

Digitális technika I

Dr. Göllei Attila, Dr. Holczinger Tibor, Dr. Vörösházi Zsolt



2014

A tananyag a TÁMOP-4.1.2.A/1-11/1-2011-0104 "A felsőfokú informatikai oktatás minőségének fejlesztése, modernizációja" keretében készült. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Egyetem



Előszó

A Pannon Egyetem Műszaki Informatikai Karán tanuló mérnök informatikus alapszakos hallgatók eddig más oktatási intézmények által kiadott jegyzetekből, a kereskedelemben kapható tankönyvekből valamint az Internetről letöltött anyagokból tanulhatták a digitális technikát (BME: [ARATÓP], vagy SZE: [KERESZTP]). Hallgatóink bár ezekből is tökéletesen elsajátíthatták a tantárgy elméleti részeit, azonban nincs közöttük egyetlen olyan sem, amely a karunkon folyó képzés követelményeihez és tematikájához teljes mértékben igazodna. Jelen jegyzet ezt a hiányt hivatott pótolni.

A jegyzet készítése során fontos célul tűztük ki, hogy a meglévő előadás vázlatokra, fóliákra épülő egységes, jól hasznosítható oktatási segédanyag készüljön a képzésben lévő Digitális Technika I. - II., és a Digitális Áramkörök c. tárgyakhoz, amelyet nagy hallgatói létszámmal oktatunk a Pannon Egyetem veszprémi, és nagykanizsai képzési helyein, valamint a Szegedi Tudomány Egyetemen.

A jegyzet mind mérnök informatikus, mind villamosmérnök szakos hallgatók számára korszerű, konkrét, elméleti és gyakorlati ismereteket tartalmaz a digitális áramkörök és rendszerek tervezésének elsajátításához. A témaköröket a nagy terjedelem miatt két jegyzetben publikáljuk: a Digitális technika I. c. segédlet főként a digitális áramkörök bevezetésével, alapfogalmaival és a kombinációs logikai hálózatok szintjével foglalkozik, míg a rá épülő Digitális technika II. c. segédlet a sorrendi logikai hálózatok elméleti és gyakorlati hátterét, valamint azok komplexebb tervezési lépéseit ismerteti.

Tartalomjegyzék

Előszó	2
Rövidítések	7
1. Alapfogalmak, analóg és digitális jel, analóg és digitális rendszer, logikai rendszer.....	9
Bevezetés	9
Analóg és digitális jel.....	11
Az analóg jel.....	11
A digitális jel.....	11
Analóg és digitális rendszerek	12
2. Logikai függvények megadása	14
Logikai feladat megadása szöveges formában	14
Logikai feladat megadása időfüggvény alakjában	14
Logikai feladat megadása táblázatos módon	14
Logikai feladat megadása grafikus módon	15
Logikai feladat megadása algebrai alakban.....	15
Logikai függvények kanonikus alakjai.....	15
Az elvi logikai rajz vagy logikai vázlat.....	16
Egyváltozós logikai függvények.....	17
Többváltozós logikai függvények	17
Logikai függvények grafikus ábrázolása, Veitch-Karnaugh diagram	19
3. Logikai áramkörök TTL-CMOS rendszerek	23
Logikai áramkörrendszerek	23
DDL dióda-dióda logika	23
DTL dióda-tranzisztor logika	23
TTL tranzisztor-tranzisztor logika	23
ECL Emitter csatolású logika	24
TTL kapuáramkörök karakterisztikája	25
CMOS logikai áramkörök.....	26
4. Logikai áramkörök fizikai jellemzői	29
Logikai áramkörök dinamikus jellemzői	29
Felfutási és lefutási idő.....	29
Késleltetési, vagy terjedési idő	29
Terhelhetőség (kimeneti), Fan-out	30
Egység-terhelés (bemeneti), Fan-in	30
Egyéb jellemzők	31
A TTL és CMOS elemcsaládok összehasonlítása	31
Speciális kimeneti kapcsolatok TTL áramköröknél	31
Nyitott kollektoros és háromállapotú TTL áramkörök	32

Nyitott kollektoros kimenetek.....	32
Háromállapotú (Tri-State, Three-State) kimenetek.....	33
5. Logikai függvények egyszerűsítése.....	35
Algebrai módszer.....	35
Boole algebra.....	35
Kifejtési módszer.....	38
Grafikus módszer.....	38
Logikai függvény grafikus minimalizálása.....	39
Implikánsok kiolvasása a Karnaugh táblából.....	40
Hurkok felrajzolásának szabályai.....	44
Prímimplikánsok meghatározása.....	47
Egyszerűsített alak megadása.....	49
Példa.....	50
Nem teljesen specifikált hálózatok minimalizálása.....	53
A módszer korlátai.....	58
Számjegyes minimalizálás.....	58
Prímimplikánsok meghatározása.....	59
Egyszerűsített alak megadása.....	62
Példa.....	63
Nem teljesen specifikált hálózat minimalizálása.....	66
6. Többkimenetű logikai függvények egyszerűsítése.....	69
Grafikus minimalizálás.....	69
Nem teljesen specifikált hálózatok minimalizálása.....	75
Számjegyes minimalizálás.....	77
Nem teljesen specifikált hálózatok minimalizálása.....	80
7. Szimmetrikus logikai függvények.....	82
Bevezetés, definíciók.....	82
Műveletek, transzformációk szimmetrikus függvényekkel.....	83
1.) Logikai összeadás.....	83
2.) Logikai szorzás.....	83
3.) Szimmetrikus függvény negáltja.....	83
4.) Szimmetrikus függvény változóinak negáltja.....	84
Szimmetrikus függvények egyszerűsítési szabályai.....	86
Szimmetrikussá tehető F függvény minimalizálása egy példán keresztül TSH – Teljesen Specifikált Hálózat esetén.....	87
Szimmetrikussá tehető F függvény minimalizálása „bináris súlyok” eljárásának használatával egy példán keresztül bemutatva.....	93
8. Két- és többszintű NEM-ÉS és NEM-VAGY hálózatok tervezése.....	99

Funkcionálisan teljes rendszerek.....	99
NÉV rendszer	99
NEM-ÉS rendszer.....	99
NEM-VAGY rendszer.....	100
Kétszintű hálózat tervezése NEM-ÉS és NEM-VAGY rendszerben.....	101
Többszintű hálózat tervezése NEM-ÉS és NEM-VAGY rendszerben	104
Többszintű NEM-ÉS hálózatok	105
Többszintű NEM-VAGY hálózatok	106
9. Leggyakrabban használt kombinációs hálózatok.....	109
Kódolók	109
Dekódolók	111
7 szegmenses kijelző dekóder-meghajtó áramkörök.....	114
Multiplexerek – demultiplexerek.....	119
Multiplexerek.....	120
Demultiplexerek.....	124
Összeadó áramkörök.....	126
Félösszeadó (half-adder: HA).....	126
Teljes összeadó (full-adder: FA).....	127
Paritás-generáló és figyelő áramkörök	129
Digitális komparátorok.....	130
10. Kombinációs hálózatok megvalósítása Memóriával, illetve Programozható Logikai Eszközökkel	134
Memóriák: bevezetés.....	134
Kombinációs hálózatok és a memóriák kapcsolata	135
Kombinációs hálózatok és a Programozható logikai áramkörök kapcsolata.....	141
Miért lehet fontos a programozható logikai eszközök alkalmazása?	141
Programozható logikai áramkörök.....	142
Makrocellás típusok	143
FPGA kapu-áramkörök	147
11. Hazárd jelenségek	149
Hazárd jelenségek háttere: bevezetés	149
Hazárdok kialakulása:.....	149
a.) Jelterjedési (propagation delay) vagy „megszólalási” késleltetés:	149
b.) Összeköttetési (interconnection delay) késleltetés:.....	149
a.) Statikus hazárd.....	150
b.) Dinamikus hazárd.....	153
c.) Funkcionális hazárd	156
További példák hazárd jelenségekre:.....	158

Irodalomjegyzék164

Rövidítések

Kifejezés	Angol	Magyar
BCD	Binary Coded Decimal	Binárisan Kódolt Decimális számok
CAD	Computer-Aided Design	Számítógéppel segített tervezés
CG	Carry Propagate	Átvitel terjesztő egység
CMOS	Complementer Metal-Oxid Semiconductor	Komplementer-technológiájú fém-oxid félvezetők
CP	Carry Generate	Átvitel generáló egység
DDL	Diode-Diode Logic	Dióda-dióda logika
DMC	Differential Manchester Coding	Differenciális Manchester kódolás
DNF	Disjunctive Normal Form	Diszjunktív Normál Alak
DTL	Diode-Transistor Logic	Dióda-transzosztor logika
ECL	Emitter Coupled Logic	Emitter-csatolású logika
EDA	Electronic-design Automation	Elektronikai tervezés-automatizálás
FA	Full Adder	Teljes Összeadó
FF	Flip-Flop	Élvezérelt tároló
FPGA	Field Programmable Gate Arrays	Újraprogramozható Kapuáramkörök
GAL	Generic Array Logic	Generikus Tömb Logika
HA	Half Adder	Fél Összeadó
IC	Integrated Circuit	Integrált Áramkör
KH	Combinatorial Network Logic	Kombinációs Hálózat
KNF	Conjunctive Normal Form	Konjunktív Normál Alak
KV	Karnaugh-Veicht	Karnaugh Veicht diagramm (Karnaugh tábla)
LS	Low-power Shottkey	Kis-teljesítményű Shottkey-dióda
MC	Manchester Coding	Manchester kódolás
NRZ	Non-Return to Zero	Nullára nem-visszatérő (kódolás)
NTSH	Non-totally Specified Network	Nem-Teljesen Specifikált Hálózat
OC	Open-collector	Nyílt-kollektoros (logikai kapu)
PAL	Programmable AND Logic	Programozható ÉS logikai eszköz
PCB	Printed Circuit Board	Nyomtatott Áramkör
PLA	Programmable Logic Array	Programozható Logikai Tömb
PLD	Programmable Logic Devices	Programozható Logikai Eszközök
QM	Quine McCluskey	Quine McCluskey (Számjegyes minimalizálás)
RCA	Ripple Carry Adder	Átvitelkezelő (több-bites) Összeadó
RZ	Return to Zero	Nullára visszatérő (kódolás)
SH	Sequential Network Logic	Sorrendi (szekvenciális) Hálózat
TS	Tri-State	Három-állapotú (0, 1, Z) kimenetek

TSH Totally Specified Network
TTL Transistor-Transistor Logic

Teljesen Specifikált Hálózat
Tranzisztor-Tranzisztor Logika

1. Alapfogalmak, analóg és digitális jel, analóg és digitális rendszer, logikai rendszer

Bevezetés

A digitális technika hosszú évek óta zajló folyamatos előretöréséről és alkalmazási területének szinte végtelen tárházáról talán nem kell beszélnünk azoknak, akik kezükbe veszik ezt a jegyzetet. Ennek ellenére mégis érdemes néhány, talán érdekességnek vagy kuriózumnak számító dolgot megemlíteni és némi történeti áttekintést adni a digitális jelek, jelzések kialakulásáról és felhasználási jelentőségének folyamatos elterjedéséről.

A „digitális jel” kifejezést legtöbbször valamilyen elektronikus, tudományos, talán kissé misztikus kategóriákhoz társítják, pedig a szó szoros értelmében vett „digitális” nagyon sok, egyszerű dolog is lehet. A kifejezés nem csak a modern elektronikában és a számítástechnikában használt bináris elektronikus digitális rendszerekkel áll összefüggésben. A digitális rendszerek valójában nagyon ősi eredetűek és sem az nem szükséges, hogy binárisak, sem az, hogy elektronikusak legyenek.

A kifejezés a latin *digitus (ujj)* szóból származik. Az ujjakat lehet használni a diszkrét számolás céljára, hiszen az ujjainkkal diszkrét egész számokkal tudunk számolni.

A digitális jel egy tartományon belül is csak adott diszkrét értékeket vehet fel. Két diszkrét érték vagy állapot közötti érték nem megengedett. A digitális áramkörökben leggyakrabban két diszkrét állapotot használunk. Ennek az elsődleges oka, hogy a legegyszerűbben és legbiztonságosabban két állapotot tudunk egymástól megkülönböztetni. Több állapot megkülönböztetése bonyolultabb és megbízhatatlanabb lenne.

