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

Digitális technika
VIMIAA01

Fehér Béla
BME MIT

Digitális Technika

- **Elméleti alapok**
  - Boole algebra
  - Logikai függvények
  - Kombinációs hálózatok
- **Specifikáció, reprezentáció, konverzió**
  - Alapelemek, kapuk, kétszintű hálózatok
  - A szorzatösszeg (SOP, Sum of Product) realizáció
  - Nem teljesen specifikált hálózatok
  - Minimalizálási eljárások
  - Többszintű hálózatok, a globális optimalizáció

Boole Algebra

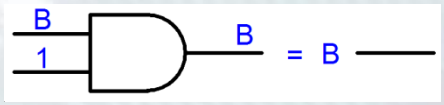
- **A Boole algebra**
  - A {B, +, *, ¯} négyes a Boole algebra, ahol
    - B az elemek (konstansok, változók) halmaza
    - a +, *, és ¯ pedig műveletek a B elemei között, additív, multiplikatív és ellentett képzés jelentéssel
  - Az algebra különböző alkalmazási területeken is megjelenik
    - Logikai algebra ÉS, VAGY, INV
    - Halmazalgebra ∩, ∪, ¯
    - Kapcsolási algebra: soros, párhuzamos, megszakító
  - A műveletek jelölései adott környezetekben eltérőek lehetnek

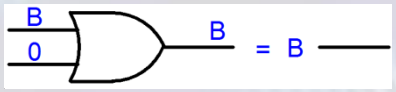
Boole Algebra

- **Axiómák**
  - B elemeire
    - A1: B=0, ha B≠1 A1d: B=1, ha B≠0 Kétértékű
    - A2: $\overline{0} = 1$ A2d: $\overline{1} = 0$ NOT, invertálás
  - A konstansműveletekre
    - A3: 0 * 0 = 0 A3d: 1 + 1 = 1 ÉS / VAGY
    - A4: 1 * 1 = 1 A4d: 0 + 0 = 0 ÉS / VAGY
    - A5: 0*1=1*0=0 A5d: 1+0=0+1=1 ÉS / VAGY
  - Dualitás elve: Ha a Boole kifejezésekben a 0 és 1 szimbólumokat és a * és + műveleteket felcseréljük, az állítások érvényesek maradnak (Fentiekben Aid jelöli az egyes axiómák duálisát)
  - Műveletvégzési sorrend: ( ) → $\overline{\ }$ → * → +

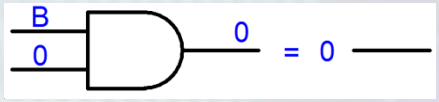
Boole Algebra

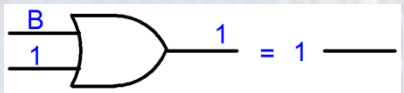
- **Egyetlen változóra vonatkozó tételek**
  - T1: B * 1 = B T1d: B + 0 = B Egységelemek



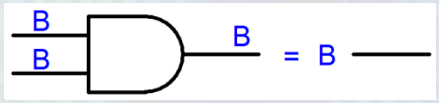


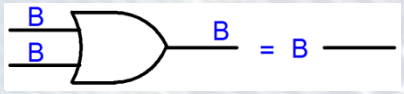
  - T2: B * 0 = 0 T2d: B + 1 = 1 Nullaelemek



