Kombinációs hálózatok egyszerűsítése

Cél: A specifikációval megadott KH ***legolcsóbb megvalósítás***a

Specifikáció: szöveges specifikáció, Boole algebrai kifejezés, normál alak, igazságtábla, de az egyszerűsítés előtt ***igazságtáblává kell alakítani, ha nem abban van megadva***.

Mi a legolcsóbb?
Megvalósítási ***környezettől függ*** (alkatrész).
Pl. Ha önálló kapuk állnának rendelkezésre, a legkevesebb kapu és legkevesebb bemenet szám lehet a ***célfüggvény***.

Diszkrét IC-kből (egy IC több kaput is tartalmazhat, hogy hányat, azt a benne levő kapuk bemenetszáma is befolyásolja), ***a legkevesebb IC*** felhasználása lehet a célfüggvény.

Programozható logika esetén pl. a ***legkevesebb erőforrás*** (logikai cella) felhasználása lehet a célfüggvény.

A példáinkban az egyszerű számíthatósága miatt a a ***legkevesebb kapu bemenet szám***ra törekszünk.

Hogyan számolható a bemenetek száma?
Pl. a megvalósított kapcsolási rajz alapján.
2 szintű megvalósítás esetén felrajzolás nélkül is egyszerűen számolható.

Legyen a megvalósítandó függvény:

F=A.B./C + /A.B.D + C./B

Minden termet egy annyi bemenetű kapu valósít meg, amennyi a változóinak száma (Nv). Annyi kapu szükséges, ahány term van a függvényben (Nt).

A fenti esetben 2 db 3 bemenetű kapu és 1 db 2 bementű kapuval valósítható meg a függvény, így az 1. szinten összesen 8 bemenet van.
Nv1+Nv2+Nv3=8, ahol Nvi az egyes termek változószáma.

A 2. szinten egy annyi bemenetű kapu szükséges, ahány termet tartalmaz a függvény (Nt).

A fenti esetben itt egy 3 bemenetű OR kapu szükséges. (Nt=3)

Így az összes bemenetek száma SUM(Nvi)+Nt=11.

Kétszintű hálózattal megvalósított kombinációs hálózat egyszerűsítése

A következőkben először 2 szintű KH-ok egyszerűsítésével foglalkozunk.

Az egyszerűsítés elve, az 1 változóban különböző (1 Hamming távolságú) termek megkeresése, majd az eltérő változó elhagyása az

A.B + A./B = A.(B + /B) = A (diszjunktív alak esetén) vagy az
(A + B)(A + /B)=A + A./B + A.B =A (konjunktív alak esetén) azonosság alapján

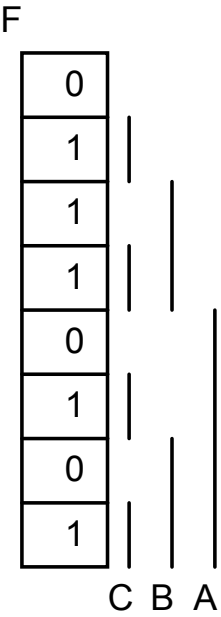
Az 1 Hamming távolságú termek megkeresésére kézi (ZH-ban jól használható) ***grafikus módszer***ként az un. ***Karnaugh táblát*** használjuk.

A ***Karnaugh tábla*** egy speciálisan kialakított igazságtábla.

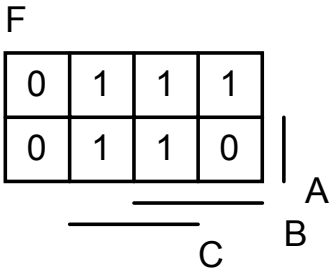
Hagyományos igazságtábla

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

"Peremezett" igazságtábla



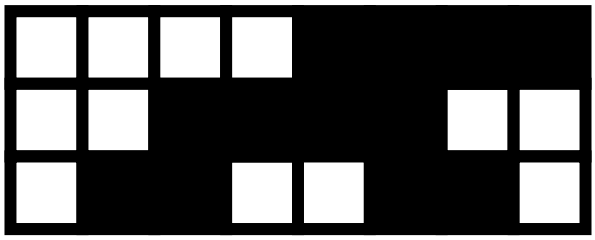
Karnaugh tábla



A Karnaugh tábla peremezése olyan, hogy az egymás melletti sorok ill. egymás alatti oszlopok egy változóban térnek el, az un. Gray kód szerint.

A Gray kód

A Gray kód egy pozíciókód. Az alábbi ábra egy vízszintes szakaszt 8 részre osztva kódol, Gray kóddal.



Tükrözéses módszerrel lehet kisebb bitszámú pozíció kódból nagyobbat készíteni.

a. Induljunk ki a 1 bites Gray kódból (most a kisebb helyfoglalás miatt majd az egymás alatti számok adják a kódot:

b. Folytassuk a kódok felírását fordított sorrendben (tükrözés).

c. A régi kódok elé írjunk 0-át, a tükrözöttek elé pedig 1-et.

