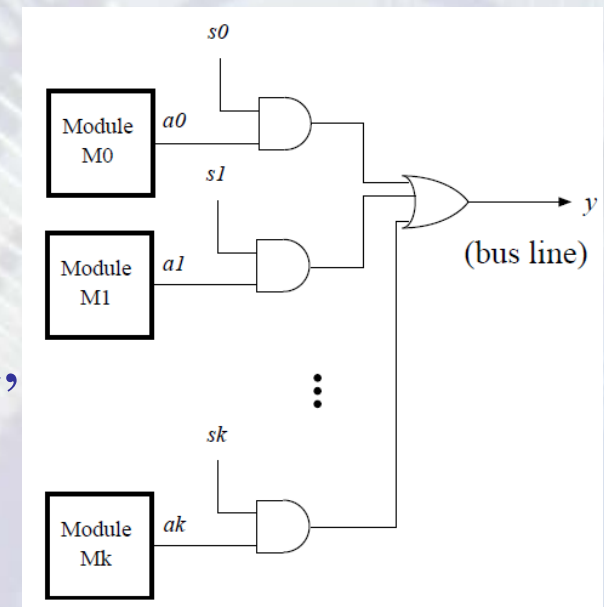
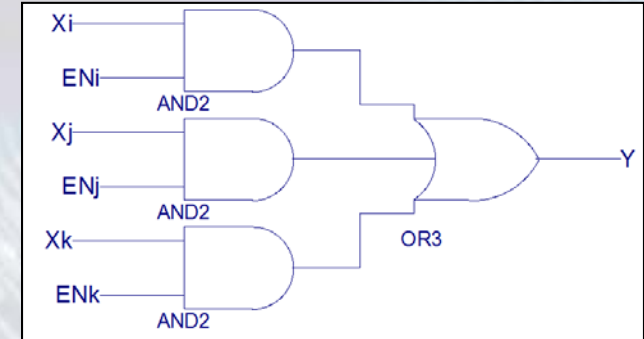
# Digitális technika áramköri alapok

- **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-state, 3 állapotú, Hi-Z kimenet
  - Áramköri megoldás2: Nyitott kollektoros kimenet



# Digitális technika áramköri alapok

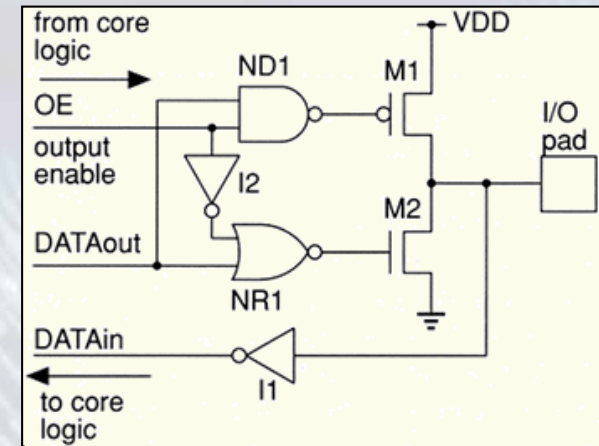
- **Logikai megoldás:**
  - **ÉS-VAGY kapuhálózat**  
Ha  $EN_i = 1$ , a kimenet =  $X_i$ ,  
ha  $EN_i = 0$ , akkor 0, azaz  
nincs engedélyezve a meghajtás
- **Használata:**
  - Minden forrás kimenetét egy-egy ÉS kapu „kapuzza” a közös buszra
  - Az engedélyezés (itt **si**) a helyes működéshez **1-az-N-ből** kódolású, de nem okoz meghibásodást, ha véletlenül nem az (de helyes működést sem..)



# Digitális technika áramköri alapok

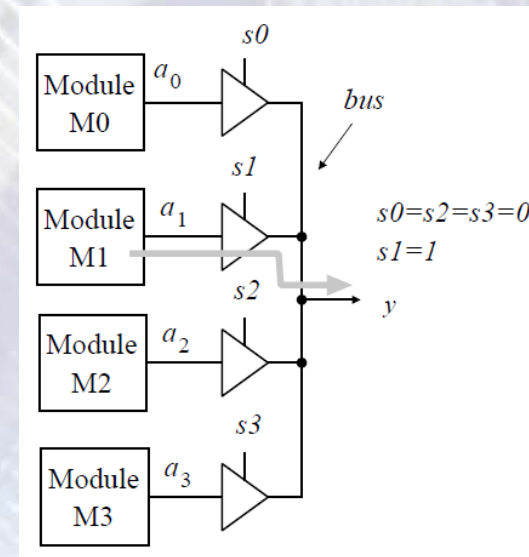
- **Áramköri megoldás 1:**

- Speciális, 3 állapotú, tri-state, Hi-Z kimenet. Ha  $OE = 1$ , a kimenet = DATAout, ha  $OE=0$ , akkor Hi-Z, azaz kikapcsolt, nincs meghajtva