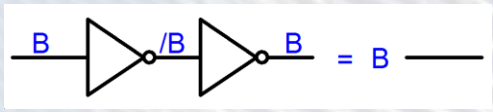


  - T3: B * B = B T3d: B + B = B Idempotencia

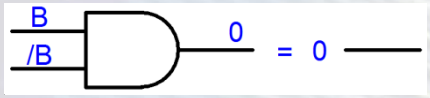


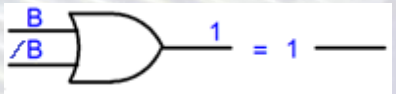


  - T4: $\overline{\overline{B}} = B$ Involúció



  - T5: $B * \overline{B} = 0$ T5d: $B + \overline{B} = 1$ Komplementer



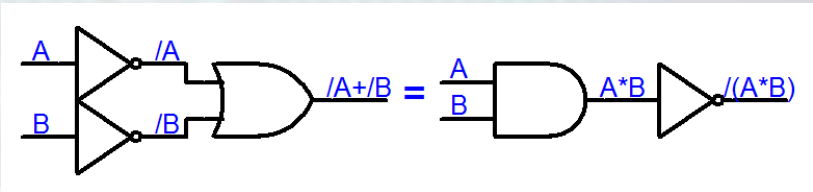


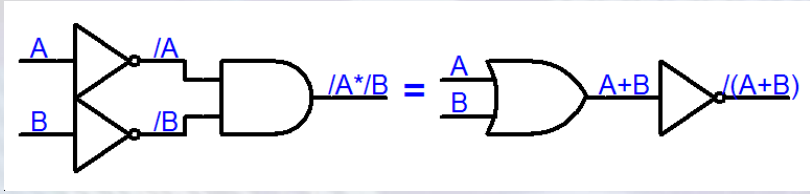
Boole Algebra

- **Két (vagy több) változóra vonatkozó tételek**
  - T6: B*C=C*B T6d: B+C=C+B Kommutativitás
  - T7: (B*C)*D=B*(C*D) T7d: (B+C)+D=B+(C+D) Asszociativitás
  - T8: (B*C)+(B*D)=B*(C+D) Disztributivitás
  - T8d: (B+C)*(B+D)=B+(C*D)
  - T9: B*(B+C)=B T9d: B+(B*D)=B Elnyelés
  - T10: (B*C)+(B* $\overline{C}$)=B T10d: (B+C)*(B+ $\overline{C}$)=B Összevonás
  - T11: (B*C)+($\overline{B}$*D)+(C*D) =B*C+($\overline{B}$*D) Konszenzus
  - T11d: (B+C)*($\overline{B}$ +D)*(C+D) =(B+C)*($\overline{B}$+D)
  - T12: $\overline{B_0 * B_1 * B2 \ldots.} = \overline{B_0} + \overline{B_1} + \overline{B_2} \ldots.$ De-Morgan tétel
  - T12d: $\overline{B_0 + B_1 + B2 \ldots.} = \overline{B_0} * \overline{B_1} * \overline{B_2} \ldots.$

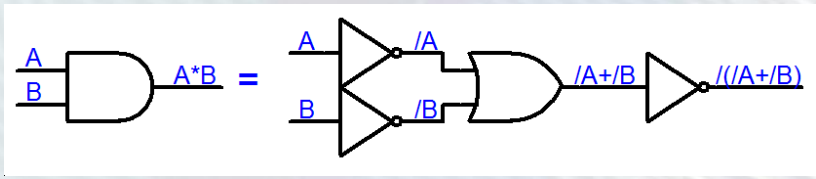
Boole Algebra: De Morgan tétel

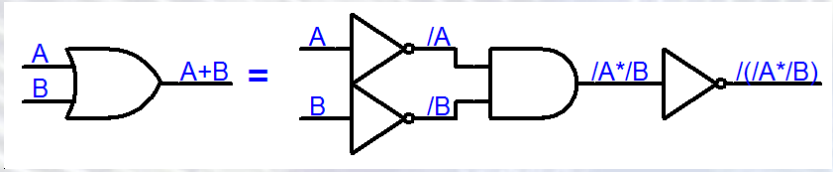
- **A De Morgan tétel 2 változóra, részletesebben**
  - $F = \overline{A} + \overline{B} = \overline{A * B}$ $\quad F = \overline{A} * \overline{B} = \overline{A + B}$

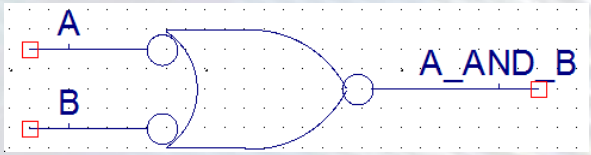


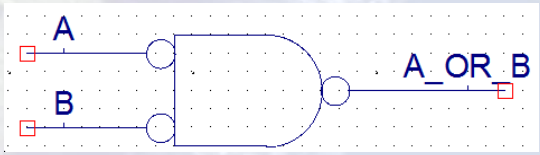


  - Más formában felírva, kifejezőbb:
  - $A * B = \overline{\overline{A} + \overline{B}}$ $\quad A + B = \overline{\overline{A} * \overline{B}}$









Logikai függvények

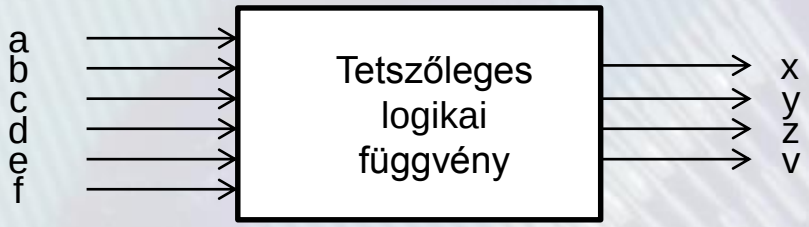
- **A logikai függvények: Boole változók között Boole algebra szabályai szerint értelmezett leképezések, pl.**

  Y = A + B*C + /A*/B*C + A*C

- **Logikai függvények leírásakor előforduló fontosabb kifejezések**
  - **Változók:** Az elsődleges logikai változók A,B,C
  - **Literálok:** A fenti változók előfordulásai ponált vagy negált értelemben, azaz A, /A, B, /B, C
  - **Szorzat (Product Term, PT):** Önálló literálok és ÉS kapcsolatú kifejezéseik, azaz A, B*C, /A*/B*C, A*C
  - **Szorzatösszeg (Sum-of-Product, SOP):** A kifejezések azon formája, ami szorzatok VAGY kapcsolatából áll.