c. **0011**

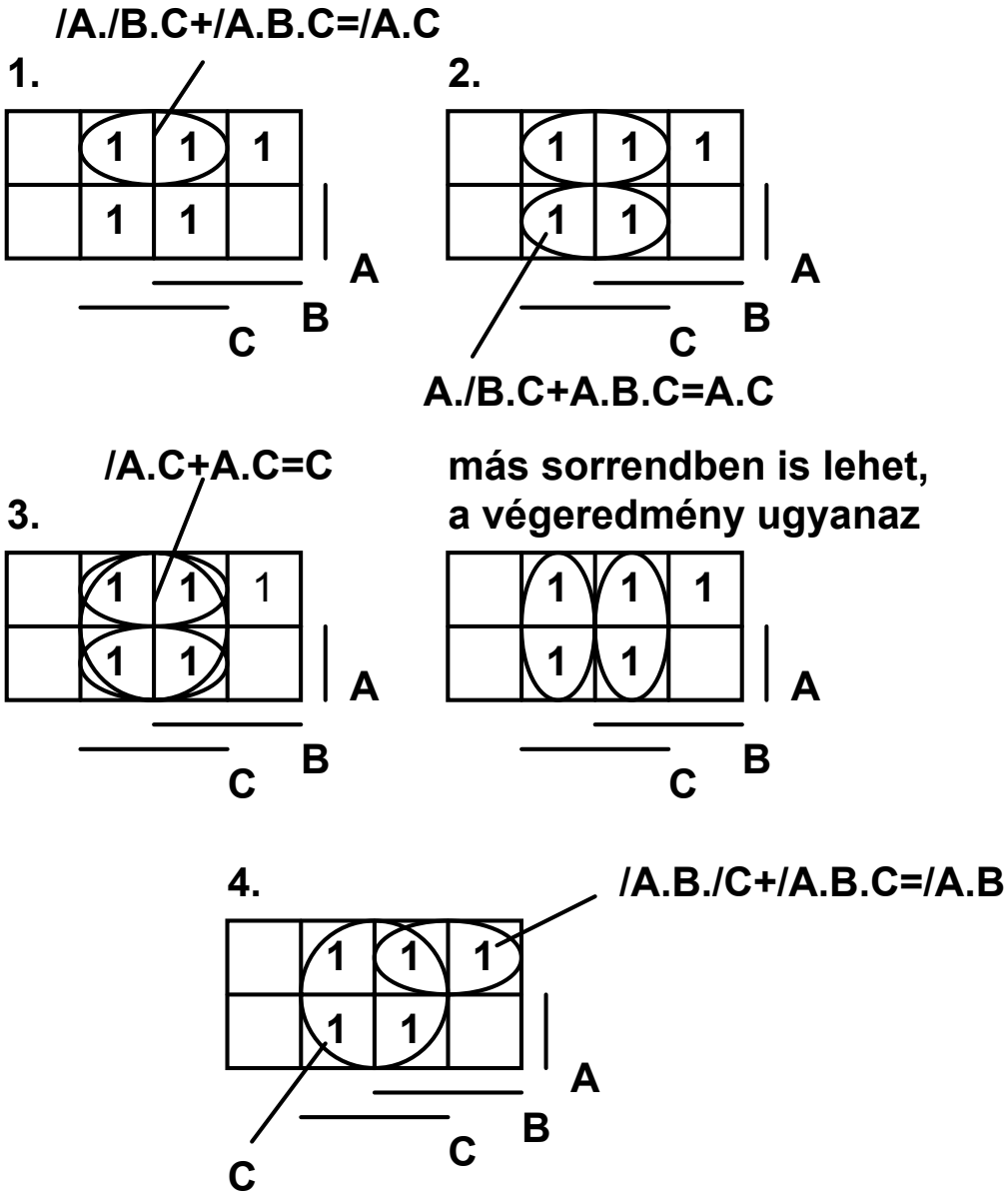
a. 01 b. 01*10* 0110

Az egymás alatti számok adják a kódszavakat:
(00, 01, 11, 10)

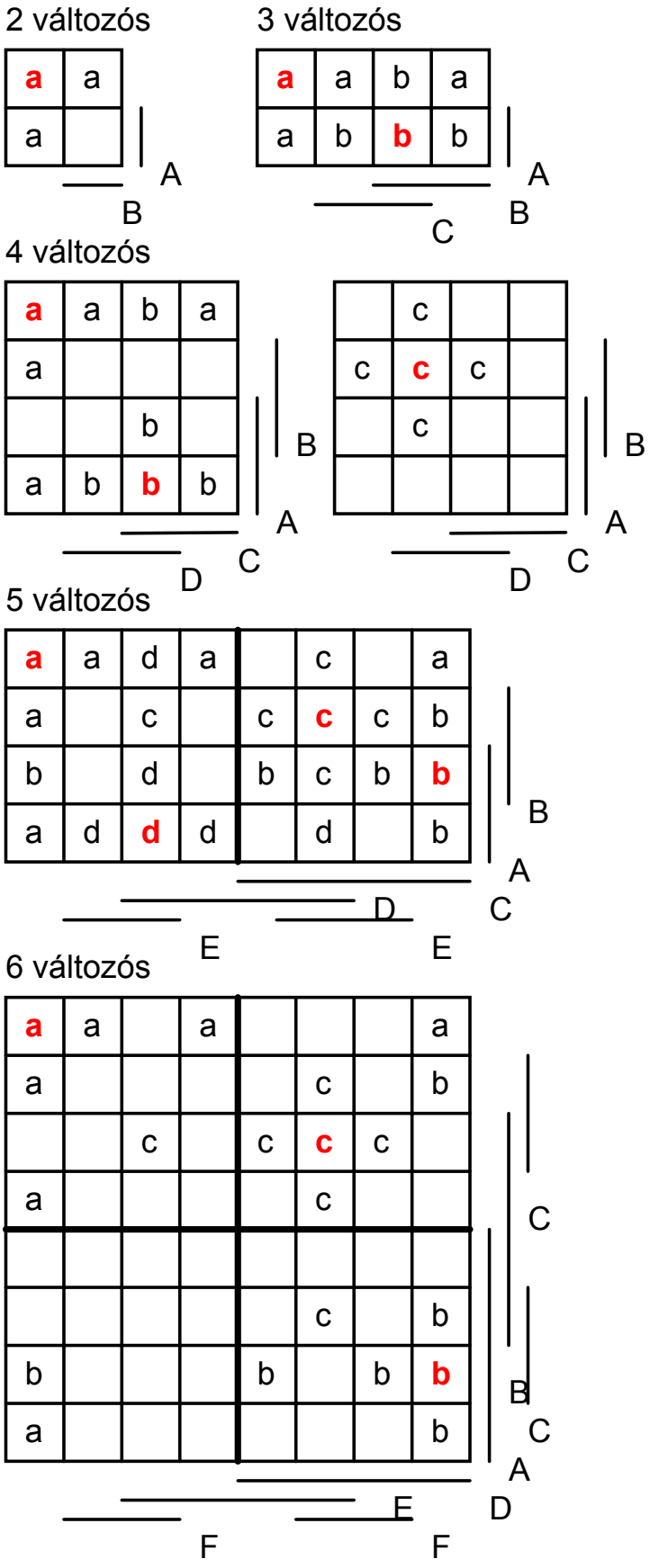
Hasonlóan folytatva a 3 bites Gray kód:
0000 1111
0011 *1100*
0110 *0110*