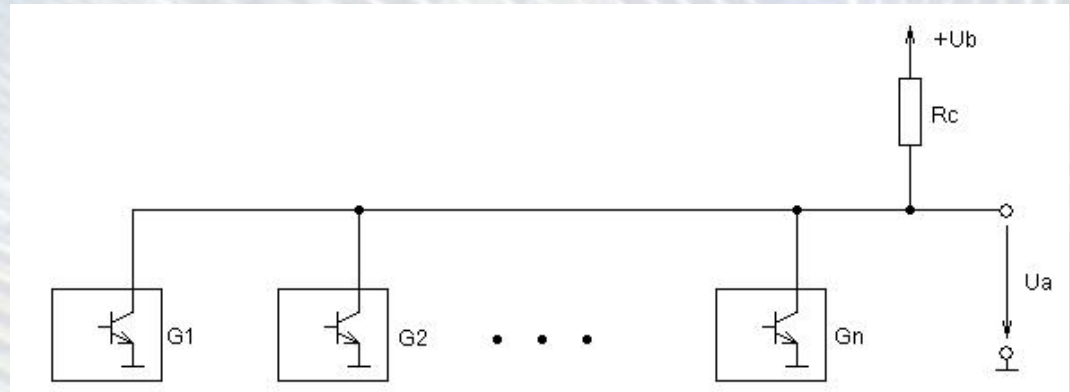
- **Használata:**

- Minden forrás Tri-state meghajtón keresztül kapcsolódik a közös buszra
- Az engedélyezés (itt **si**) **szigorúan 1-az-N-ből** kódolású, a legjobb, ha egy dekóder kimenete (több kimenet egyidejű engedélyezése áramköri meghibásodást okoz)



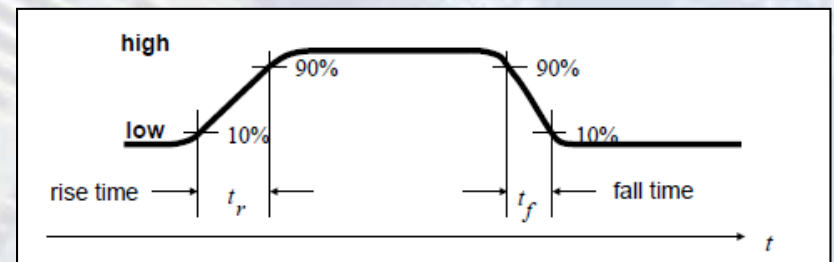
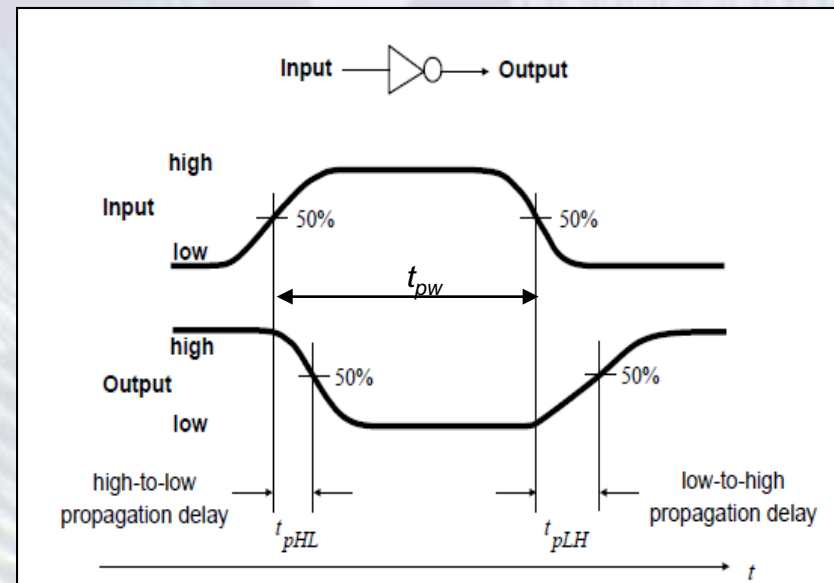
# Digitális technika áramköri alapok