Logikai függvények

- **Általános esetben a logikai függvényeket, mint bemenet-kimenet típusú memória mentes (emlékezet nélküli) leképezéseket tekintjük**



- **Logikai függvények a bemeneti változók értékének minden lehetséges kombinációjához a kimeneti változók 0 vagy 1 értékét rendelik**
- **A logikai függvényeket kombinációs hálózatokkal realizálhatjuk, mert a kimeneteik értéke egy adott pillanatban csak és kizárólag a bemeneti változók aktuális értékétől függ**

Elemi logikai függvények

- **Konstans logikai függvények**
  - 0 bemenet, $2^{2^0} = 2$ különböző függvény
    - $(F_0=0, F_1=1)$

BEM	KIM	
	F0	F1
nincs	0	1

- **Egyetlen bemenet**
  - 1 bemenet, $2^{2^1} = 4$ különböző függvény
  - $(F_0=0, F_1=A, F_2=/A, F_3=1)$

BEM	KIM			
A	F0	F1	F2	F3
0	0	0	1	1
1	0	1	0	1

- **Kettő bemenet**
  - 2 bemenet, $2^{2^2} = 16$ különböző függvény
  - $(F_0=0, F_1=A*B, F_3=A, F_C=/A, F_7=A+B, \ldots\ldots F_F=1)$

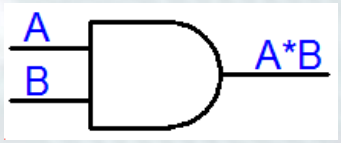
BEM		KIM															
A	B	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	A*B	A*/B	A	/A*B	B	A^B	A+B	/(A+B)	/(A^B)	/B	A+/B	/A	/A+B	/(A*B)	1

Elemi logikai függvények

- **Az elemi logikai függvények részletes specifikációja**

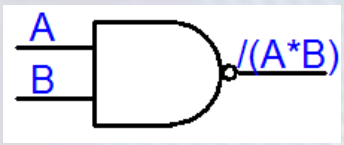
AND

BEM		KIM
A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1



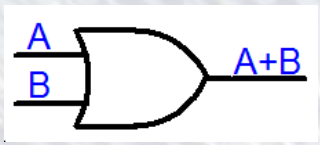
NAND

BEM		KIM
A	B	/(A*B)
0	0	1
0	1	1
1	0	1
1	1	0



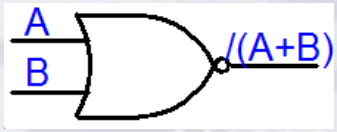
OR

BEM		KIM
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



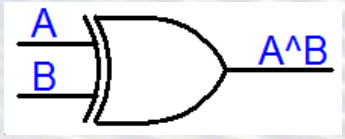
NOR

BEM		KIM
A	B	/(A+B)
0	0	1
0	1	0
1	0	0
1	1	0



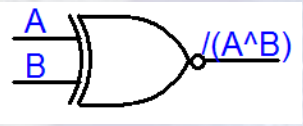
XOR

BEM		KIM
A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0



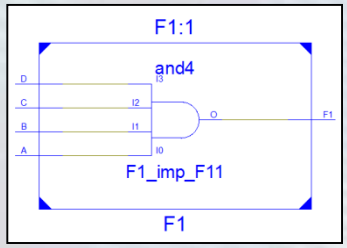
XNOR

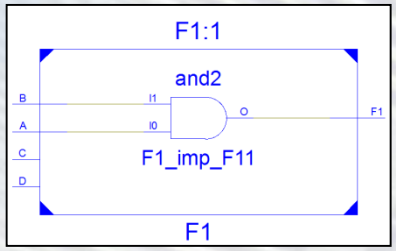
BEM		KIM
A	B	/(A^B)
0	0	1
0	1	0
1	0	0
1	1	1

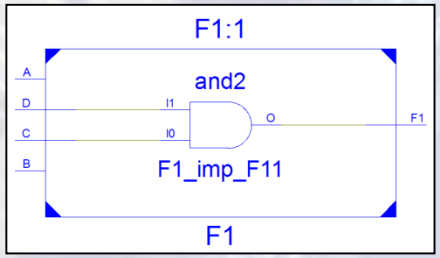


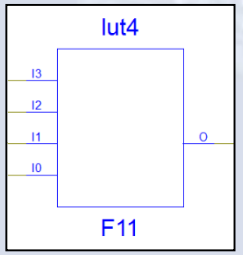
Elemi logikai függvények

- **Több bemenetű logikai függvények**
  - 3 bemenet, $2^{2^3} = 256$ különböző függvény
  - 4 bemenet, $2^{2^4} = 65536$ különböző függvény
    - Hamar kezelhetetlennek tűnő komplexitás
    - Ezek persze nagyrészben hasonló jellegű függvények, azaz
      - akár a bemenetek felcserélgetésével,
      - akár a bemenetek/kimenet invertálásával azonos formára hozhatók
    - Tehát pl. egy 4 bemenetű ÉS kapu a 4 bemeneti változó minden lehetséges ÉS kifejezését realizálni tudja, legfeljebb az F=C*D és F=A*B 2 bemenetű függvények esetén a nem használt bemenetekre „1” szintet kell adni.