Az előbbieket magyarázatára nézzünk néhány példát, a mai „digitális kor” előtti időkből, ahol a digitális jelek használata már mindennapnak számított, csak legfeljebb nem nevezték nevén a folyamatot.

- A jelzőtűz vagy a füstjelzés talán egyike a legősibb, - nem elektronikus - jelzésformáknak. A füst vagy a tűz megléte vagy hiánya valamilyen jelenséghez vagy eseményhez kapcsolva nagy távolságú kétállapotú jelátvitelt tett lehetővé. A füst jelzését például egy pokróccal, mint modulátorral ki- bekapcsolgatva bonyolultabb jelzésformákat is megvalósíthattak.
- A gyertya az ablakban az elmúlt korok egyik jellegzetes jelzésformája a „tiszt a levegő” átvitt értelmű jelzés tudtára adásának.
- Nem emberi találmány az élő szervezetek tulajdonságait kódoló DNS kód. Az „A”, „C”, „T”, és „G” –vel jelölt különböző kombinációk úgy is felfoghatók, mint egy négyjegyű számkód, mely az információt hordozza, illetve átviszi két egymást követő generáció között. A teljes igazsághoz hozzátartozik, hogy a DNS-ben a négyféle *nukleotid* egymáshoz képesti térbeli elrendeződései is hordoznak információt, valamint a közöttük kialakuló másodlagos kötések is.
- Sokak számára ismert a Morze kód, mint információ átviteli kódrendszer. Ez ötféle jel variációiból áll. Ezek a pont, vonás, vagyis rövid és hosszú jelzések, rövid szünet (a betűk elválasztására), közepes szünet (a szavak elválasztására) és a hosszú szünet (a mondatok elválasztására). A morzekód rendszerének köszönhetően az így kódolt üzenetek többféle módon is eljuthatnak a címzetthez, azaz többféle közvetítő közeg is használható. Ilyenek az elektromosság (elektromos távíró), vagy a fény (villanófény).
- A Braille rendszer vakok számára készült, a vakok „olvasását” lehetővé tevő 6 bites bináris formátumú kódrendszer, karakterek kódolására. A karakterek kódjait egy pontokból álló mintázat alkotja.
- A szemafor jelzéseknél rudakat, vagy zászlókat tartanak meghatározott helyzetben. A jelző eszközök egymáshoz képesti elhelyezkedése kódolja az üzenetet, amit a megfigyelő adott távolsáig képes észlelni. Vasúti jelzők vagy a reptéri irányítók jelzései működnek ezen a módon.

- A tengeri jelzőzászlók különböző jelzései az ABC különböző betűit jelképezik, ennek segítségével tudnak a hajók egymásnak látótávolságon belül üzenetet küldeni.
- Végezetül egy megtörtént eset a régmúltból, amely jól rámutat elődeink leleményességére és arra, hogy „digitális módon”, hogyan lehet megoldhatatlannak tűnő problémákat is megoldani.

Bar Kohba (családnevén Simon Ben Kosziba) az 1.- 2. században élt zsidó szabadságharcos a zsidó történelem legendás alakja, az utolsó ókori függetlenségi háború (i.sz. 133-135) hadvezére volt. Róla nevezték el később a *barkóba* játékot. A legenda szerint Simon Bar Kohba elé vittek egy kirabolt és megcsonkított embert. A rablók a szerencsétlenek a nyelvét is kivágták, hogy ne vallhasson ellenük, hiszen abban az időben az írástudás még nem volt elterjedt. Bar Kohba azonban olyan kérdéseket tett fel az áldozatnak, amelyekre az egyszerű biccentéssel vagy fejrázással, tehát igen, nem válaszokkal felelhetett. Így kiderült, hogy a tettesek az áldozat bátyjai voltak, akiket aztán el is fogtak, és elnyerték megérdemelt büntetésüket.

Talán nem teljes a lista, de jól érzékelteti, hogy információhordozó közeg jellegére való tekintet nélkül nagyon sok minden alkalmas a digitális információ reprezentálására és átvitelére.

Napjainkban a digitális rendszerek előretörését és fejlődését misen érzékelteti jobban, hogy nagyon sok olyan rendszer, amely eredetileg csak analóg módon volt elképzelhető és realizálható, mára szinte kizárólagosan digitális megvalósításban létezik, vagy rohamléptekben alakul át és az eredeti analóg rendszer lassan feledésbe merül.

A hangrögzítés első megvalósítása egy puha viaszhengerbe karcolt, a hang rezgésének megfelelő barázda volt. (Edison fonográfja) Majd következett a hasonló elven működő bakelit lemez, ahol a barázdák rezgéseit piezoelektromos, majd mágneses elven működő „hangszedők” alakították analóg elektromos jellé. Az elektronika fejlődésével megjelentek a mágneses hangrögzítők, a magnetofonok, ahol először acélhuzalra majd mágnesezhető anyaggal bevont műanyag fóliára rögzítették a hangjeleket, természetesen ekkor még analóg módon. 1983-ban jelent meg a CD lemez (Compact Disc), az első digitális hanghordozó, amely a köznapi felhasználásban is elterjedt, és már digitális formában tárolta a hang információt. Ne feledkezzünk meg a DAT magnetofonokról sem, ahol a hangot digitalizálva rögzítették az analóg magnetofonok hordozóihoz hasonló mágnesszalagra. Mára szinte kizárólag a digitális rögzítés és visszajátszás az elterjedt, nem csak a CD lemezek formájában, hanem a számítógépeken, mobiltelefonokon, hanglejátszókon elterjedt különböző formátumok esetében (wav, mp3, flac, stb.) is.

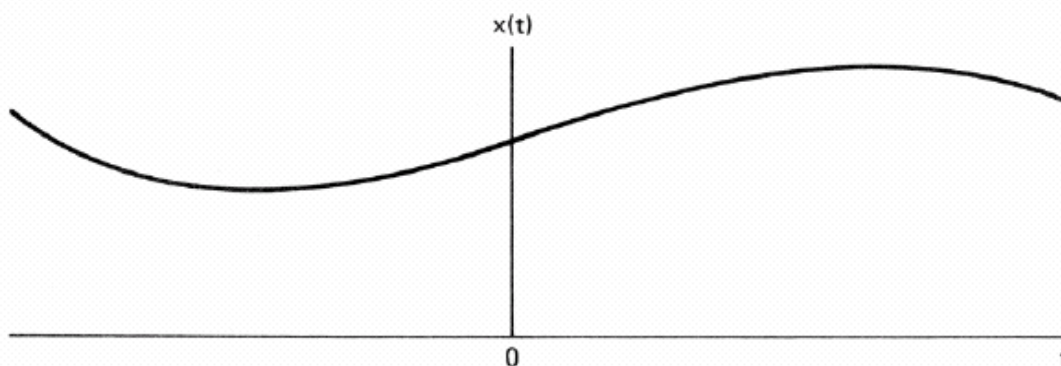
A videójelek (videómagnó, televízió, stb.) néhány éve még szintén analóg módon kerültek rögzítésre és sugárzásra. Mára a digitális műsorrögzítés szinte teljesen kiszorította a szalagos videó magnetofonokat a piacról. A DVD és a Blu-Ray nemcsak kisebb térfogaton képes ugyanannyi információ tárolására, de minőségében is túlszárnyalja az analóg elődöket. A műholdas és a földi műsorszórás is lassan, egyeduralkodó módon digitálissá válik.

A telefonon történő hangátvitel is néhány éve csak analóg módon zajlott. Ma a beszélgetések nagytöbbsége digitális vonalakon folyik, ami egyrészt csökkenti a nagytávolságú átvitelek esetén adódó hangminőség romlást, valamint lehetővé teszi egy csatornán egyidejűleg több beszélgetés lefolytatását (idő-multiplexelt átvitel). Az internet elterjedésével a szolgáltatók is egyre jobban kihasználják ezt a világot behálózó rendszert, mivel nem kell új adatátviteli vonalakat létesíteni. Az Internet ma már országokat, városokat, sőt az egyes felhasználókat is közvetlenül összeköti egymással. Sokszor nem is gondolunk bele, hogy otthoni analóg, vonalas telefonkészülékekkel kezdeményezett hívásunk milyen digitális csatornákon jut el a tárcsázott állomás irányába. A mobil kommunikáció esetében is hasonló a helyzet, a régi analóg rendszerek helyett ma kizárólagosan digitálisakat használnak.

Analóg és digitális jel

Az analóg jel

Analóg jel-egy megengedett tartományon belül bármilyen tetszőleges értéket felvehet. Például mutatóanalóg műszerek, mutatómérleg nyelve. Olyan jelek, amelyeknek minden független változója folytonos. A folytonos idejű jelek két időpont között *végtelen* sok értékre vannak definiálva, ezért *bármely* időpontra felvesznek értéket.



1.1 ábra: Folytonos jel

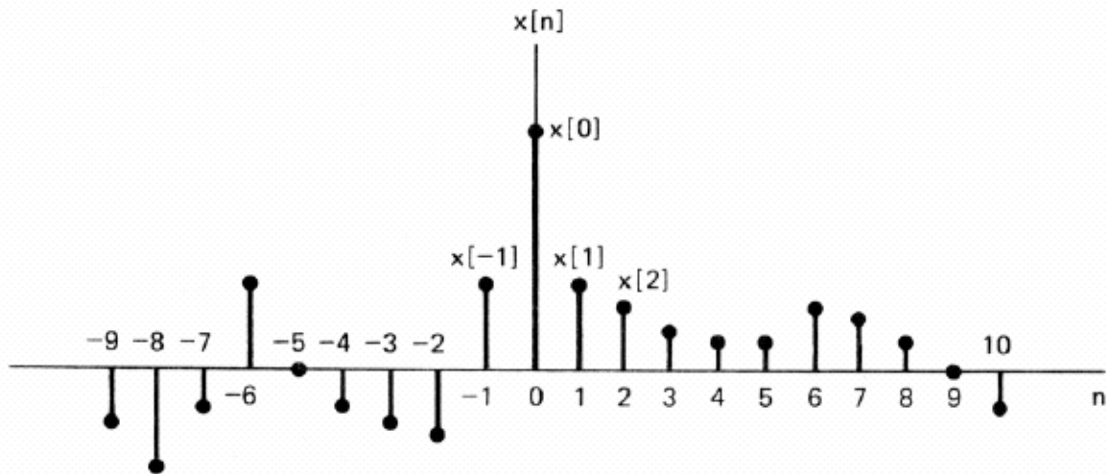
Az analóg jel egy folyamatosan változó jel idő és amplitúdó szerint egyaránt. Az analóg kifejezést többnyire elektronikus értelemben használják, bár mechanikai, pneumatikus, hidraulikus és más rendszerek is használhatnak analóg jeleket. Az analóg jel a közvetítő eszköz valamilyen tulajdonságát használja ki a jel információtartalmának továbbításához. Például a barométer mutatójának forgása révén vagyunk képesek a megfelelő jelnyomásra vonatkozó információtartalmát megjeleníteni. Elektronikus értelemben a leginkább használt tulajdonság az, hogy a feszültségváltozást szorosan követi a frekvencia, az áramerősség és a töltés megváltozása. Továbbá a beszéd átvitelének egyszerű módja. Az első mobiltelefonos hálózatok is analógok voltak.

Példák az analóg jelekre a mindennapi életünkben:

- Rajz, festmény
- Hagyományos eljárással készült fényképek
- Bakelitlemez

A digitális jel

Olyan jel, amelynek minden független változója diszkrét értékű. A diszkrét idejű jelek csak diszkrét időpontokra vannak definiálva, ezért mindig csak meghatározott időközökben vesznek fel értékeket, és az időközök között nem definiáltak.