- **Áramköri megoldás 2: Nyitott kollektoros kimenet**
- A kapukimenet csak lehúzó áramot tud generálni (csak az alsó tranzisztorok vannak beépítve, az ábrán TTL bipoláris tranzisztorok)
- Pozitív logikában huzalozott ÉS : A kimenet csak akkor magas, ha minden kimenet kikapcsolt, azaz egyik sem aktív alacsony.
- Negatív logikában huzalozott VAGY: A kimenet akkor alacsony, ha legalább egy kimenet alacsony.
- A kimeneti magas szintet a felhúzó  $R_c$  ellenállás biztosítja (lassú jelváltás)



# Logikai kapuk időzítési jellemzői

- **A valódi logikai kapuk fizikai áramkörök**
  - A logikai bemenetek feldolgozása, a kimeneti jel megjelenése nem azonnal történik
  - A logikai kapuk időzítési paramétereit
  - $t_{pHL}$  terjedési idő  $H \rightarrow L$
  - $t_{pLH}$  terjedési idő  $L \rightarrow H$
  - $t_{pW}$  pulzusszélesség
  - $t_r$  felfutási idő
  - $t_f$  lefutási idő
    - A jelváltás minősége



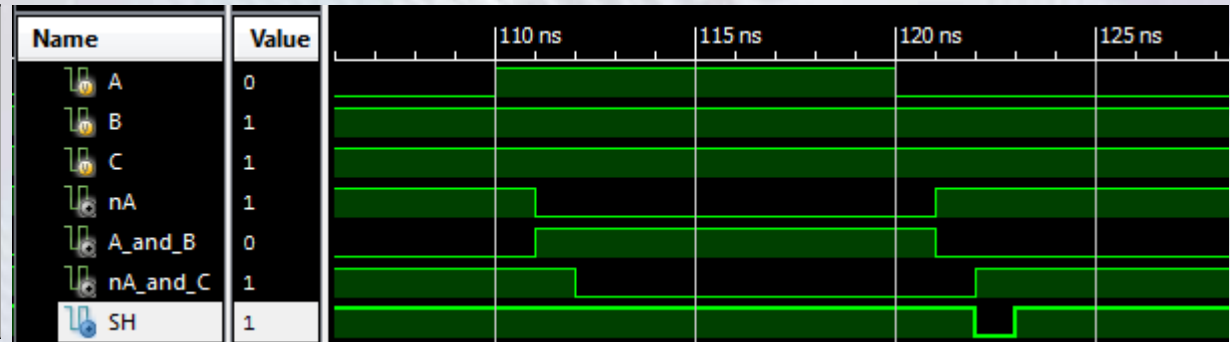
# Logikai kapuk időzítési jellemzői

- A logikai bemenetek feldolgozása, a kimeneti jel megjelenése nem azonnal történik
- A jelterjedési utak jellemzően eltérő késleltetéssel rendelkeznek (más típusú kapuk, nagyobb terhelés) → a kimeneteken tranziens (glitch) is felléphet
- A tranziensek veszélyét (hazárd) teljesen nem tudjuk elkerülni, ezért együtt élünk velük.
- Általános szabály: Kombinációs hálózatok kimenetén tetszőleges bemeneti változások után a kimeneti jel értéke a terjedési idő maximális értékéig bármikor változhat, azaz határozatlan lehet. Ezt funkcionális hazárdrnak nevezzük.
- Kombinációs hálózat kimenetét ezért nem használjuk működtető jelként (ahol a jel éle a hatásos esemény)

# Logikai kapuk időzítési jellemzői

- **Hazárdok egyszerűbb esetei: Statikus hazard**  
Egyszerű kétszintű kapuhálózatokban egyetlen bemeneti jel változására léphet fel
- **$SH = A * B + \neg A * C$  függvényben  $B=C=1$  és  $A$  változik**
  - A hálózat kimenetén a várt stabil 1 érték helyett a  $0 \rightarrow 1 \rightarrow 0$  léphet fel, attól függően, hogyan realizáltuk az áramkört

```
module hazard(  
  input A, B, C,  
  output SH, DH );  
  // minden művelet 1 ns késleltetésű  
  assign #1 nA = ~A;  
  assign #1 A_and_B = A & B;  
  assign #1 nA_and_C = nA & C;  
  assign #1 SH = A_and_B | nA_and_C;  
  assign #1 DH = SH & nA;  
endmodule
```