Logikai függvények specifikációja

- **A logikai függvények különböző módokon specifikálhatók**
  - Egyértelmű, teljes specifikáció:
    - Igazságtábla
    - Algebrai normál alak
      - Diszjunktív normál alak, DNF, tehát SOP azaz Sum-of-Products
      - Konjuktív normál alak, CNF, tehát POS azaz Product-of-Sums
    - Karnaugh tábla, grafikus forma
  - Több formában is megadható, logikailag ekvivalens specifikációk
    - Szöveges specifikáció
    - Általános algebrai alak
    - Kapcsolási rajz
- **A specifikációk egymásba alakíthatók, konvertálhatók**

Logikai függvények specifikációja

- **A továbbiakban két egyszerű, 3 változós logikai függvényt fogunk példaként alkalmazni az alaptulajdonságok bemutatására**
- **Szöveges specifikációk**
  - F1: A függvény a bemeneti változók paritását jelzi. A kimenet jel értéke 1, ha a bemeneten páratlan számú aktív jel van, egyébként 0.
  - F2: A függvény egy többségi szavazást jelző áramkör. A kimenet jel értéke 1, ha bemenetei között több az aktív jel, mint az inaktív, egyébként 0.

Logikai függvények specifikációja

- **Egyértelmű, teljes specifikációk:**
  - Minden esetben explicit módon definiálják a bemeneti változók **minden** kombinációjához tartozó kimeneti értékeket, tehát az adott (igazságtáblázatos, grafikus K-tábla vagy kanonikus DNF/CNF algebrai) formában nem létezhet más, eltérő értelmű specifikáció

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

A \ B C	00	01	11	10
0	0	1	0	1
1	1	0	1	0

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 \ B C	00	01	11	10
0	0	0	1	0
1	0	1	1	1

- F1 = /A*/B*C+/A*B*/C+A*/B*/C+A*B*C = SOP(1,2,4,7)
- F2 = /A*B*C +A*/B*C +A*B*/C +A*B*C = SOP(3,5,6,7)

Logikai függvények specifikációja

- **Több formában is megadható, logikailag ekvivalens specifikációk**
  - Mindegyik ugyanazt a függvényt írja le, csak különböző formákban, értelmezésben
- **Szöveges specifikációk**
  - F1: A függvény a bemeneti változók paritását jelzi. Ha a bemeneten páratlan számú aktív jel van, a kimenet jel értéke 1, egyébként 0.
  - F1 alternatív: A függvény egy bináris 1 bites teljes összeadó (ai,bi,ci) összeg kimenete. Az összeadás művelet szabályai szerint, az összeg kimenet értéke: 0+0+0=0, 0+0+1=1, 0+1+1=0, és van átvitel, végül 1+1+1=1 és van átvitel.
  - F2: A függvény egy többségi szavazást jelző áramkör. Ha bemenetei között több az aktív jel, mint az inaktív, akkor a kimenet 1, különben 0.
  - F2 alternatív: A függvény egy bináris összeadó átvitel kimenete. Az összeadás művelet szerint akkor van átvitel, ha az (ai,bi,ci) bemenetek állapota a 0+1+1 vagy 1+1+1 feltételeknek felel meg.

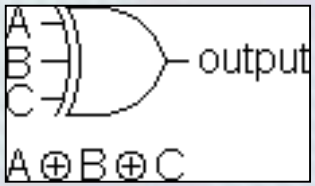
Logikai függvények specifikációja

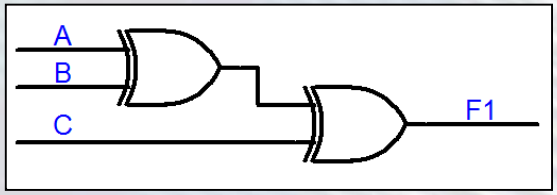
- **Több formában is megadható, logikailag ekvivalens specifikációk**
  - Mindegyik ugyanazt a függvényt írja le, csak különböző formákban, értelmezésben
- **Általános algebrai alakban**
  - F1 = /A*/B*C+/A*B*/C+A*/B*/C+A*B*C DNF, SOP
  - F1'=A XOR B XOR C XOR forma
  - F1 = (A+B+C)*(/A+B+/C)*(A+/B+/C)*(/A+/B+/C) CNF, POS