1.2 ábra: Digitális jel

A digitális jel valamely változó jelenségnek, vagy fizikai mennyiségnek diszkrét (nem folytonos), megszámlálhatóan felaprózott, s így számokkal felírt értékein alapul. A digitális rendszerek sokkal inkább számokat (leginkább bináris számokat) használnak bevitelhez, feldolgozáshoz, átvitelhez, tároláshoz, vagy megjelenítéshez, mint az értékek folytonos spektrumát (ez utóbbit ugyanis az analóg rendszerek használják), vagy a nem-numerikus szimbólumokat, mint a betűk, vagy ikonok. A digitális szót leggyakrabban a számítástechnika és az elektronika területén használják, különösen azokon a területeken, ahol a való világ információit konvertálják át bináris számokká. Ilyenek például a digitális hang és a digitális fényképezés. A digitális adat-átvivő jelek az elektronikus, vagy optikai impulzus két lehetséges értéke közül az egyiket vehetik fel. A logikai 1 (van impulzus) vagy 0 (nincs impulzus) értékeket. Az ilyen rendszerű eszközöknél gyakran egy "e-" előtag utal a digitális mivolta, mint az e-mailnél, vagy az e-könyvnél, bár nem minden elektronikus rendszer digitális.

Példák a digitális jelekre a mindennapi életünkben:

- ClipArt galéria digitális képei
- Digitális fényképezőgép adatcsomagjai
- CD, DVD, HD-DVD, Blu-Ray

Analóg és digitális rendszerek

Egy szám valamely berendezésben megvalósított megjelenési formáját a *szám ábrázolásának* nevezzük. Ez valamilyen fizikai mennyiséggel való megfeleltetéssel történik, általában azzal arányos fizikai mennyiség. A szám nagyságát a legtöbb elektronikus berendezésben a villamos feszültség nagyságával ábrázoljuk.

Ha az így megjelenő mennyiség a megengedett legkisebb és legnagyobb érték között bármekkora lehet, akkor analóg számbábrázolásról beszélünk. Ha csak kitüntetett, diszkrét értékeket vehet fel, úgy digitális az ábrázolás.

Nézzünk egy hétköznapi példát a kettő közti különbség megértésére! A kereskedelemben többféle személymérleget vásárolhatunk. Léteznek olyanok, amelyeknél egy számlap forog, és egy mutató segítségével olvashatjuk le a testtömeget. Vannak olyanok is, amelyek kijelzőjén számok jelennek meg, és így tudhatjuk meg, hány kg a testtömegünk. Ebben az esetben a mérleg fél kg-onként mér, pontosabban jelez ki.

Mit mutat a mérleg akkor, ha valakinek mondjuk pontosan 76 kg 22 dkg a testtömege? Az első analóg kijelzésű mérleg esetében 76 kg 22 dkg-ot mutat a mérleg, a másik esetben 76 kg-ot. Mi a helyzet

akkor, ha a mérlegen álló ember kezébe adunk egy 54 dkg tömegű tárgyat? A „mutató” analóg mérlegnél 76 kg 76 dkg-ot, a „számos” mérleg esetében 77 kg-ot olvashatunk le.

Mi történik, ha csak egy 10 dkg-os tárgyat adunk oda a mérlegen álló ember kezébe? A „mutató” analóg mérlegnél látszik a változás, 76 kg 32 dkg-ot fog mérni, a „számos”, digitális kijelzésű továbbra is 76 kg-ot.

Láthatjuk, hogy a „mutató” mérleg tetszőleges kis változást megmutat (amennyiben eléggé részletes és pontos a skálabeosztása), a „számos”, digitális mérlegnél csak fél kg-os pontossággal tudjuk leolvasni a mért adatot, két megengedett érték között nem. Az első mérleg analóg rendszerű, hiszen a megengedett értékhatárok között tetszőleges értékeket vehet fel, a második digitális, mert csak előre meghatározott értékeket mér.

Miért van mégis az, hogy a műszaki világ az analóg technikáról egyre több téren igyekszik áttérni a digitális rendszerekre, ha az analóg technikával – úgy tűnik – nagyobb mérési pontosság érhető el?

A magyarázat ideális és a valós viszonyok különbségében rejlik. A leolvasási nehézségen túl mindig jelentkezik két tényező, ami miatt lassan minden téren áttérnek a digitális berendezésekre. Az egyik tényező a zaj és a külső zavaró jelek hatása, ami minden fizikai folyamatot terhel. A szám ábrázolási pontosságát a valóságban a zaj mindig behatárolja. Ez különböző módszerekkel csökkenthető, de nem szüntethető meg, ráadásul a zajvédelem költségei és járulékos „mellékhatásai” is problémát okozhatnak.

A másik tényező a feldolgozó berendezések torzítása. Míg a zaj véletlenszerű, addig a torzítás elméletileg prognosztizálható és korrigálható. Azonban ez a korrekció függ a hőmérséklettől, a tápfeszültség változásaitól és még sok minden mástól, ezáltal olyan bonyolulttá válik, hogy ismét a költség és a súly korlátjába ütközünk.

Hogyan oldja meg ezt a problémát a digitalizálás? A megengedett legkisebb és legnagyobb feszültség szintet egész számú egyforma tartományra osztjuk, és csak a tartományok közepének megfelelő feszültség szintek reprezentációját engedjük meg. A zaj is és a torzítás is megváltoztatják a jelet, de ha a zavaró hatás miatt még nem csúszik át a feszültség egy másik tartományba, úgy még a helyes értéként „azonosítható” és helyreállítható a szintje. (A mérleg esetében, amikor 10 dkg-ot adunk a mérlegen álló kezébe, az analóg mérleg megérezte a zavaró körülmény okozta változást, a digitális nem.) Az információ-feldolgozó egységek szerepe a feldolgozáson túl a jelszintek helyreállítása, vagyis kondicionálása is.

A digitális tervezés során biztosítani kell, hogy a zaj ne érje el a tartományszélesség, más néven felbontás értékének a felét, vagyis a kritikus hibát.

Nyilvánvaló, hogy a berendezés zavarvédeltsége a „biztonsági tartományok” szélességével arányos, vagyis minél kevesebb részre osztjuk a rendelkezésre álló feszültségtartományt, tehát csökkentjük a felbontást, zavarvédelmi szempontból annál megbízhatóbb a rendszer. Természetesen a felbontás csökkentésével az információ tartalom és pontosság is csökken (például csak 5 kg pontossággal tudom megmérni a súlyomat a mérlegen). A zavarvédeltség és pontosság tekintetében tehát kompromisszumot kell kötnünk, azaz minden esetben az elvárásoknak megfelelő, de azon túl nem mutató rendszereket célszerű tervezni.

Ha ilyen módon kezeljük az információt, akkor azt megfelelő műszaki feltételek mellett tetszőlegesen sokszor másolhatjuk és nagy távolságra is átvihetjük, a jel sérülése, torzításának megnövekedése nélkül. Ez viszont az analóg rendszerekhez képest hatalmas előnyt jelent.

2. Logikai függvények megadása

A digitális technika nyelvén megfogalmazható feladatok nagy többsége felírható logikai döntések sorozataként. A logikai döntések állításokból épülnek fel, a logikai állítások végeredményét pedig a bennük foglalt logikai kapcsolatok és a logikai változók értékei határozzák meg. A logikai döntések végeredménye a függő változó, mint kimeneti érték. A független és a függő változók közötti kapcsolatot a logikai függvények írják le.

A logikai függvényeket a következő módokon lehet megadni:

- Szöveges formában,
- Időfüggvény alakjában,
- Táblázatos módon,
- Grafikus módon,
- Algebrai alakban.

A fenti megadási módok bármelyikéből bármelyikbe átírható a függvény, a megadási módok kölcsönösen egyértelműek.

Logikai feladat megadása szöveges formában

Ez tulajdonképpen a feladat leírása elbeszélés alapján. Nem tömör, nem gyakorlatias. Bár a legtöbb információt hordozhatja, mégis ritkán alkalmazzák. Inkább csak kiegészítésként, vagy a feladat bizonyos részeinek pontosításához adják meg. További hátránya, hogy nem alkalmas műveletvégzésre és az ebből az alakból történő átírás algoritmizálható a legkevésbé.

Logikai feladat megadása időfüggvény alakjában

Az időfüggvénnyel történő felírás legnagyobb előnye, hogy a bemeneti és kimeneti változók pontos időzítése – egymáshoz való időbeni elhelyezkedésük – pontosan látható. Nagyon szemléletes megadási mód. Bár a megadás egyértelmű, de a konkrét feladat kizárólag az időfüggvényből nem mindig olvasható ki részletesen.

Logikai feladat megadása táblázatos módon

Mivel az irányítási feladatokban előforduló bemeneti és kimeneti változók száma és a megfogalmazott állítások száma is véges, a feladatot, a változókat tartalmazó táblázat formában is megadhatjuk.

A logikai kapcsolatok táblázatos megadásának módja az igazságtáblázat. A táblázat oszlopai a logikai változókat tartalmazzák (független változók + függő változók), a táblázat sorainak száma pedig megegyezik a független változók kombinációinak a számával.

Néhány példa egyszerű logikai függvények táblázatos megadására:

$$Q = AB$$

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

$$Q = A + B$$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

$$Q = \bar{A}$$

A	Q
0	1
1	0

Összetettebb függvények táblázatos megadásának módja a következő:

- A független és a függő változók számának megfelelően meghatározzuk a táblázat oszlopainak számát.
- A független változók oszlopaiba beírjuk a független változók összes lehetséges kombinációjának megfelelő bináris értékeket.
- A független változók kombinációihoz megadjuk a függő változók megfelelő értékeit.

Logikai feladat megadása grafikus módon

A logikai függvények grafikus úton történő megadása a Veitch-Karnaugh táblák segítségével lehetséges. Ezen felírasmóddal és a segítségével történő függvényegyszerűsítés lehetőségével a későbbiekben részletesen foglalkozunk.

Logikai feladat megadása algebrai alakban

A logikai függvények algebrai alakjában a logikai változók és a Boole-algebrának megfelelő logikai műveletek szerepelnek. Az így felírt logikai függvény rendezett **ÉS-VAGY** alakú. A logikai műveleti jelekkel összekapcsolt egységekben az összes független logikai változó szerepel ponált vagy negált alakban.

Példa az algebrai alakra:

$$Q = A\bar{B}C + AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C$$

Logikai függvények kanonikus alakjai

Ha a logikai függvények algebrai úton történő megadása során definiálunk egy olyan alakot, amelynek tulajdonságai kizárják, hogy egy adott logikai függvényhez több, ilyen tulajdonságú algebrai alak tartszon, akkor ezt az alakot kanonikus vagy normál alaknak nevezzük. A kanonikus alaknak kétféle típusa létezik, a diszjunktív illetve a konjunktív kanonikus normálalak.

Nézzük először a diszjunktív normál alakot. Egy háromváltozós logikai függvény felírható például a következő alakban:

$$F(A, B, C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + ABC$$

Az így felírt függvényalaknak a következő tulajdonságai vannak:

- Mindegyik szorzat alak egy olyan függetlenváltozó-kombinációt képvisel, amelyhez tartozó függő változó, - teát kimeneti változó érték- logikai 1 értékű.
- Mindegyik szorzatban az összes független változó szerepel ponált vagy negált értékével.

A fenti tulajdonságokból következik, hogy minden teljesen határozott logikai függvénynek csak egy ilyen alakja létezik. A logikai szorzatok egy-egy olyan logikai függvénynek tekinthetők, melynek függvényértéke csak egy független változó kombináció esetén 1. Ezeket a speciális elemi logikai függvényeket *mintermeknek* nevezzük és

$$m_i^n$$

módon jelölik, ahol n a független változók számát, i a mintermnek megfelelő független változó kombinációt jelölő bináris érték decimális megfelelője.

Ezek alapján a fenti függvény a következő alakban is felírható:

$$F(A, B, C) = m_2^3 + m_3^3 + m_4^3 + m_6^3$$

A teljesen határozott logikai függvény egyértelműen megadható azoknak a szorzatoknak a VAGY kapcsolatával is, ahol a megfelelő függetlenváltozó-kombináció esetén a függvényérték 0.