(000, 001, 011, 010, 110, 111, 101, 100)

A Gray kód szerint peremezett Karnaugh táblán a szomszédos rubrikák (a széleket is szomszédosnak tekintve) 1 változóban különböznek. Ezért a szomszédos termek megkeresése kis gyakorlás után “szemre” viszonylag egyszerű.
Az alábbi K-táblán az üresen hagyott rubrikákhoz 0 érték tartozik.
Az egyszerűsítés lépésenként elvégezve:



A 2-6 változós Karnaugh táblák rubrikáinak szomszé-dossági viszonyai



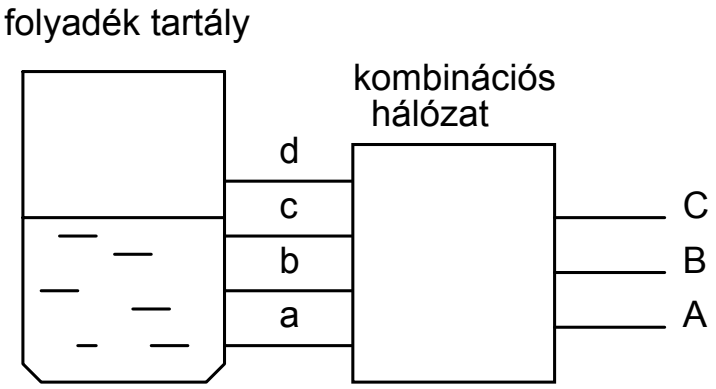
Az igazságtáblában szereplehetnek ***közömbös bejegyzések*** is! Jelölése: **x** vagy **-**

- A bemeneti kombináció, amelynél a közömbös bejegyzés szerepel, ***nem fordul elő a bemeneten***

- A bemeneti kombináció előfordulhat a bemeneten, de ***a kimenetre csatlakozó logika nem veszi figyelembe***

A közömbös bejegyzések egyszerűsítést tesznek lehetővé, mivel a közömbös bejegyzéseket az egyszerűsítéshez legmegfelelőbb logikai értékkel vehetjük figyelembe.

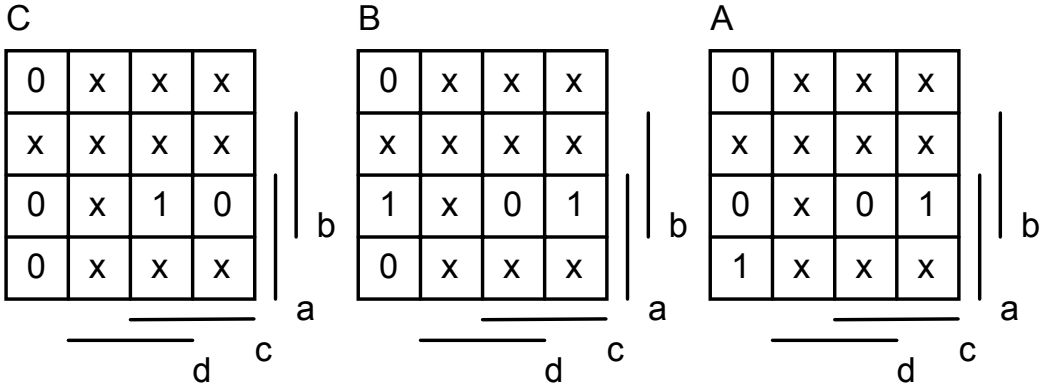
Pl. Folyadékszint mérő.
A készülék az a-d bemenetein érzékel és a CBA kimenetén jelzi a folyadék állását.



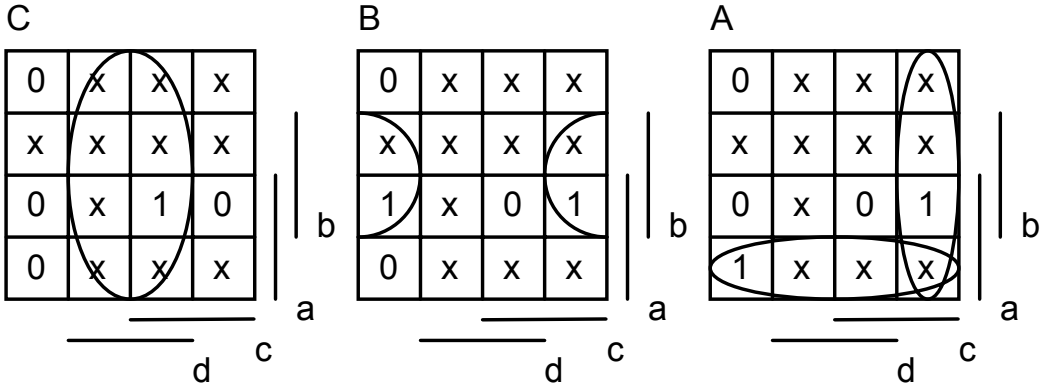
A folyadékszint mérő igazságtáblája:

a	b	c	d	C	B	A
0	0	0	0	0	0	0
0	0	0	1	-	-	-
0	0	1	0	-	-	-
0	0	1	1	-	-	-
0	1	0	0	-	-	-
0	1	0	1	-	-	-
0	1	1	0	-	-	-
0	1	1	1	-	-	-
1	0	0	0	0	0	1
1	0	0	1	-	-	-
1	0	1	0	-	-	-
1	0	1	1	-	-	-
1	1	0	0	0	1	0
1	1	0	1	-	-	-
1	1	1	0	0	1	1
1	1	1	1	1	0	0

A folyadékszint mérő Karnaugh táblái:



A diszjunktív alakban keresett megoldás szerint egyszerűsített Karnaugh táblák:

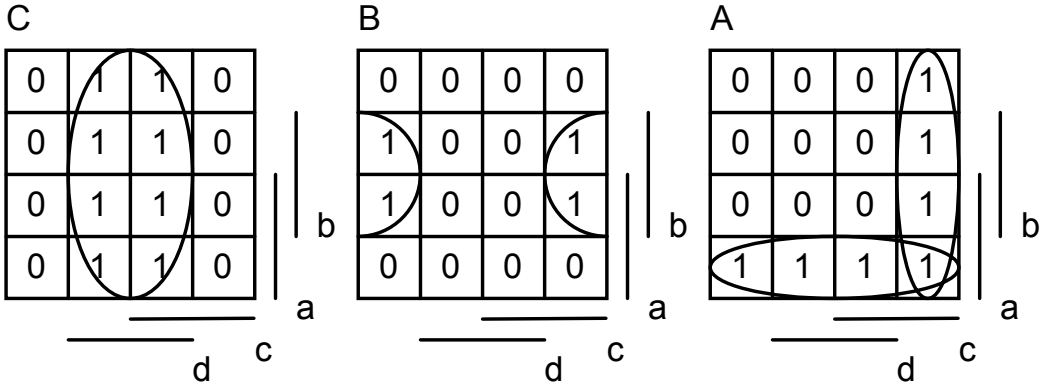


C = d
1 bemenet

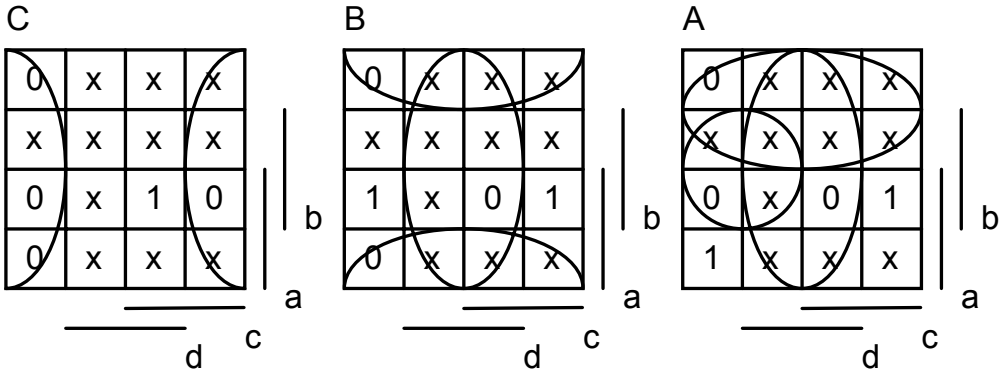
B = b./d
2 bemenet

A = a./b + c./d
6 bemenet

Az egyszerűsített függvény igazságtáblája:



A konjunktív alakban keresett megoldás szerint egyszerűsített Karnaugh táblák:

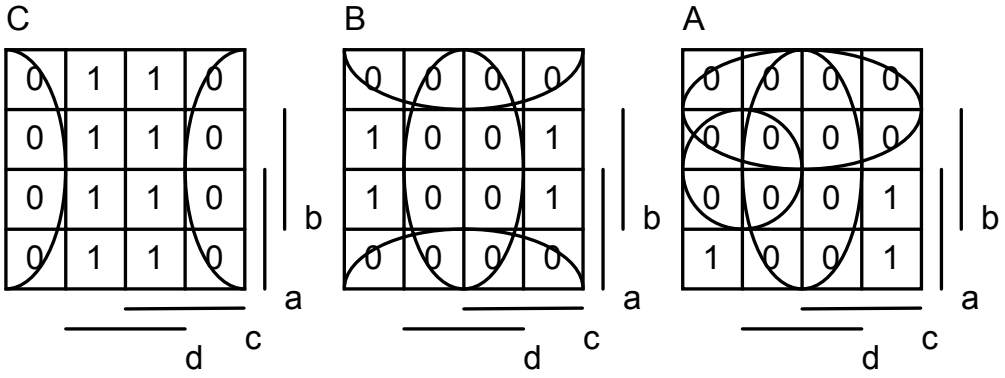


C = d
1 bemenet

B = b./d
2 bemenet

A = a./d.(/b+c)
5 bemenet

Az egyszerűsített függvény igazságtáblája:



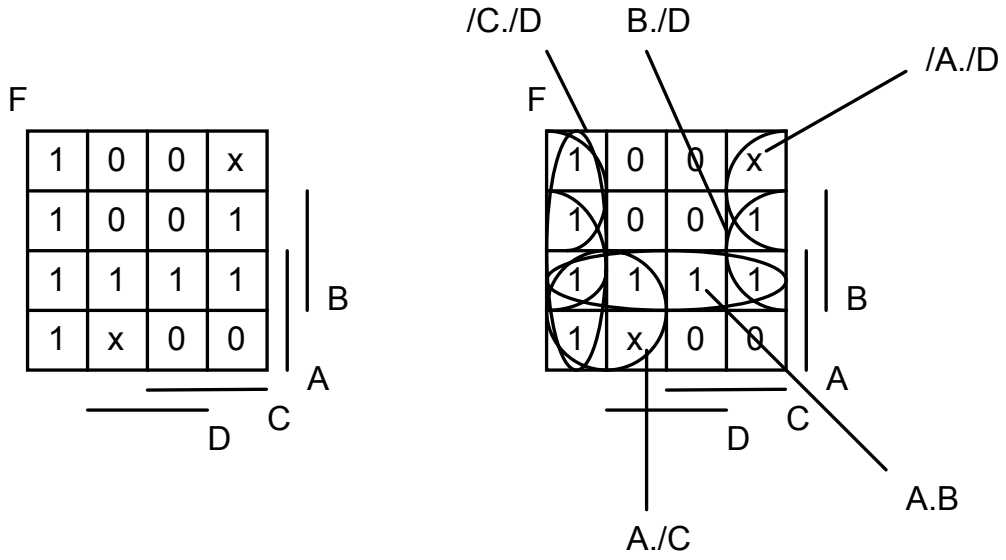
A konjunktív alakú megoldás az egyszerűbb.