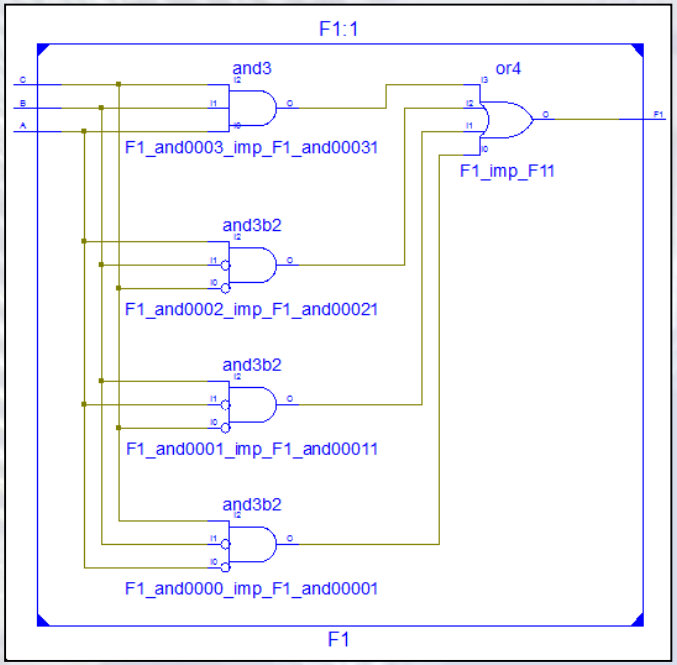
  - F2 = /A*B*C +A*/B*C +A*B*/C +A*B*C DNF, SOP
  - F2' = A*B + B*C + A*C SOP, de nem DNF
  - F2 = (A+B)*(B+C)*(A+C) CNF, POS

Logikai függvények specifikációja

- **Több formában is megadható, logikailag ekvivalens specifikációk**
  - Mindegyik ugyanazt a függvényt írja le, csak különböző formákban, értelmezésben
- **Kapcsolási rajzok az F1 függvényre**

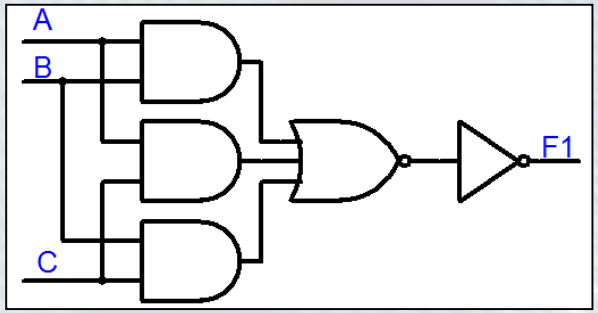


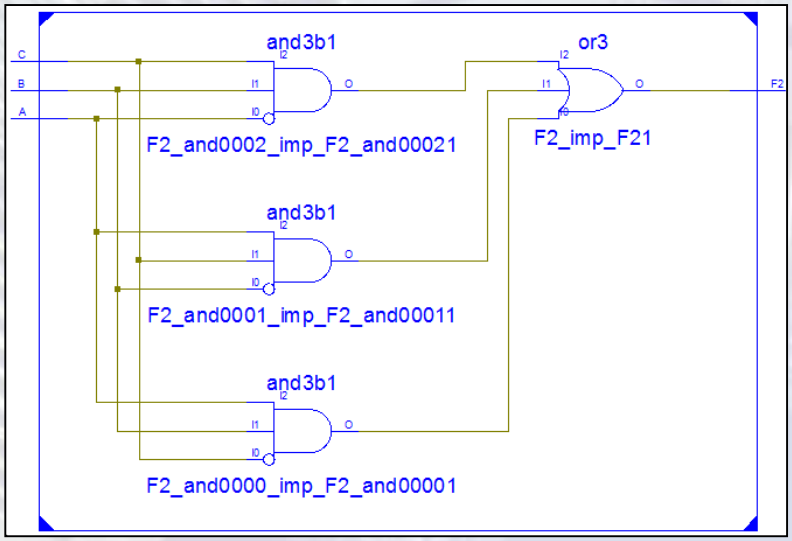




Logikai függvények specifikációja

- **Több formában is megadható, logikailag ekvivalens specifikációk**
  - Mindegyik ugyanazt a függvényt írja le, csak különböző formákban, értelmezésben
- **Kapcsolási rajzok az F2 függvényre**





Logikai függvények specifikációja

- **Nem teljesen specifikált függvények**
  - Gyakran a logikai függvények bemenetén nem fordul elő a bemeneti változók minden kombinációja
  - Vagy bizonyos kimeneti értékeket valamilyen okból nem használunk fel (Adatkimeneti bitek érvénytelen jelzés mellett)
- **Ezekben az esetekben a kimenetekhez tetszőleges érték rendelhető (akár 0, akár 1). Ezt közömbös, Don't Care (DC) bejegyzésnek nevezzük és x, -,* jelöléssel jelöljük.**
- **A tervezésnél ezeket a bejegyzéseket egyenként, egyedileg úgy választjuk meg, hogy a realizációnál minél egyszerűbb feltételeket kapjunk. (lásd később)**