Mivel $F = 0$ esetén $\bar{F} = 1$, és az $F = 0$ -hoz tartozó kombinációk logikai összegét képezzük, akkor az F függvény negáltjának a diszjunktív kanonikus alakját kapjuk. Az előbbi függvény esetén, az

$$\overline{F(A, B, C)} = \overline{A\bar{B}\bar{C}} + \overline{\bar{A}BC} + \overline{A\bar{B}C} + \overline{ABC}$$

alakhoz jutunk, amit a függvény negáltjának diszjunktív kanonikus alakjának nevezünk. A mintermekre vonatkozó előbbi jelölést használva:

$$\overline{F(A, B, C)} = m_0^3 + m_1^3 + m_5^3 + m_7^3$$

Teljesen határozott logikai függvények esetén tehát, a függvény negáltja azokat a mintermeket tartalmazza, amelyeket az eredeti függvény nem tartalmazott.

Ha képezzük a függvény negáltjának a negáltját, akkor a már megismert összefüggések alapján természetesen az eredeti függvényt kapjuk vissza. Ha ez mellett még alkalmazzuk a De Morgan azonosságot, egy másik kanonikus alakot kapunk.

Az előbbi függvénnel elvégezve a műveletet, a következő alakot kapjuk:

$$\begin{aligned} \overline{\overline{F(A, B, C)}} &= F(A, B, C) = \overline{\overline{A\bar{B}\bar{C}} + \overline{\bar{A}BC} + \overline{A\bar{B}C} + \overline{ABC}} = \\ &= (A + B + C) \cdot (A + B + \bar{C}) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C}) \end{aligned}$$

Az így kapott függvényalaknak a következő tulajdonságai vannak:

- Logikai összegek szorzatai szerepelnek benne. Mindegyik összeg alak egy olyan függetlenváltozó-kombinációt képvisel, amelyhez tartozó függő változó, - teát kimeneti változó érték- logikai 0 értékű.
- Mindegyik összegben az összes független változó szerepel ponált vagy negált értékével. Az összegekben a 0 értékű logikai változók ponált, az 1 értékű változók negált alakban szerepelnek.

A fenti két tulajdonságokból következik, hogy minden teljesen határozott logikai függvénynek csak egy ilyen alakja létezik. Az így kapott kanonikus alakot konjunktív kanonikus alaknak nevezzük. A logikai összegek egy-egy olyan logikai függvénynek tekinthetők, melynek függvényértéke csak egy független változó kombináció esetén 0. Ezeket a speciális elemi logikai függvényeket *maxterm*eknek nevezzük és

$$M_i^n$$

módon jelölik, ahol n a független változók számát, i a maxtermnek megfelelő független változó kombinációt jelölő bináris érték decimális megfelelője. Az i index képzésénél is a ponált változóknak 1, a negált változóknak 0 értéket tulajdonítunk. Ez megállapodáson alapul.

Ezek alapján a fenti függvény konjunktív kanonikus alakja következő módon is felírható:

$$F(A, B, C) = M_7^3 + M_6^3 + M_2^3 + M_0^3$$

A függvények megadási módját tovább is egyszerűsíthetjük oly módon, hogy megadjuk a függvény alakot (minterm, vagy maxterm), a függvényben szereplő változók számát, és a függvényben szereplő terem sorszámait.

Diszjunktív alak esetében a logikai összegzést Σ -val, konjunktív alak esetében a logikai szorzást Π -vel jelöljük. A jelölések fölé írjuk a változók számát.

$$F = \overset{n}{\Sigma}(\dots\dots)$$

$$F = \overset{n}{\Pi}(\dots\dots)$$

A függvényekben szereplő terem sorszámát a szimbólumot követő zárójelek között soroljuk fel vesszővel elválasztva.

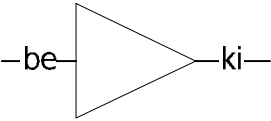
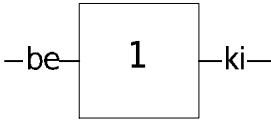
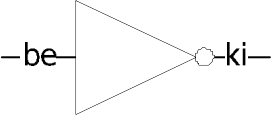
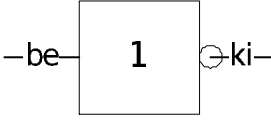
Az elvi logikai rajz vagy logikai vázlat

A logikai funkciókat az áramkörök megvalósítási módjához közel álló szimbólumokkal is szokták jelölni. Ez a megoldás nagyban segíti az áramkörök tervezését és megvalósítását. Többféle

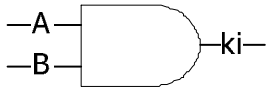
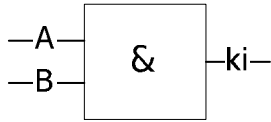
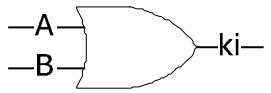
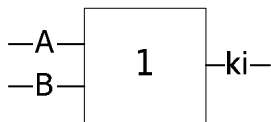
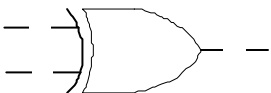
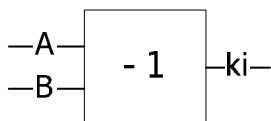

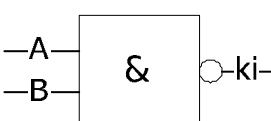
szabványos rajzjelölés is elterjedt az utóbbi időben. Mi a hazánkban és Európában érvényes jelölésmódot használjuk, melyet néhány egyszerű szempont szerint az alábbiakkal jellemezhetünk:

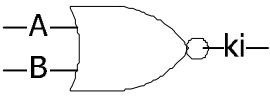
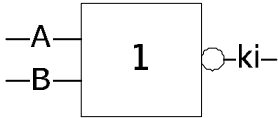
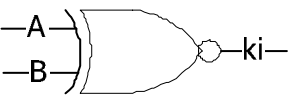
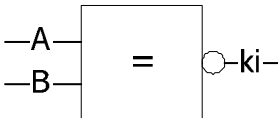
- A logikai szimbólumok jelölésére négyszöget használnak.
- A négyszög belsejébe írják a logikai funkcióra utaló jelölést.
- A független változókat (bemenetek) jelölő csatlakozások a keret bal oldalához illeszkednek.
- A függő változókat (kimenetek) jelölő csatlakozások a keret jobb oldalához illeszkednek.

Egyváltozós logikai függvények

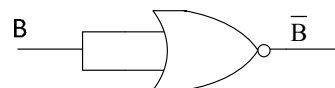
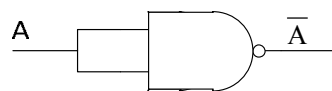
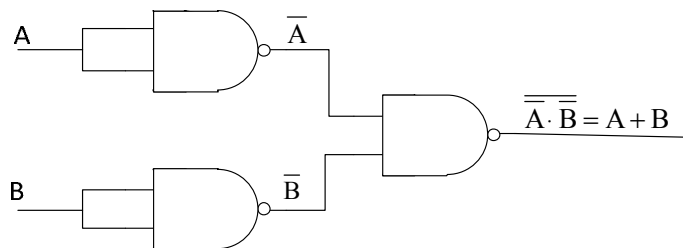
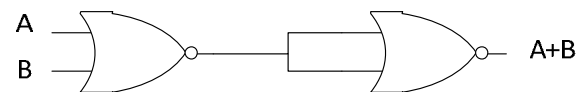
	Igazságtáblázat	Nemzetközi szabványjelölés	Magyar szabványjelölés						
Jelmásoló, buffer	<table border="1"> <thead> <tr> <th>be</th> <th>ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	be	ki	0	0	1	1		
be	ki								
0	0								
1	1								
Negálás, inverter	<table border="1"> <thead> <tr> <th>be</th> <th>ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	be	ki	0	1	1	0		
be	ki								
0	1								
1	0								

Többváltozós logikai függvények

	Igazságtáblázat	Nemzetközi szabványjelölés	Magyar szabványjelölés															
ÉS kapcsolat (AND)	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	ki	0	0	0	0	1	0	1	0	0	1	1	1		
A	B	ki																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
VAGY kapcsolat (OR)	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	ki	0	0	0	0	1	1	1	0	1	1	1	1		
A	B	ki																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Antivalencia kapcsolat (XOR)	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Ki	0	0	0	0	1	1	1	0	1	1	1	0		
A	B	Ki																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
NEM-ÉS kapcsolat (NAND)	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Ki	0	0	1	0	1	1	1	0	1	1	1	0		
A	B	Ki																
0	0	1																
0	1	1																
1	0	1																
1	1	0																

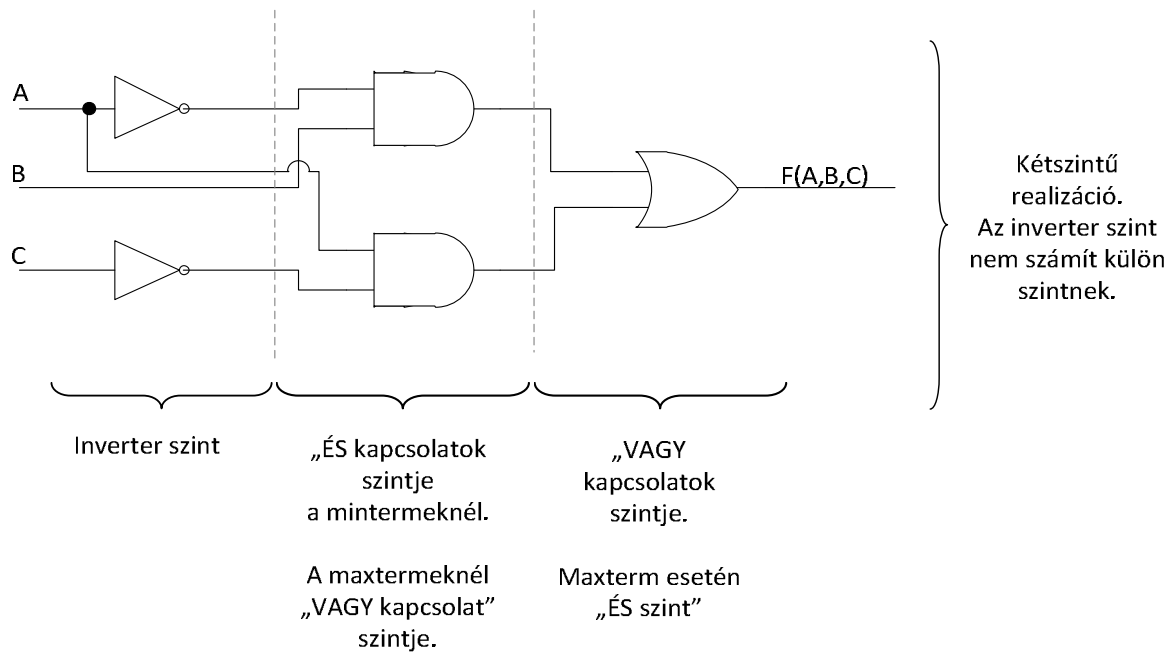
	Igazságtáblázat	Nemzetközi szabványjelölés	Magyar szabványjelölés															
NEM-VAGY kapcsolat (NOR)	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Ki	0	0	1	0	1	0	1	0	0	1	1	0		
A	B	Ki																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Ekvivalencia kapcsolat (NXOR)	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Ki</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Ki	0	0	1	0	1	0	1	0	0	1	1	1		
A	B	Ki																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Szemléltető példák: (függvények realizálása):

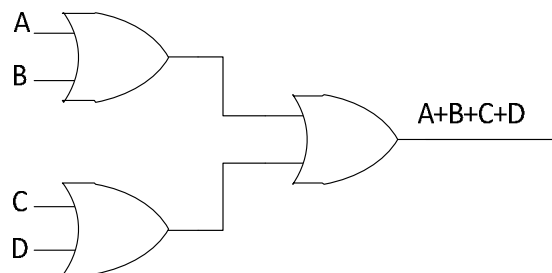


Szemléltető példa: (függvény kétszintű realizációja):

$$\begin{aligned}
 F(A, B, C) &= \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + AB\overline{C} = \\
 &= \overline{A}B(\underbrace{C + \overline{C}}_1) + A\overline{C}(\underbrace{B + \overline{B}}_1) = \overline{A}B + A\overline{C}
 \end{aligned}$$



Szemléltető példa (2-bemenetű kapukból 4-bemenetű készítése):



Logikai függvények grafikus ábrázolása, Veitch-Karnaugh diagram

A grafikus ábrázoláshoz használt Karnaugh-Veitch diagram (KV tábla) tulajdonképpen egy speciális esete az igazságtáblának. A diagram lehet síkbeli, vagy több változó esetén térbeli geometriai alakzat. A táblázat négyzet alakú elemi cellákat tartalmaz.