A minimalizálás teljes algoritmusa:

1. Keressük meg az összes prímimplikánst

Prímimplikáns: olyan term, amelyből nem hagyható el több változó (nem egyszerűsíthető tovább).

Példa:



A prímimplikánsok:

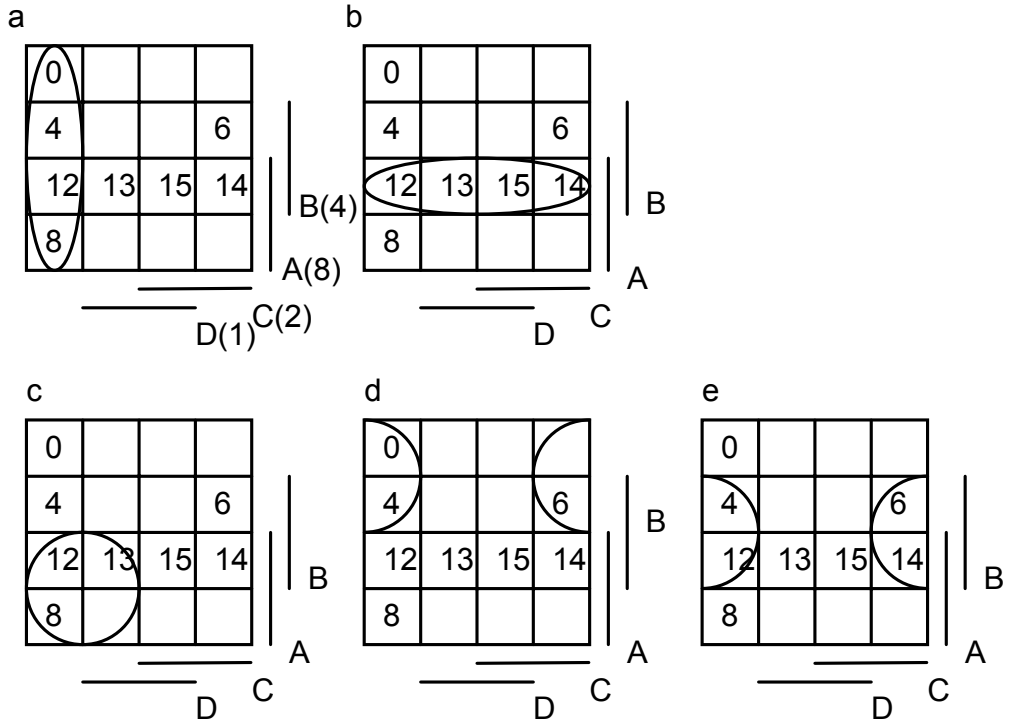
a: /C./D b: A.B c: A./C d: /A./D
e: B./D

2. ***Válasszuk ki a prímimplikánsok közül a legkevesebbet, amely <u>lefedi</u> az igazságtábla összes mintermjét*** (***maxtermjét*** konjunktív megvalósítás esetén).

A fedés megvalósításához az ún. ***prímimplikáns tábla*** ad segítséget. Egyszerűbb esetekben ennélkül "szemre" is elvégezhető az egyszerűsítés.

A tábla oszlopait a minterm (maxterm) sorszámokhoz rendeljük. Itt a közömbös bejegyzésekhez tartozó sorszámok nem szerepelnek, mivel azokat nem kell lefedni.

Az alábbi Karnaugh táblákban külön-külön feltüntettük, az egyes prímimplikánsok által lefedett mintermeket.



Prímimplikáns tábla

	0	4	6	8	12	13	14	**15**
a	+	+		+	+			
b					+	+	+	+
c				+	+	+		
d	+	+	+					
e		+	+		+		+	

Lehetnek olyan mintermek, amelyeket csak egy prímimplikáns fed le. Az ilyen a mintermeket ***megkülönböztetett mintermek***nek nevezik.

Azon prímimplikánsokat, melyek legalább egy megkülönböztetett mintermet tartalmaznak, ***lényeges prímimplikáns***oknak hívjuk.

Példánkban a 15-ös minterm megkülönböztetett minterm, a b prímimplikáns pedig lényeges prímimplikáns.

A lényeges prímimplikánsok feltétlenül szükségesek a fedéshez, mivel más nem tudja helyettesíteni őket.

A továbbiakban azt kell kideríteni, hogy a lényeges prímimplikánsokon kívül még melyek szükségesek feltétlenül a fedéshez.

Prímimplikáns tábla

	0	4	6	8	12	13	14	**15**
a	+	+		+	+			
b					+	+	+	+
c				+	+	+		
d	+	+	+					
e		+	+		+		+	
	a+d	a+d+e	d+e	a+c	+	+	+	+

A prímimplikáns tábla utolsó sorában feltüntettük, hogy egy-egy oszlophoz tartozó mintermet mely prímimplikánsok képesek lefedni, ill. hogy a lényeges prímimplikáns miket fed le. Pl. a 0-ás lefedéséhez az a-ra vagy a d-re van szükség.

Mivel az összes lényeges prímimplikáns által le nem fedett prímimplikánst le kell fedni, ezt az alábbi Boole kifejezéssel fogalmazhatjuk meg:

$S=(a+d)(a+d+e)(d+e)(a+c)$

Ez egy csupa ponált változóból álló kifejezés, melyet ezért könnyű egyszerűsíteni, az elnyelési szabály alkalmazásával.