Logikai függvények realizációja

- **A specifikációk alapján a tervező több, egymással logikailag ekvivalens, de egyéb paramétereiben jelentősen eltérő megoldás közül választhat**
- **Ez lehetőséget ad egyedi szempontok szerinti optimalizálásra. Tipikus optimalizálási szempontok:**
  - Legkevesebb alkatrész (jelentése technológia függő)
  - Leggyorsabb működés (legalább a kritikus jelre)
  - Meglévő „raktárkészletből” építkezés
- **Mindezek a célok a logikai függvények egyszerűsítésével, a redundanciák kihasználásával érhetők el. Ez a logikai függvények tervezésének tárgya.**

Logikai függvények realizációja

- **Kétszintű hálózat, SOP realizáció (POS is hasonló…)**
- **A logikai függvények egyszerűsítése a szorzat kifejezések „szomszédosságán” alapulnak.**
  - Ha két szorzat kifejezésben ugyanazon változók vannak és csak egyetlen változó szerepel ponált és negált értelemben, akkor az kiegyszerűsíthető

KIF*/A + KIF *A = KIF * (/A + A) = KIF * 1 = KIF

  - A függvény minimalizálása az ilyen (és további „egyszerű” algebrai) átalakításokon alapul.
  - Sok esetben átmeneti „bővítés” vezet jelentős redukcióra és az algoritmusok ezt ki is használják

Minimalizálás algebrai úton

- **Algebrai minimalizálás**
  - **F = /A*/B*/C + A*/B*/C + A*/B*C**
    - F = /B*/C*(/A+A) + A*/B*C
    - F = /B*/C(1) + A*/B*C
    - F = /B*/C + A*/B*C
  - **F = /A*/B*/C + A*/B*/C + A*/B*C**
    - F = /A*/B*/C + A*/B*/C + A*/B*/C + A*/B*C
    - F = /B*/C*(/A+A) + A*/B*(/C+C)
    - F = /B*/C*(1) + A*/B*(1)
    - F = /B*/C + A*/B
- **Láthatóan globális keresés szükséges**

Minimalizálás Karnaugh táblában

- **A Karnaugh tábla egy szemléletes eszköz a mintermek közötti kapcsolatok bemutatására**
- **Minterm:** Olyan szorzat, amelyben minden változó szerepel, ponált vagy negált értelemben,
  1 K-tábla cella = 1 Igazságtábla sor

	0	1
0	0	1
1	2	3

A \ B	0	1
0		
1		

	00	01	11	10
0	0	1	3	2
1	4	5	7	6

A \ B C	00	01	11	10
0				
1				

	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

A B \ C D	00	01	11	10
00				
01				
11				
10				

- A cél szomszédos cellákból 2^n méretű hurkok keresése, 1,2,3,…bemeneti változóra

Minimalizálás Karnaugh táblában

- **A szomszédosság esetei n=4 bemeneti változóra**
  - M= 1, 2, 4 mintermre F = CD+BD+AB/C+/A/B/C/D

	00	01	11	10
00	1 0	0 1	1 3	0 2
01	0 4	1 5	1 7	0 6
11	1 12	1 13	1 15	0 14
10	0 8	0 9	1 11	0 10

A B \ C D	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	1	1	0
10	0	0	1	0

BEMENETEK					KIM
INDX	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
A	1	0	1	0	0
B	1	0	1	1	1
C	1	1	0	0	1
D	1	1	0	1	1
E	1	1	1	0	0
F	1	1	1	1	1

- **Speciális szomszédosságok:**
  - pl. 0-2, 4-6, 12-14, 8-10 sor végi cellák
  - pl. 0-8, 1-9, 3-11, 2-10 oszlop cellák
  - pl. 0-2-10-8 sarokcellák

Minimalizálás Karnaugh táblában

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

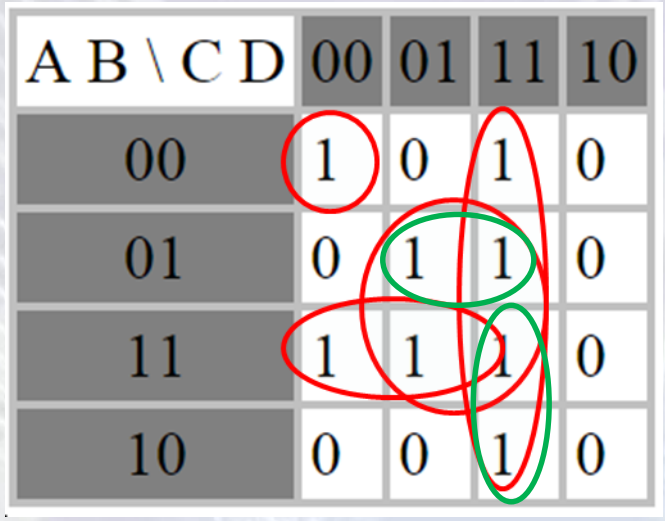
	00	01	11	10
0	1 0	0 1	0 3	0 2
1	1 4	1 5	0 7	0 6