A táblázatnak annyi cellája van ahány független változó kombináció lehetséges adott számú független változó esetén. Tehát a cellák száma = 2^n , ahol n a független változók száma. A táblázat tehát 2,4,8,16,32 vagy 64 elemi cellát tartalmaz. 2 illetve 4 cella esetén ritkán alkalmazzuk, a kevés független változó miatt nem indokolt az ilyen módon végzett egyszerűsítés, bár szemléltetésre ekkor is alkalmas. 6 változó felett viszont már a térbeli táblázat sem használható.

Az alábbi ábra egy háromváltozós Karnaugh táblázatot mutat a szokásos jelölésekkel.

		BC		C		B	
		00	01	11	10		
A	0	Y_0	Y_1	Y_3	Y_2		
	1	Y_4	Y_5	Y_7	Y_6		

A táblázatnak 8 cellája van, ezek két sorban és négy oszlopban helyezkednek el, de a sorok és oszlopok elhelyezése meg is cserélődhet. A táblázat celláinak jobb alsó sarkában kis indexek találhatóak. Ezek az elemi cella által „lefedett” minterm vagy maxterm bináris indexének decimális megfelelői. A táblázat szélein az un. peremezés található. Ez segít beazonosítani a cellák által lefedett logikai változót. Látható, hogy a peremre írt logikai változó jele azokat a cellákat „takarja”, amelyek bináris indexében az adott változó ponált értékkel szerepel.

A fenti ábrán a háromváltozós tábla egy lehetséges kitöltése és a hozzá tartozó peremezés látható. (A a legnagyobb helyiértékű változó).

A fenti szabályok alapján lehetséges a táblázat más kitöltése is. A következő ábrán egy másik háromváltozós tábla elrendezése látható.

		AB		B		A	
		00	01	11	10		
C	0	Y_0	Y_2	Y_6	Y_4		
	1	Y_1	Y_3	Y_7	Y_5		

Négy változó esetén a táblázat 16 elemi cellát tartalmaz. Az előbbi megfontolások alapján a táblázat egy lehetséges kitöltési formája a következő:

		CD		C			
		00	01	11	10		
A	00	Y_0	Y_1	Y_3	Y_2		
	01	Y_4	Y_5	Y_7	Y_6		
	11	Y_{12}	Y_{13}	Y_{15}	Y_{14}		
	10	Y_8	Y_9	Y_{11}	Y_{10}		
		D				B	

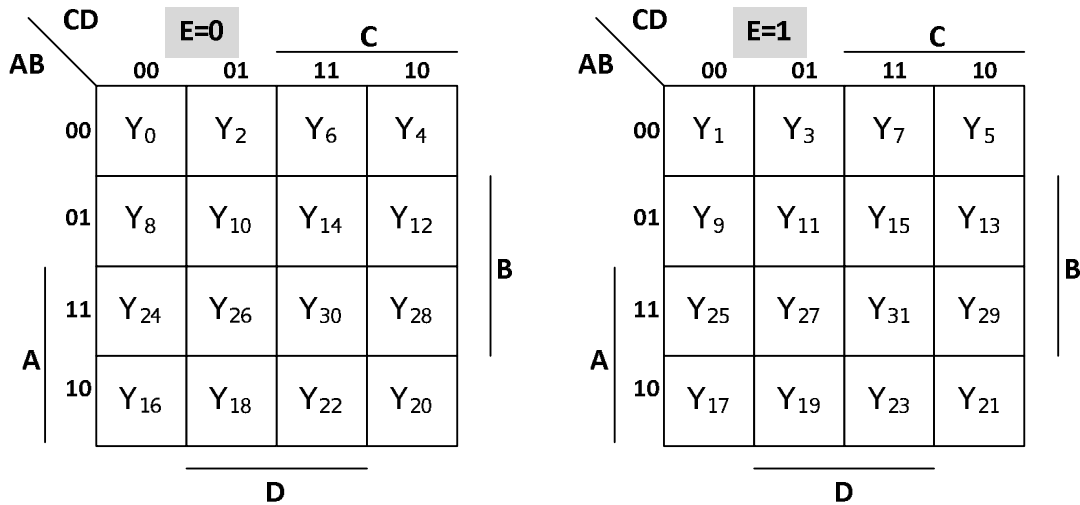
A példák alapján látható, hogy a táblázatok úgy kell elképzelni, mint a logikai változók halmazát ábrázoló területet, amit mindig két ténfélre osztunk. Az egyik félben az adott független változó ponált

vagy igaz értékei, a másikon a negált vagy hamis értékei találhatók. A táblázat megfelelő kitöltésével elérhető, hogy egyszerre több logikai változóra is teljesüljön ez a feltétel.

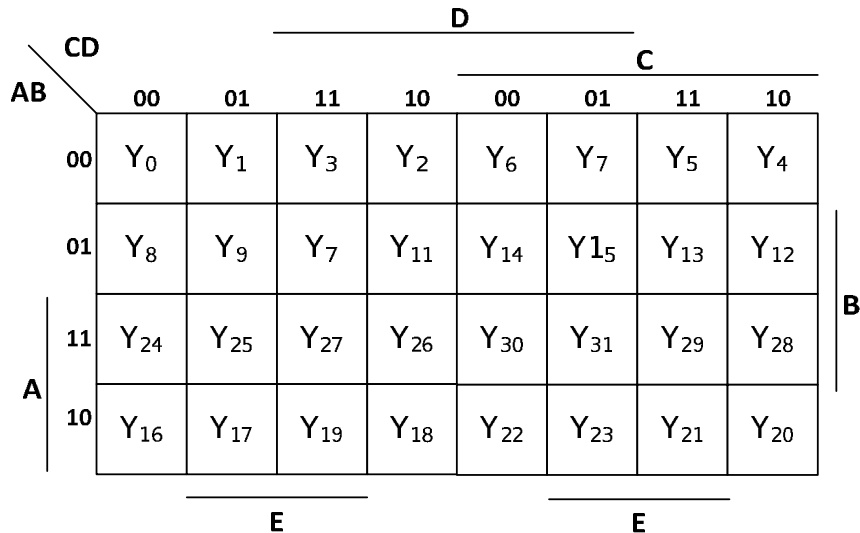
Öt logikai változó esetén már kissé bonyolultabb a helyzet. A logikai változókat tartalmazó táblázat már nem bővíthető az eddigi módon úgy, hogy ne veszítsük el a táblázatos ábrázolás legnagyobb előnyét a szemléletességet.

Ezért az ötödik változó ábrázolásához egy újabb táblázatot készítünk az alábbi módon. A két táblát tulajdonképpen egymás mögött kell elképzelnünk, hogy a későbbiekben ismertetendő szomszédossági feltételek egyértelműbben kivehetőek legyenek.

Példa: (5 változós függvény Veitch-Karnaugh táblája):

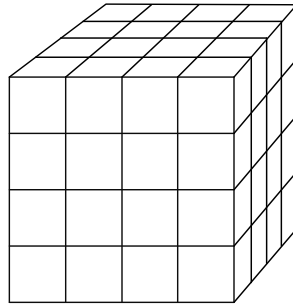


Ötváltozós Veitch-Karnaugh tábla síkjai egymás mellé helyezve, egy másféle változókiosztás szerint. Ebben az elrendezésben nehezebben ismerhetők fel a termek közötti kapcsolatok.



Hat változó esetén a táblázatunk egy térbeli kockává bővül. Ezen belül már a szemléletesség és a szomszédossági viszonyok tisztázása is nagy gyakorlatot igényel.

Példa: n=6 változó (A,B,C,D,E,F) változó esetén a Karnaugh tábla, egy kocka lenne. (Az alábbi ábra jelölések nélkül.)



Lehetséges a $n=6$ változó (A,B,C,D,E,F) esetén még a síkbeli ábrázolás is, de ekkor már az ábrázolási mód előnyei egyre nehezebben aknázhatók ki.

		CBA							
		FED							
		000	001	011	010	100	101	111	110
000									
001									
011									
010									
100									
101									
111									
110									

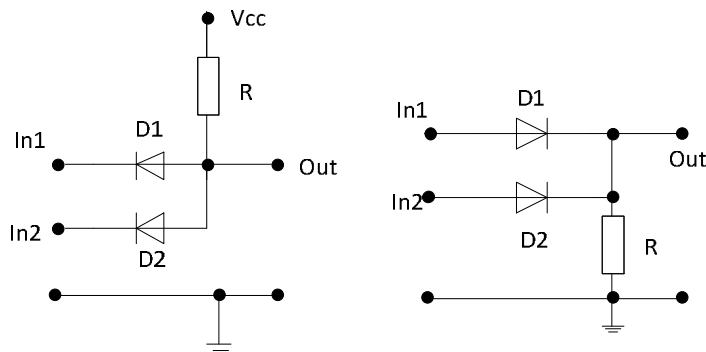
A példából látható, hogy a grafikus ábrázolást és a függvényegyszerűsítést maximum 6 változóig lehetséges elvégezni, de már hat változó esetén is komoly feladat lenne.

3. Logikai áramkörök TTL-CMOS rendszerek

Logikai áramkörrendszerek

DDL dióda-dióda logika

Logikai műveletek realizálására használt, diódákat és munkaellenállást tartalmazó kapuáramkör. A diódás kapcsolók ideálistól eltérő tulajdonsága miatt a kapuk kimenetén megjelenő logikai feszültségszintek a névlegestől eltérnek. A jelszint és jelalak torzulás miatt diódás logikából két fokozatnál többet nem kapcsolnak egymás után. A jelszint torzulások például inverterek, segítségével regenerálhatók. Főként ezen hátránya miatt ma már nem alkalmazzák integrált áramkörökben.

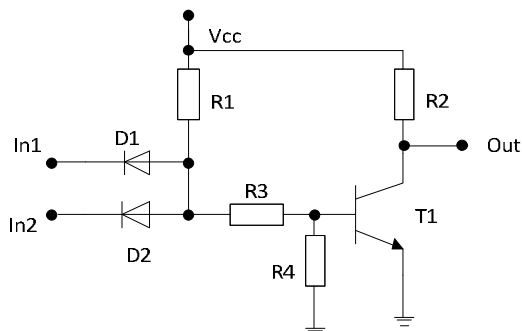


3.1 ábra: Diódás logikai ÉS, valamint diódás logikai VAGY kapu rajza

DTL dióda-tranzisztor logika

Az áramköri megoldás lényege, hogy a logikai művelet végzését továbbra is a diódás logika végzi, azonban a jelszint csökkenést egy tranzisztoros inverter segítségével helyreállítjuk. A beépített felhúzó- és lehúzóellenállások a munkapont beállítását segítik.

Integrált áramkör formájában ezt a megoldást sem használják már, azonban gyakran kell olyan logikai kapcsolatot létrehozni, ahol célszerű két diódát beépíteni egy újabb integrált áramkör beforrasztása helyett, ezért érdemes néhány szót ejteni róla.