$(a+d)(a+d+e) = a + d$
$(d+e)(a+c) = ad + cd + ae + ce$
$(a + d)(ad + cd + ae + ce) = \underline{ad + ae + cd}$

A lényeges prímimplikánst is figyelembe véve 3 fedés létezik:

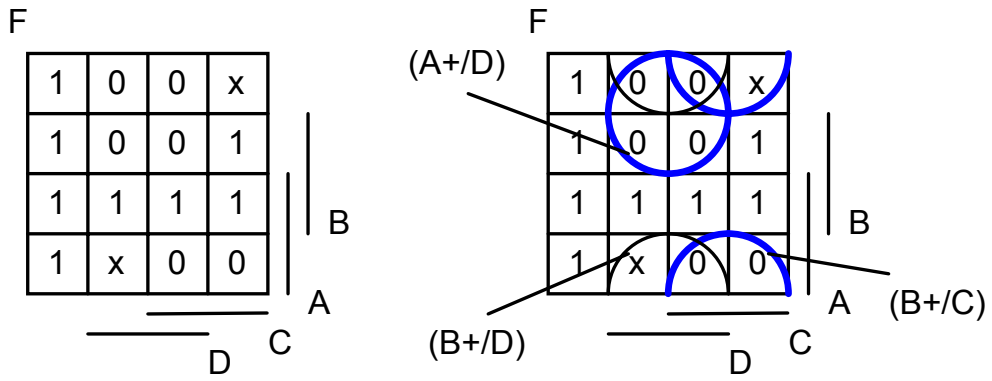
abd + abe + bcd

Az ezekhez tartozó függvények:

abd: F = /C./D + A.B +/A./D
abe: F = /C./D + A.B + B./D
bcd: F = A.B + A./C+ /A./D

Ezek itt egyformán minimálisak, a kapu bemenetek száma 9.
Általában a különböző megoldásokhoz különböző kapu bemenetszám tartozhat, ekkor a megoldások közül ki kell választani egy legegyszerűbbet.

Konjunktív alakban keresve a megoldást:



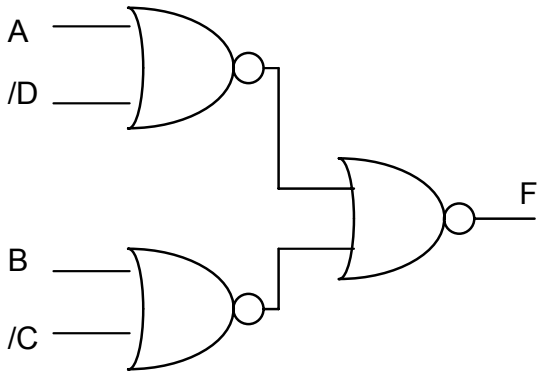
A prímimplikánsok:
a. (A+/D) b. (B+/C) d. (B+/D)

Lefedési tábla nélkül, “szemre” is látható, hogy az a és b prímimplikáns lefedi az összes maxtermet.

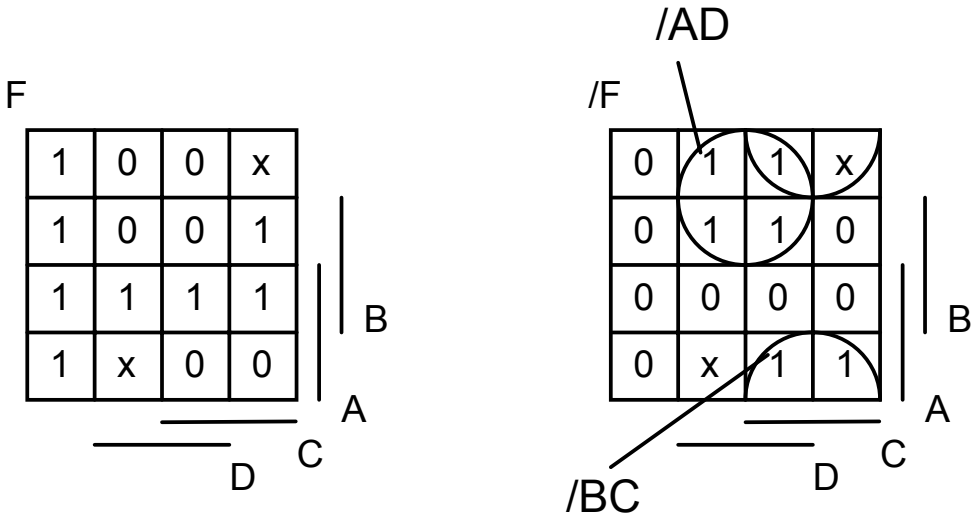
F = (A + /D)(B + /C)

Ehhez 6 kapu bemenet szükséges, így egyszerűbb mint a diszjunktív alakú megoldás.

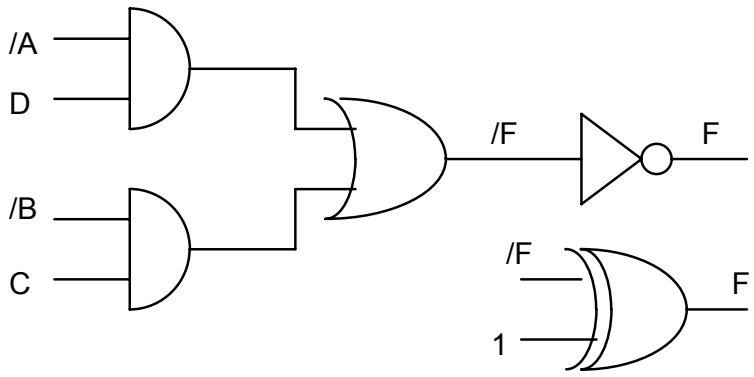
A kapcsolási rajz homogén NOR megvalósítása:



Ha meg van kötve, hogy a megoldást milyen alakban kell keresni és a másik alakban lenne az egyszerűbb, akkor a függvény negáltját lehet megvalósítani, majd negálni. (Például a PLD-ben diszjunktív megvalósítás lehetséges, és a függvényt egy EXOR kapuval meg lehet invertálni.)
Az előbbi példa esetén /F-at diszjunktív alakban megvalósítva majd invertálva:



F = /(/A.D + /B.C)



Számjegyes minimalizálás (Quine-McCluskey módszer)

A logikai függvények minterm (maxterm) indexét úgy képezzük, hogy mintermben ABC sorrendben szereplő változókhoz ponált esetben 1-et, a negált esetben 0-át rendelünk, majd az így kapott bináris szám decimálissá (10-es számrendszrűvé) alakítjuk.
Pl. /A./B./C.D

0 0 0 1

A ***prímimplikánsok*** meghatározásakor a szomszédos termeket kell megkeresni és összevonni. Pl. /A./B./C./D + /A./B./C.D = /A./B./C./D

A számítógépes módszer a mintermeket a bináris indexükkel reprezentálja. A szomszédos (egy változóban ellentétes előjelű) termek indexének bináris formája 1 bitben tér el. Az a változó esik ki, ahol az eltérés van.