A \ B C	00	01	11	10
0	1	0	0	0
1	1	1	0	0

- **A kiolvasható hurkok a (0,4) és a (4,5) mintermekből álló kettes hurkok.**
- **Így a minimális realizáció: F = /B*/C + A*/B**
- **Hatékony módszer n≤ 4 esetre.**
- Megjegyzés: Egy cella többszörös lefedése olyan, mintha az algebrai alakba többször beírtuk volna

Minimalizálási algoritmusok

- **Módszerek a logikai függvények minimális SOP realizációjának előállítására**
  - **Implikáns**: Olyan **szorzat** logikai függvény, amely része az eredeti függvénynek, azaz minden 1-es értéke szerepel abban is.
  - Prímimlikáns: Olyan implikáns, amely maximális méretű.
  - Az implikánsok **szorzatok**, tehát hurkok a K-táblában
  - Minden szorzat meghatároz egy implikánst, de csak a maximalizált méretű hurkok prímimplikánsok (pl. /ABD és ACD implikánsok)

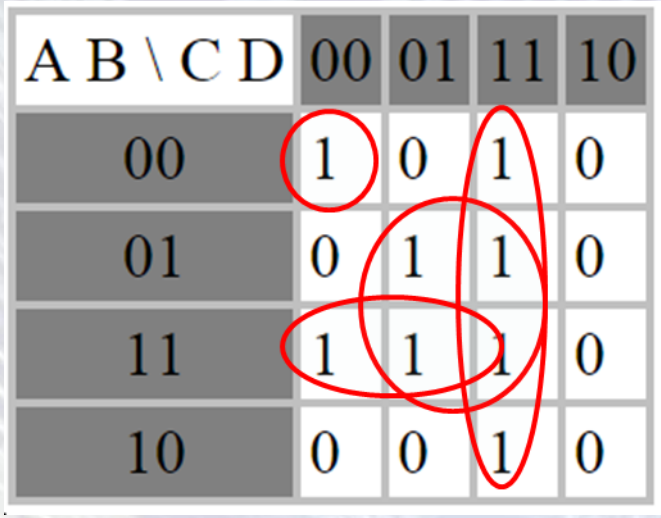


A B \ C D	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	1	1	0
10	0	0	1	0

- Az I_1= CD, I_2=BD I_3=AB/C és I_4=/A/B/C/D szorzatok pedig prímimplikánsok

Minimalizálási algoritmusok

- **Lényeges prímimplikáns: Olyan kifejezés, aminek van olyan „1”-es cellája, amit az eredeti függvényből csak ez tartalmaz. Tehát a teljes függvény a lényeges prímimplikáns nélkül nem realizálható.**
  - Ebben a példában mindegyik I_1= CD, I_2=BD, I_3=AB/C, I_4=/A/B/C/D prímimplikáns lényeges prímimplikáns, tehát **$F = I_1+I_2+I_3+I_4$**



A B \ C D	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	1	1	0
10	0	0	1	0

- **Általános esetben a realizáció tartalmazza a lényeges prímimplikánsokat és a maradékból egy olyan készletet, hogy minden „1”-es cella legalább 1-szer le legyen fedve.**

Minimalizálási algoritmusok

- **Példa 2:**
- Ebben a példában a prímimplikánsok a következők: I_1= /A/B, I_2=/AD, I_3=BD, I_4=AB/C
- **Lényeges prímimplikáns az I_1 (két sarok miatt), I_3(ABCD) és I_4 (AB/C/D) miatt.**
- Tehát a realizáció biztosan tartalmazza I_1, I_3 és I_4 lényeges prímimplikánsokat és az I_2 prímimplikáns már nem szükséges

A B \ C D	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	1	1	1	0
10	0	0	0	0

- **F’ = /A/B + AB/C + BD**

Minimalizálási algoritmusok

- **Példa 3: Nem teljesen specifikált függvény**
- Ebben a példában vannak DC, azaz tetszőlegesen megválasztható függvényértékek (* cellák). Az 5 ilyen cellából 3-at „1”-nek választunk, ezek kedvezően növelik az egyébként is szükséges lefedő hurkokat
- A maradék 2-t nem használjuk és így 0 értéket reprezentálnak a kimeneten.
- **Minden prímimplikáns lényeges, I_1 = /A/B/C, I_2 = /AD, I_3 = A/D**
- A DC bejegyzések kihasználásával lényegesen egyszerűbb a realizáció
- F = /A/B/C + /AD + A/D
  - XOR függvénnyel F’ = /A/B/C + A XOR D