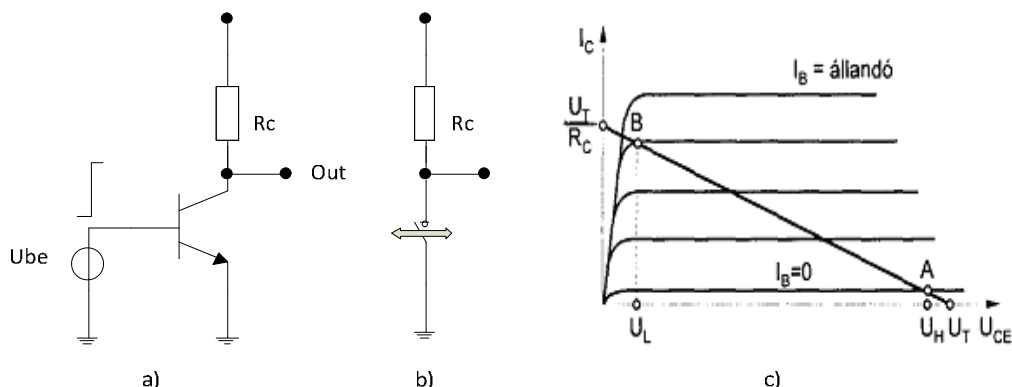


3.2 ábra: DTL dióda-tranzisztor logikai kapcsolás

TTL tranzisztor-tranzisztor logika

Az elektronikában a mai napig igen kedvelt és gyártott logikai áramkörtechnológia. Gyártástechnológiája egyszerű, olcsó. Az előző megoldásokhoz képest itt a logikai művelet végzését is tranzisztoros kapcsolással oldják meg. Az úgynevezett multiemitteres bemeneti fokozat (V1) a kapcsolástechnikájának köszönhetően előállítja a kimenetén a logikai értéket. Az ezt követő fokozatok csak jelszintillesztést és végerősítő funkciót látnak el. Hátránya, hogy a precíz működéshez szükséges munkapont csak egyetlen tápfeszültség érték esetén garantált, ezért a TTL áramkörök +5V $\pm 5\%$ tápfeszültségről működtethetők. A TTL áramkörök 74xxx vagy 54xxx típusjelöléssel kerültek

forgalomba. Ezen belül különböző altípusokat fejlesztettek ki az áramkör család egyes tulajdonságainak további javítása céljából.



3.3 ábra: Bipoláris tranzisztor kapcsolóüzeme
a) elvi rajz b) helyettesítő kapcsolás c) Bipoláris tranzisztor kimeneti jelleggörbéi

A bipoláris tranzisztor működéséhez negatív és pozitív töltéshordozókra egyaránt szükség van, NPN vagy PNP típusú lehet. A bipoláris tranzisztor a logikai áramkörökben földelt emitterű alapkapcsolásban, mint kétállapotú kapcsolóelem működik. A tranzisztor telített, és lezárt állapota megfelel egy bekapcsolt, illetve kinyitott kapcsolónak. Az R_c kollektor ellenállás munkaegyenese a tranzisztor kimeneti jelleggörbéiből kimetszi a telített, illetve a lezárt állapotnak megfelelő munkapontokat. A telített tranzisztor kollektorán egy kis maradék fesz lép fel. U_L sohasem lehet pontosan 0V, hanem annál egy kicsit pozitívabb. A nagyobb feszültség szint U_H sosem éri el U_T tápfeszültség értékét.

LS altípus

Közvetlenül az első TTL IC-k kifejlesztése után megjelent az LS széria. Például: 74LS47. Az LS a „low power shottky” kifejezés rövidítése, amely technológiailag annyit jelent, hogy egy shottky diódát építettek a bázis és a kollektor közé az ún. Miller kapacitás hatásának csökkentése érdekében. Ezáltal kisebb áramokkal hajtva a logikai kaput közel ugyanazt a sebességet érték el.

F altípus

Ennek a megoldásnak pontosan az ellentétével gyorsítottak az F sorozat működési sebességén, például 74F93, amelynél megnövelve az áramkör fogyasztását gyorsítottak a kapu működésén.

AC és ACT altípus

Advanced CMOS TTL input kompatibilis. Igen gyors logikai kapuk, de már nem a hagyományos TTL áramköri technológiával készülnek. Az alkalmazott kapcsolástechnika CMOS áramkörökre épül (lásd később), de TTL kompatibilis bemenő jelszinteket fogadnak.

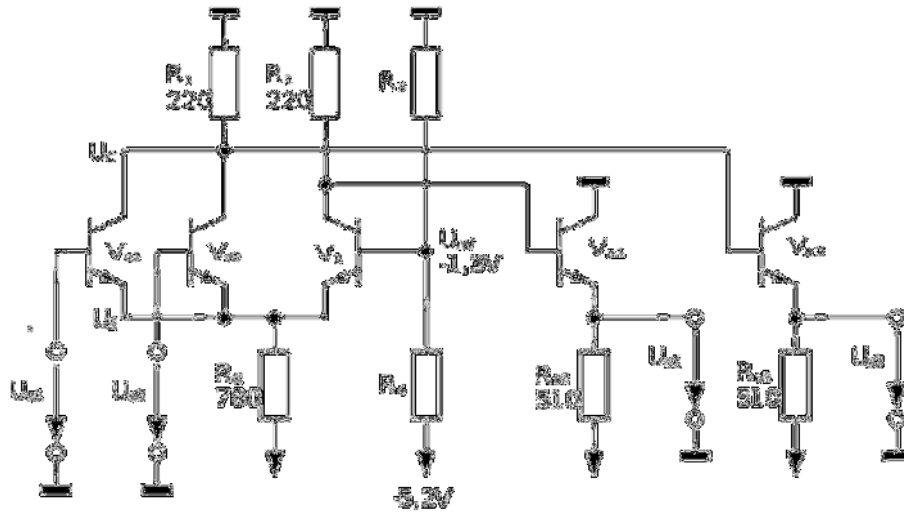
HC és HCT altípus

A legfrissebb tagja a 74xxx integrált áramköröknek, azonban ez a széria sem TTL, hanem CMOS technológiával készült. A HC tipikusan 2-6 (max 7) voltos tápfeszültséggel működik, a HCT-nél a T utal a „TTL kompatibilis” szóra, amely azt jelenti, hogy 4,75 - 5,25 V közötti logikára lett optimalizálva, itt gyorsabb működésű, mint a HC család.

ECL Emitter csatolású logika

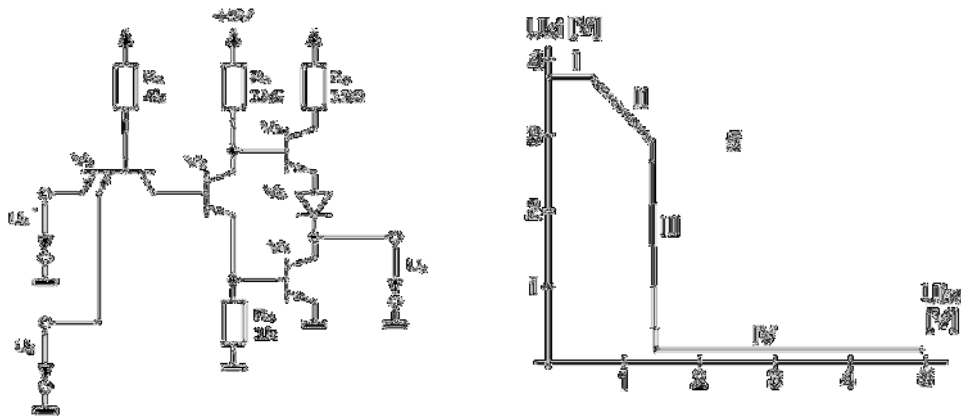
Egy logika áramkör terjedési-késleltetési ideje jelentősen lecsökken, ha a tranzisztorok telítéses üzemmódját kiküszöböljük. Az ECL áramkörös logikai hálózatokban nincs szükség inverterre. Ugyanis az ECL, kapuk két komplementes kimenettel rendelkeznek. Valamennyi bipoláris áramkör család közül

az ECL kapcsolások kapukésleltetési ideje a legkisebb. Működési sebességüket tekintve még a Schottky-TTL kapcsolásoknál is gyorsabbak.



3.4 ábra: ECL logikai VAGY kapu felépítése

TTL kapuáramkörök karakterisztikája



3.5 ábra: Egy hagyományos TTL ÉS alap kapuáramkör és transzfer karakterisztikája.

Az I. szakaszban, amikor a bemeneti feszültség alacsony, a kimenet feszültsége kb. 3,6 V, ez a TTL magas logikai szintnek felel meg, ami úgy adódik ki, hogy az 5V-os tápfeszültségből levonódik a végfokozat tranzistorjának bázis-emitter feszültsége és egy diódának a nyitófeszültsége (mindkettő kb. 0,7 V).

A II. szakaszban a V2 tranzisztor már vezet, de V4 még lezárt állapotban van. A kimenő feszültség fokozatosan csökken.

A III. szakaszban V4 is vezet. Az eszköz ilyenkor úgy működik, mint egy ellenütemű kimenettel ellátott erősítő. A nagy erősítés miatt, a kimeneti feszültség meredeken csökken. Ez a pont kb. $U=1.4V$ bemeneti feszültségnél következik be. Körülbelül itt van a TTL alapkapu komparálási szintje (ahol a bemeneti és a kimeneti feszültség megegyezik egymással).

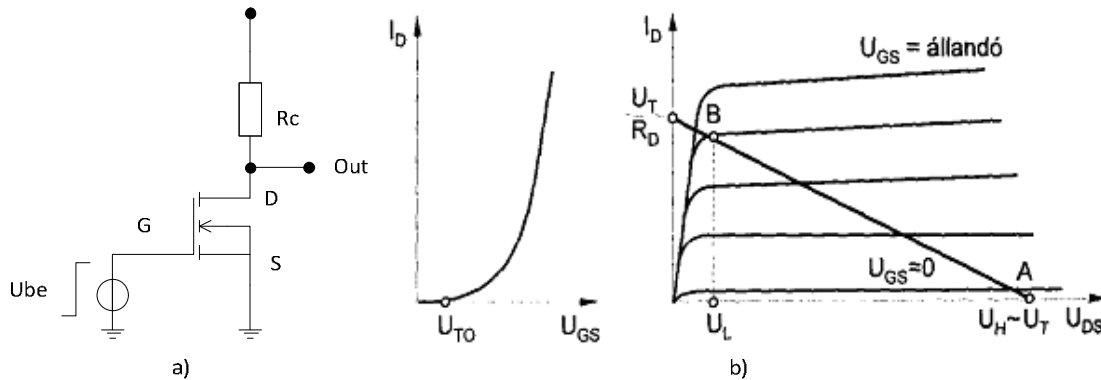
Fontos megjegyezni, hogy más altípusok esetén a komparálási feszültség eltérhet, például Schottky diódás kapcsolások esetén kicsit alacsonyabb. Az alacsonyabb komparálási feszültség nem kedvez a zavarvédelem szempontjából, viszont a Schottky diódás kapuknak sokkal meredekebb a

karakterisztikája (a II.-es szakasz rövidebb vagy hiányzik), ami viszont kedvezőbb az áramkör működése szempontjából.

A IV. szakaszban V3 zárt (az üzembiztos lezárásban segít a D3 dióda), V2 és V4 telítésben van. A kimeneti feszültség kb. 0,2 V-ra csökken. Ez felel meg a TTL alacsony logikai szint feszültségének.

CMOS logikai áramkörök

A MOS tranzisztorok felületigénye szembetűnően kisebb, mint egy bipoláris tranzisztoré. MOS áramköröknek jelentősen kisebb a teljesítményfelvétele is. A CMOS a komplementer technológiára utal, ami annyit jelent, hogy N és P csatornás félvezetőket is használnak az áramkörök kialakításánál.