Pl. /A./B./C./D + /A./B./C.D = /A./B./C

0000 0001 000-

Tehát az egy bitben eltérő számokat kell megkeresni, és az eltérő bit helyhez rendelt változót elhagyni (itt ezt "-„-szal jelöltük)

A szomszédos termeket gyorsabban meg lehet találni, ha először a bináris minterm indexeket a bennük levő 1-esek száma (*bináris súlyuk*)

alapján sorbarendezzük. Ekkor már csak a súlyuk szerint egymás melletti indexeket kell egymáshoz hasoníltani.

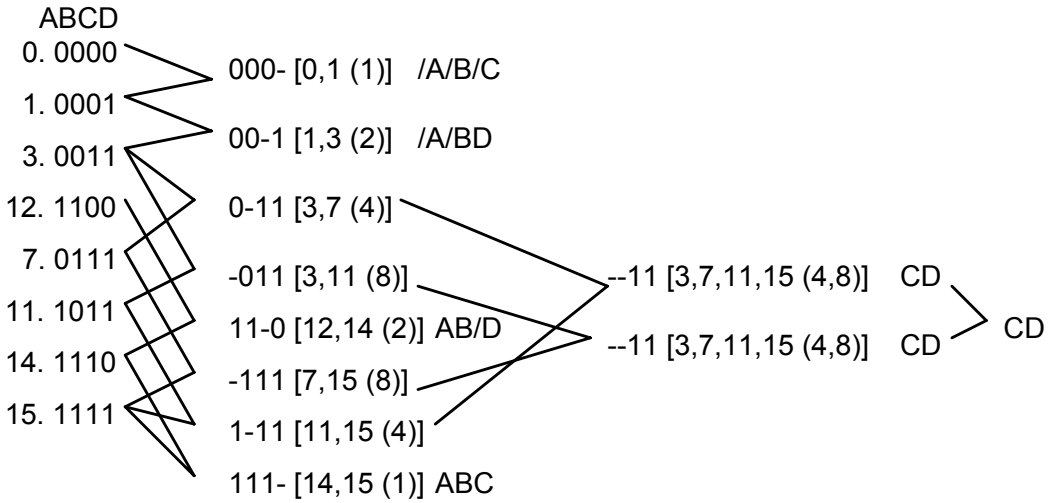
Az algoritmust egy mintapéldán mutatjuk be.

f=/A/B/C/D + /A/B/CD + /A/BCD + AB/C/D + /ABCD +A/BCD +ABC/D +ABCD =

$m_{0000} + m_{0001} + m_{0011} + m_{1100} + m_{0111} + m_{1011} + m_{1110} + m_{1111}$

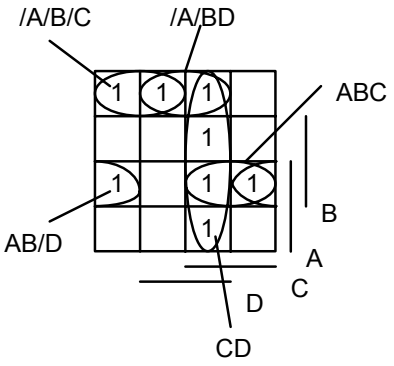
A továbbiakban csak az indexet írjuk le.

Bináris súly szerint rendezve:



Az “[ ]”-jelek közé téve az Arató könyvben használt jelöléseket is feltüntettük.

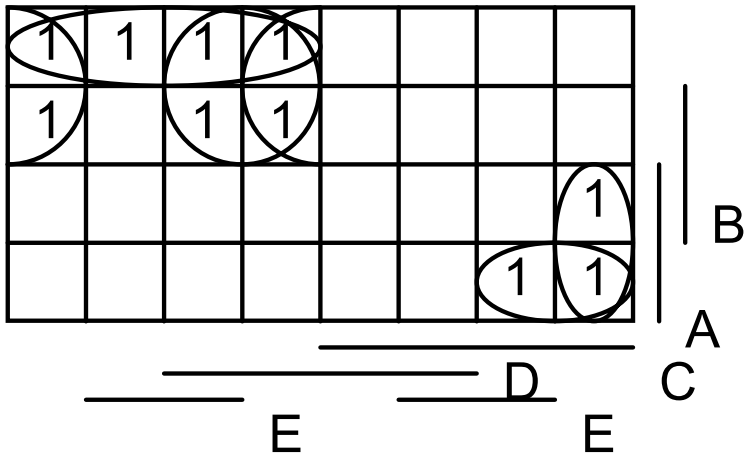
Karnaugh táblával ugyanezt az eredményt kapjuk:



f=CD +ABC+/A/BD+/A/B/C + AB/D

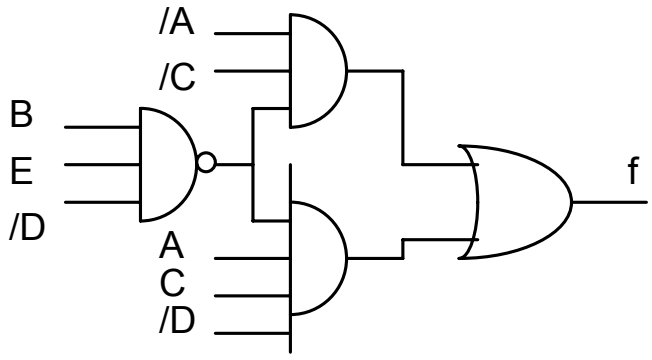
Logikai fügvények realizálása többszintű hálózattal

1. ***Algebrai átalakítással hozzuk többszintű hálózattal realizálható alakba a függvényt***



A kétszintű minimális diszjunktív alak 22 bemenettel valósítható meg.