A B \ C D	00	01	11	10
00	1	1	1	0
01	0	*	1	0
11	1	*	0	*
10	1	0	*	*

Minimalizálási algoritmusok

- **Sok bemeneti változóra ezek a kézi módszerek (algebrai, K-tábla) már nem megfelelőek**
- **Léteznek számítógépes algoritmusok**
  - Algoritmus
    - Véges számú lépésben megoldja a problémát
    - Véges idő alatt leáll egy valamilyen megoldással
  - Algoritmusok minősítése
    - Optimális: Megtalálja a legjobb megoldást
    - Hatékony: Egy jó megoldást talál gyorsan
  - Kimerítő algoritmus
    - Megtalálja az optimális megoldást
    - Esetleg hosszú időbe telik
  - Heurisztikus algoritmus
    - Hatékony, gyors
    - Jó megoldást talál, de nem feltétlenül az optimálisat

Minimalizálási algoritmusok

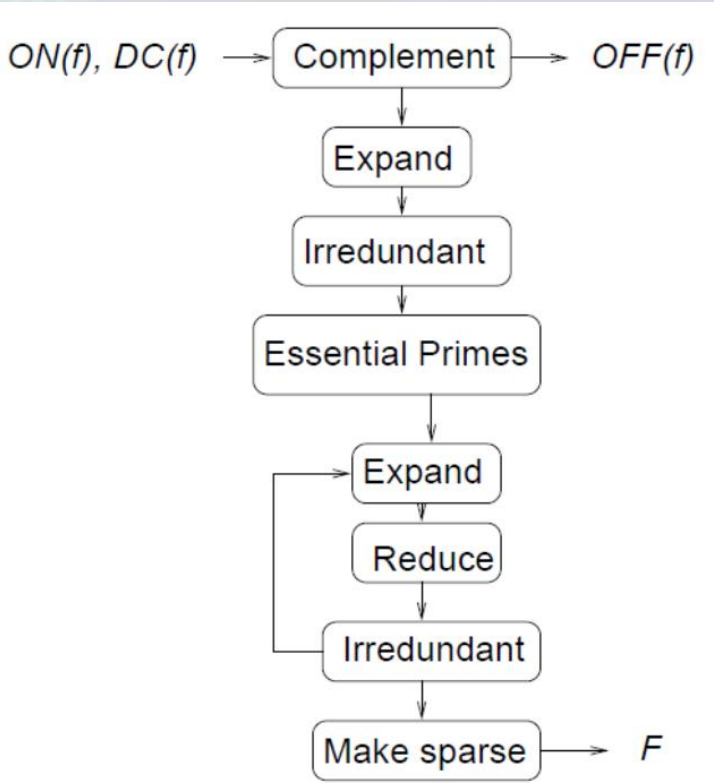
- **Quine-McCluskey (50-es évek)**
  - Kimerítő teljes algoritmus
  - Az eddig megismert lépéseket ismétli
  - Először generálja az aktív mintermeket
  - Ezek alapján implikánsokat állít elő
  - Megkeresi az összes prímimplikánst
    - Figyelembe veszi a Don’t Care jelzéseket is)
  - Kiválasztja a lényeges prímimplikánsokat
  - Összeválogatja a még szükséges prímimlikánsokat, úgy, hogy a legkisebb költségű megoldást adják
  - Sajnos ez a kimerítő teljes keresés a számítógépes végrehajtás mellett is túl hosszú futásidőt igényel
    - Az optimális megoldás költsége túl nagy

Minimalizálási algoritmusok

- **Espresso**
  - Heurisztikus algoritmus
    - Találjunk elfogadható idő alatt egy elegendően jó megoldást (nem kell feltétlenül az optimális megoldás)
    - Lokális keresési algoritmus
    - Nem generáljuk az összes mintermet és prímimplikánst
    - Viszont lépésenként finomítjuk (esetleg visszalépve) az addig elért lefedési készletet
    - Egyértelmű minimumhely esetén biztosan megtalálja, több lokális minimum esetén új feltételekkel újraindulva esetleg javítható a megoldás megbízhatósága

Minimalizálási algoritmusok

- **Espresso**
  - A jelenleg legnépszerűbb algoritmus
  - Kiemelkedően hatékony a logikai függvények minimalizálásában
  - Három fő lépés
    - Bővítés: a mintermek/implikánsok méretének növelési kísérlete
    - Redukció: A bővítés fordítottja, kisebb hurkok kialakítása, finomabb hurkok, jobb teljes lefedés érdekében
    - Többszörös lefedés kiváltása: az implikánsok olyan készlete, ahol egy-egy 1-est minél kevesebb hurok fed le
    - Ezek iteratív ismétlése, sokszor visszalépve



- **Szinte minden logikai hálózatgeneráló szintézis program ezt használja**

Második előadás vége