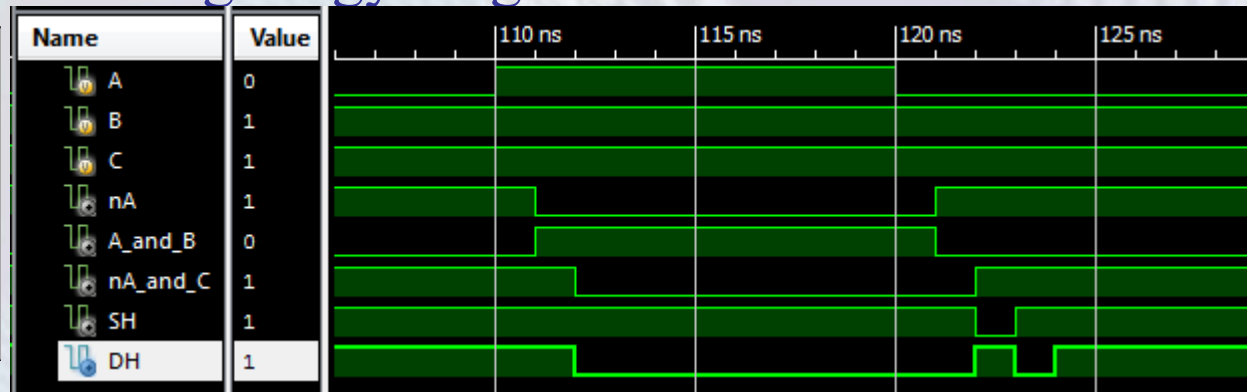


- Hasonlóan, más függvényben előfordulhat a  $1 \rightarrow 0 \rightarrow 1$  tranziens
- Védekezés: Redundáns lefedő hurkot alkalmazunk, a  $B * C$  hazardmentesítő szorzatot is beépítjük  $\rightarrow +1$  ÉS kapu
- **FPGA LUT 1 bemeneti változásra garantáltan hazardmentes**

# Logikai kapuk időzítési jellemzői

- **Hazárdok egyszerűbb esetei: Dinamikus hazard**  
A statikus hazard megléte miatt lép fel, több szintű (>2) kapuhálózatokban, egyetlen bemenet változása esetén
- **$DH = (A * B + /A * C) \& /A$ , ahol  $B=C=1$  és  $A$  változik**
- Előfordulásához szükséges egy meglévő statikus hazard

```
module hazard(  
  input A, B, C,  
  output SH, DH );  
  // minden művelet 1 ns késleltetésű  
  assign #1 nA = ~A;  
  assign #1 A_and_B = A & B;  
  assign #1 nA_and_C = nA & C;  
  assign #1 SH = A_and_B | nA_and_C;  
  assign #1 DH = SH & nA;  
endmodule
```



- A hálózat kimenetén a várt sima  $0 \rightarrow 1$  átmenet helyett a  $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$  jellegű tranzienses átmenet történik
- Más függvényben a  $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$  jellegű tranziens történhet.
- Védekezés: A statikus hazard megszüntetése a szükséges helyen

# Logikai kapuk időzítési jellemzői

- Korszerű nagysebességű áramkörökben nem csak a logikai elemek, hanem a huzalozások késleltetése sem elhanyagolható.
- A nagyszámú logikai jelre nem tudunk semmi garanciát adni arra, hogy ne változzon több jel egyszerre, vagy egymáshoz nagyon közel.
- Tehát szinte mindig a funkcionális hazard esete fordul elő, ami ellen viszont nem tudunk védekezni
- Ezért a hazardmentesítést ritkán tekintjük valódi tervezési célnak → Viszont a hazardra nem érzékeny áramköröket tervezünk, ez egyszerűbb
- Megjegyzés: Az univerzális logikai elemeket használó, MUX vagy LUT alapú memória táblázatos realizációknál a hazardmentesítés szempontja tehát nem merül fel.



# Digitális technika 3. EA vége