3.6 ábra: MOS tranzisztor kapcsolóüzeme
a.) Elvi rajz b.) Növekményes MOS tranzisztor jelleggörbéi

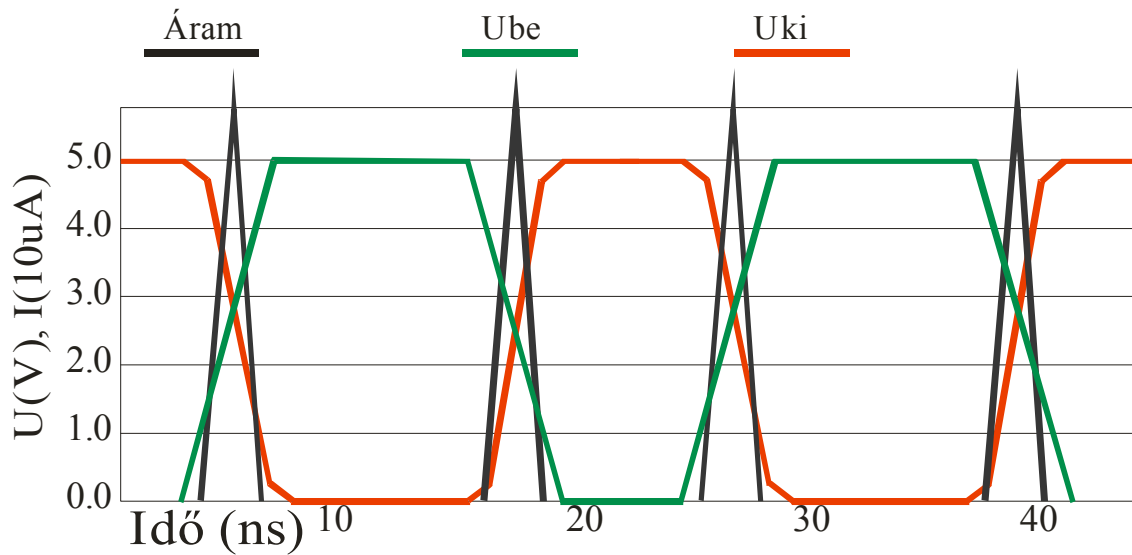
Az integrált logikai áramkörökben a MOS térvezérlésű tranzisztort alkalmazzák. A *gate* egyenáramú bemeneti ellenállása kb. $10^{14}\Omega$, amely azt jelenti, hogy a MOS tranzisztor ideális feszültségvezérelt elemként viselkedik. A bipoláris tranzisztorhoz hasonlóan használható kétállapotú kapcsoló üzemmódban is.

Hatalmas előnye ennek a gyártástechnológiának, hogy 0 Hz-en közel nulla az áramfelvétele, a frekvencia növelésével az áramfelvétele egy határig közel lineárisan növekszik, ugyanis az átkapcsolásokkor rövid időre "összenyitnak" a MOSFET-ek, továbbá a jelváltás során a mindenhol jelen levő szórt (parazita) kapacitásokat is fel kell tölteni, illetve ki kell sütni.

A MOS áramkörök nem statikus állapotban mérhető fogyasztása tehát két részből tevődik össze:

- Egymásba vezetés – A bemenőjel felfutásának egy szakaszában mindkét tranzisztor egyszerre vezet,
- Töltés-pumpálás – Jelváltásokkor a kimeneten lévő terhelést 1-re váltáskor a „p” tranzisztoron keresztül tápfeszültségre töltjük, majd 0-ra váltáskor az „n” tranzisztoron keresztül kisütjük.

Az 500 MHz alatti üzemi frekvenciájú digitális áramkörök majdnem 100%-a erre a technológiára épül. MOS inverter bemeneti és kimeneti jelalakjai és a jelszint váltásokkor fellépő áramcsúcsok. Az áramcsúcsok a kimeneti fokozat tranzisztorainak egymásba nyitásából adódnak.

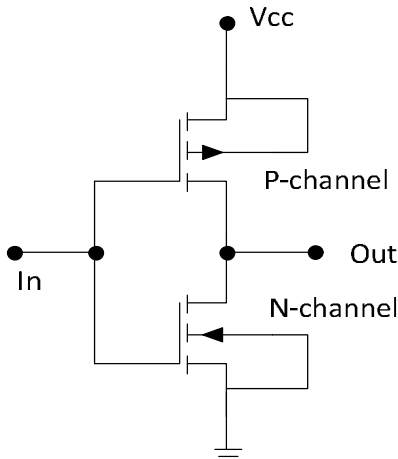


3.7 ábra: MOS inverter kapcsoló üzemenek jelalakjai

CMOS áramkörök előnyei:

- A logikai szintek egyértelműek, $U_H = U_{DD}$, $U_L = 0V$
- A statikus áramfelvétel = 0 mA
- Azonos fel-és lekapcsolási idők
- Gyors működés
- Széles tápfeszültség tartomány ($U_t = 3V-18V$)

Egy egyszerű felépítésű CMOS inverter áramkör rajza látható a következő ábrán.

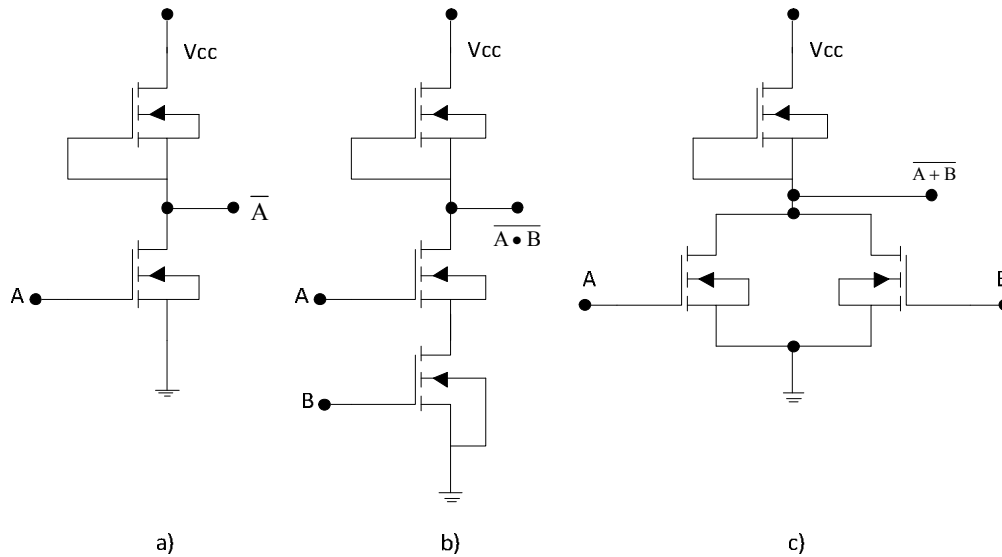


3.8 ábra: CMOS inverter áramkör rajza

A bemeneti feszültség hatására hol a felső, hol az alsó tranzistor kerül vezetés állapotba, ezzel kapcsolgatva a kimeneti jelszintet.

Alacsony bemeneti feszültség szintnél az alsó N-csatornás tranzistor lezárt állapotban van, és a P-csatornás tranzistor vezet. Ezáltal a kimenet a V_{DD} tápfeszültségre van kapcsolva. Ha a kimenetet egy következő MOS áramkör gate-elektrodája terheli le, akkor elhanyagolhatóan kis veszteségi áram folyik a tápegységből a kimenet felé, valamint a föld felé.

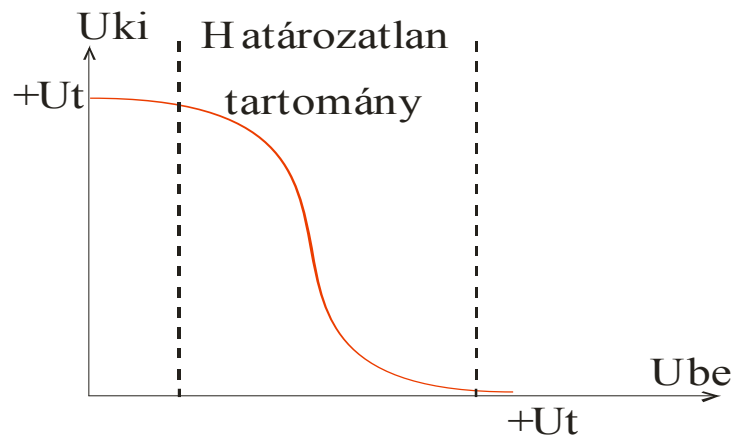
Abban az esetben, ha a bemeneti jel megközelítőleg V_{DD} értékű, akkor az N csatornás tranzisztor vezet, és a P-csatornás lezárt. A kimeneti feszültségszint 0V. Jelentősebb áramfelvétel csak akkor lép fel, amikor a kapu az egyik állapotából a másikba kapcsol át.



**3.9 ábra: Kizárólag N-csatornás félvezetőkből felépülő logikai alapkü áramkörök
a) inverter b) NEM-ÉS kapu c) NEM-VAGY kapu**

Az N-csatornás tranzisztorok működésében résztvevő negatív töltéshordozók (elektronok) mozgékonyasága majdnem háromszor nagyobb, mint a P-csatornás tranzisztorok pozitív töltéshordozóinak (lyukak) mozgékonyasága. Egy azonos meredekségű N-csatornás MOS tranzisztor felületigénye körülbelül fele egy P-csatornásénak. Ezáltal kisebb a gate-kapacitása is, ami az N-MOS áramkörök nagyobb működési sebességét eredményezi. Az N-MOS kapuk kizárólag aktív elemeket tartalmaznak. A NEM-ÉS kapu kimenetén csak akkor jelenik meg logikai '0', ha a mindkét bemeneti tranzisztor logikai '1' értékű bemenőjelet kap. A NEM-VAGY kapu kimenetén logikai '1' érték csak akkor jelenik meg, ha mindkét bemenete logikai '0' értékű bemenőjelet kap.

A MOS kapu egyenáramú bemeneti ellenállása nagyon nagy értékű. A gate úgy viselkedik, mint egy kis szivárgási árammal rendelkező kapacitás fegyverzete.



3.10 ábra: CMOS inverter transzfer karakterisztikája

A CMOS áramkörök transzfer karakterisztikájának középső szakasza nagyon meredek, ez a CMOS inverter jellegzetes előnye.

4. Logikai áramkörök fizikai jellemzői

A következő táblázatban különböző logikai áramkörrendszerek jellemző tulajdonságait hasonlítottuk össze. Látható, hogy teljesítményfelvétel szempontjából a CMOS áramkör családok tulajdonságai a legkedvezőbbek, viszont itt a legnagyobb a jelkésleltetés mértéke. Bár az ECL logika ma már nem elterjedt, itt találkozhatunk a legnagyobb működési sebességgel, aminek ára a viszonylag magas fogyasztás.

Kimeneti feszültség tekintetében a TTL áramkörrendszerek feszültségei mutatják a legnagyobb szórást. A CMOS technológia nagy előnye, hogy kimeneti feszültség szintjeik gyakorlatilag megegyeznek a tápfeszültség értékekkel.

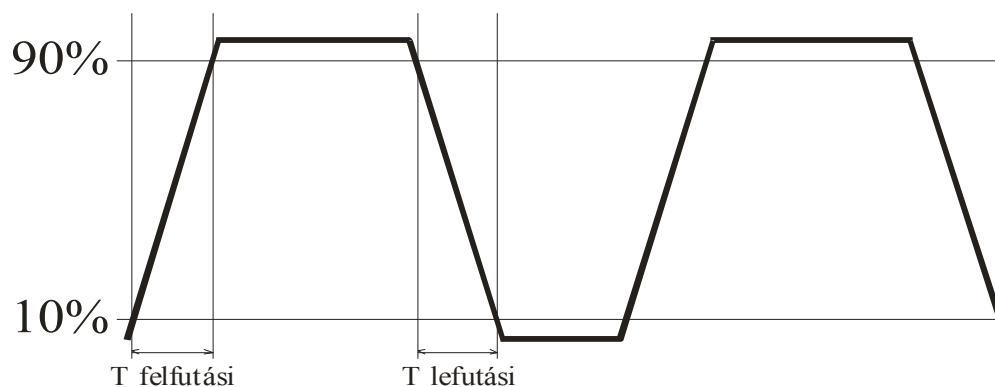
4.1 táblázat: Logikai áramkörrendszerek jellemző tulajdonságai

	Teljesítmény/kapu (mW)	Időkésleltetés (ns)	Lo level (V)	Hi level (V)
TTL	40	20	0-0,7	2,5-5
LS	10	40	0-0,7	2,5-5
CMOS	0,1	50	0-0,1	+Ut
ECL	50	5	-1,8 – -1,6	-1 – +0,2