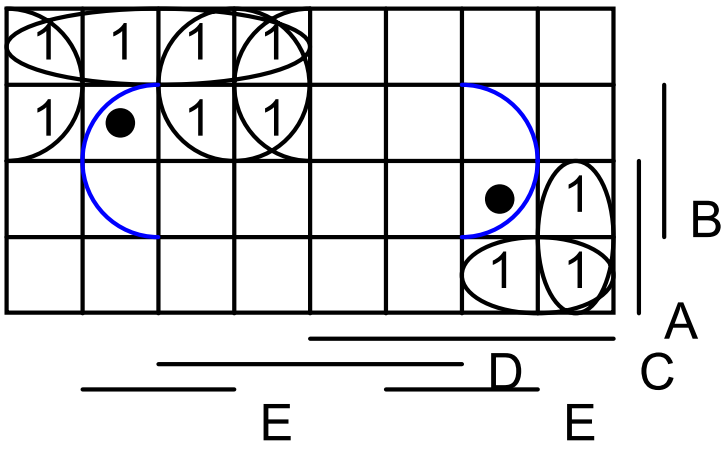
f =/A/B/C+/A/C/E+/A/CD+AC/D/E+A/BC/D=
=/A/C(/B+/E+D) + AC/D(/B+/E)=
=/A/C(/B+/E+D) + AC/D(/B+/E +**D**)=
=/A./C***./(BE/D)*** + AC/D***/(BE/D)***



A 3 szintű megvalósításhoz 12 kapu bemenet szükséges.

A többszintű realizálás ***sokszor egyszerűbb áramkört eredményez***, viszont ***hosszabb lesz a hálózat jelterjedési ideje***.

2. *Kitiltás módszere többszintű hálózat létrehozására*



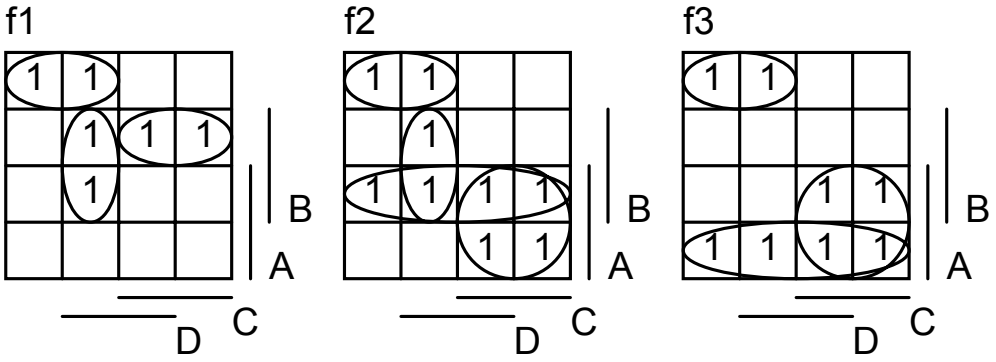
a. Ha a ponttal jelölt helyeken 1 lenne, sokkal egyszerűbb megoldás adódna. Egyszerűsítsük a függvény, mintha ez lenne az eredeti függvény.
f' = /A./C + A.C./D
b. Megfelelő (lehetőleg egyszerű) K kitiltó függvénnyel megszorozva f'-t biztosítsuk, hogy a fügvény újra az eredeti igazságtábla valósuljon meg.

f = f'.K K=**/(B/DE)**
f = **/(B/DE)** (/A/C + AC/D). =
= /A./C.**/(B/DE)** + AC/D**/(B/DE)**

Ez most ugyanarra az eredményre vezetett, mint a Boole algebrai átalakítás.

Többkimenetű kombinációs hálózat minimalizálásának elve

Legyen egy 3 kimenetű 4 változós logikai függvényünk, melynek igazságtáblája az alábbi.



f1 = **/A./B./C** + /A.B.C+B./C.D

f2 = **/A./B./C** + A.B + B./C.D+A.C

f3 = **/A./B./C** + A./B + A.C

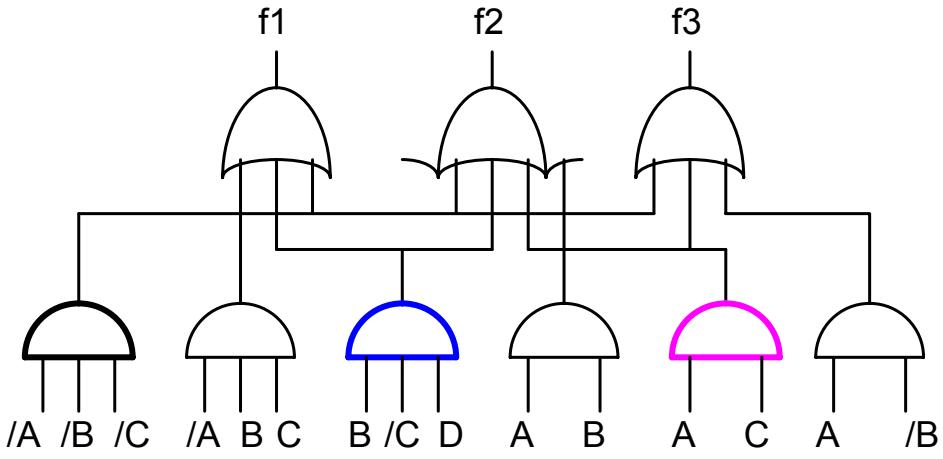
Elv: a több függvényben is előforduló (azonos) prímimplikánsokat csak egyszer valósítjuk meg.

f1,f2,f3: **/A./B./C**

f1f2: B./C.D

f1f3: --------

f2f3: A.C



Az eredeti 36 kapubemenet helyett csak 25 kell.