Logikai áramkörök dinamikus jellemzői

Felfutási és lefutási idő

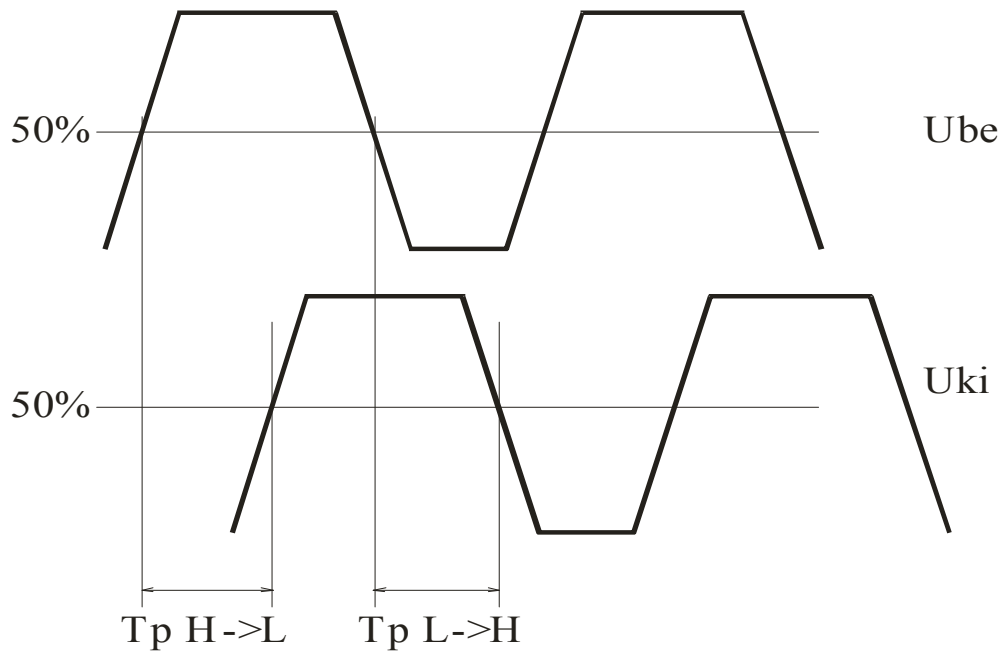
A kimeneten megjelenő jelváltozási sebesség nem végtelen. A $0 \rightarrow 1$ illetve az $1 \rightarrow 0$ átmenetek jellemzői a *felfutási* és *lefutási idők* (az angol terminológiában t_{rise} és t_{fall}). A változás kezdeténél és végénél megjelenő apró hullámok miatt a mérésüket az L és H szintek közötti feszültségkülönbség 10%-a és 90%-a között végezzük.



4.1 ábra: Jelalak: felfutási és lefutási idő

Késleltetési, vagy terjedési idő

Egy adott logikai feladat elvégzéséhez mindig bizonyos időre van szüksége az áramkörnek, ezért a bemeneti jel változását csak kissé késve követi a kimeneti jel esetleges változása. Ennek mérőszáma a *késleltetési* vagy *terjedési idő* (angolul *propagation delay*). Mérését az alábbi ábra szerint végezzük. A késleltetési idő nagyságát a kimenetre kapcsolt impedancia (főként kapacitás) is befolyásolja.



4.2 ábra: Késleltetési, vagy jelterjedési idő

Terhelhetőség (kimeneti), Fan-out

Minden logikai áramkör család esetén meg van adva, hogy egy logikai elem kimenetére legfeljebb hány másik logikai kapu bemenete csatlakoztatható (*Fan-out*). Ezt a kimenetek által leadni képes illetve a bemeneteken felvett áramok nagyságának aránya határozza meg. A TTL áramkör család áramkörei például a következő áramértékeket képesek előállítani a kimenetükön az L és H szinteken:

$$I_{kiLmax} = 16mA$$

$$I_{kiHmax} = 0.4mA$$

míg a megfelelő működésükhöz szükséges bemeneti áramok:

$$I_{beL} = 1.6mA$$

$$I_{beH} = 40\mu A$$

Ezek alapján az egy kimenetre köthető bemenetek maximális száma L és H szinten:

$$Fan-out_L = \frac{I_{kiLmax}}{I_{beL}} = 10$$

$$Fan-out_H = \frac{I_{kiHmax}}{I_{beH}} = 10$$

Az eredő terhelhetőség a kettő közül a kisebbik érték (ebben az esetben ez a két szám megegyezik):

$$Fan-out = 10$$

A terhelhetőség növelése az ugyanolyan logikai típusú áramkörök párhuzamos kapcsolásával is megoldható.

Egység-terhelés (bemeneti), Fan-in

Az áramkör családban előfordulhatnak a család többi tagjától eltérő bemeneti áramú elemek is. Az egység-terhelés azt adja meg, hogy az adott elem bemeneti áramfelvétele hány-szorosa a család többi tagjánál érvényes áramfelvételnek. Ha például a családra jellemző fan-out 10, ám egy adott elem fan-in-je 2, ebből az elemből maximálisan csak ötöt lehet egyetlen kimenetre kötni.

Egyéb jellemzők

A logikai áramköröknek még sok olyan jellemzője van, amelyeket a tervezésnél figyelembe kell venni. Most a legfontosabbakat soroljuk föl:

- *Disszipáció (teljesítményfelvétel):* az a teljesítmény, amely hővé alakul, ha a logikai áramkört 50% kitöltésű tényezőjú órajellel kapcsolgatjuk. A disszipáció bizonyos áramkör típusoknál frekvenciafüggő.
- *Jósági tényező:* az átlagos késleltetési idő és a disszipáció szorzata. Két logikai áramkör közül az a jobb, amelyik ugyanolyan teljesítményfelvétel mellett gyorsabb, illetve azonos sebességnél kevesebb energiát fogyaszt: tehát kisebb a jósági tényezője.
- *A megengedett legnagyobb és legkisebb be – és kimeneti feszültség szintek, tápfeszültség értékek.*
- *Tápfeszültség-tolerancia:* a tápfeszültség legnagyobb megengedhető ingadozása százalékban vagy feszültségértékben kifejezve.
- *A normális működéshez előírt hőmérsékleti tartomány.*
- *Tokozás:* A lábak furatba illeszthetők, vagy felületre forraszthatók-e, a tok műanyag, esetleg hőálló kerámia, stb.

A TTL és CMOS elemcsaládok összehasonlítása

A logikai áramkörök két leggyakrabban használt típusa a bipoláris tranzistorokból felépülő TTL és a térvezérlésű tranzistorokból álló CMOS család. Mindkét család különböző tokozásokban kapható, amelyek disszipációja és késleltetési ideje eltérő. Lássuk, mely paraméterek szólnak az egyik illetve a másik alkalmazása mellett!

A TTL áramkörök előnye viszonylagos gyorsasága. Késleltetési idejük kapuáramkörönként 10ns körüli standard kivitelben, de léteznek 1-2ns terjedési idejű változatok is. Teljesítményfelvételük típustól függően néhány mW-tól néhány 10mW-ig (standard) terjed. Stabil feszültségforrásra van szükségük: az általános célra gyártott darabok tápfeszültség igénye $5V \pm 5\%$, a katonai kivitelűeké $5V \pm 10\%$.

A CMOS áramkörök lassabb működésűek a TTL-eknél: a késleltetési idő standard típusoknál 100ns körüli, ami 10ns közeli értékekre csökkenthető (High Speed kivitel). Fő előnyük az alacsony teljesítményfelvétel: mivel a térvezérlésű tranzistorok bemeneti ellenállása igen nagy, a CMOS áramkörök nyugalmi állapotban gyakorlatilag nem fogyasztanak energiát (maximum néhány μW -ot). Emiatt elemes táplálású eszközökben (digitális órák, számológépek, távvezérlők, stb.) használjuk őket. Tényleges teljesítményfelvétel a logikai átkapcsoláskor történik, ezért a disszipáció teljesítményfüggő. Jellemző értéke $1\mu W/KHz$ körüli érték. Ez kb. 1 MHz feletti frekvencián meghaladja a TTL áramkörök teljesítményfelvételét, így ebben a tartományban már nem rendelkeznek az előbb említett előnnyel. A CMOS technológia másik nagy előnye a széles tápfeszültség-tartomány, amely a TTL áramkörökhöz szükséges 5V-al szemben 3-tól 18V-ig (!) terjed. Ez egy másik érv ahhoz, hogy teleppel működő eszközök tervezésénél elsősorban a CMOS technológiára gondoljunk.

E két elemcsalád mellett természetesen még számtalan különböző logikai áramkörtípussal találkozhatunk. A digitális technika szédületes iramú térnyerésének az alkatrész-technológia rohamos fejlődése adja az alapját.

Speciális kimeneti kapcsolatok TTL áramköröknél

Logikai rendszerek tervezésénél sokszor előfordul, hogy nagyon sok kimenetet kell egyetlen ÉS, esetleg VAGY kapu bemeneteire kötni. Ilyenkor rengeteg vezetékkel kell egymással párhuzamosan egy kicsiny területre vezetni, ami jelentősen bonyolítja az áramkör tervezését, de a nagyszámú bemenettel rendelkező kapuk kialakítása is gondot okoz. Szerencsére a TTL áramköröcsaládok két olyan megoldást is kínálnak, amellyel kiküszöbölhetők ezek a problémák:

Nyitott kollektoros és háromállapotú TTL áramkörök

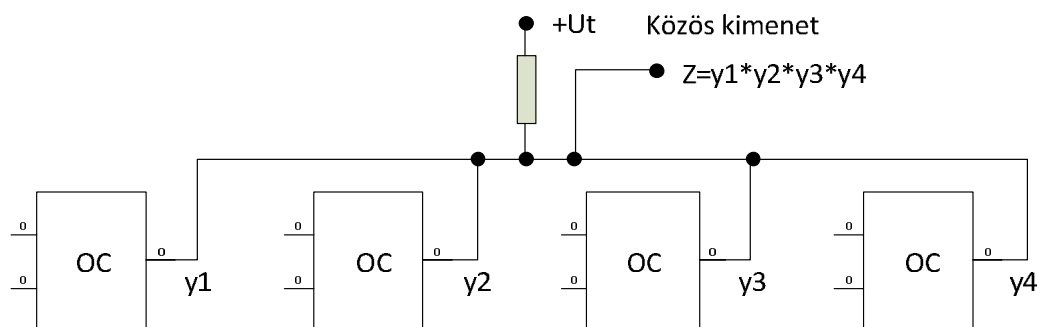
A nyitott kollektoros kimenet esetében a kapu kimeneti felhúzó ellenállásának és szintillesztő diódájának elhagyásával, nyitott kollektoros (Open Collector) kimeneti kapcsoláshoz jutunk. Működése hasonló a szabványos TTL kapuéhoz. Ennél az áramkörnél csak akkor kaphatunk logikai H szintet a kimeneten, ha beiktatunk egy külső felhúzó ellenállást a táp és a kimenet közé. Több nyitott kollektoros kapu összekapcsolásával úgynevezett huzalozott VAGY, illetve huzalozott ÉS kapcsolatot alakítható ki. A „huzalozott” elnevezés arra utal, hogy a kapuk kimeneteit vezetékkel összekötjük és a közös pontot ellenállás közbeiktatásával tápfeszültségre kötjük.

3 állapotú kimenet, ahol lehetőség van a kimeneti meghajtó fokozat leválasztására úgy, hogy a kimenet nagy impedanciás állapotba kerül. Egy E engedélyező/tiltó bemenet logikai aktív (pl. logikai '1') értékű jelet kap, addig az áramkör közös TTL kapuként működik, ha inaktív válik az engedélyező bemenet, akkor harmadik állapotba kapcsolja a kapu kimenetét, vagyis a kimenet sem alacsony, sem magas logikai szintet nem ad ki. Ebben az esetben a vele párhuzamosan kapcsolt kimeneteket működtethetjük a zárlat veszélye nélkül.

Két olyan logikai áramkör kimenete, amelyik nem nyitott kollektoros vagy nem háromállapotú, nem köthető össze. A kimeneten létrejövő feszültség szint nem tartja be a H és L szintre érvényes értékeket, ugyanakkor a kimeneti áramkörök túlterheléséhez és tönkremeneteléhez vezethet.

Nyitott kollektoros kimenetek

Léteznek olyan TTL technológiával gyártott logikai kapuk, amelyek *nyitott kollektórú (Open Collector) kimenettel* rendelkeznek. Ha több ilyen kapu kimenetét az alábbi ábra szerint összekötjük, a közös ponton úgynevezett *huzalozott ÉS kapcsolat* jön létre, megspórolva így a sok bemenetű ÉS kaput, és a párhuzamos vezetékeket.

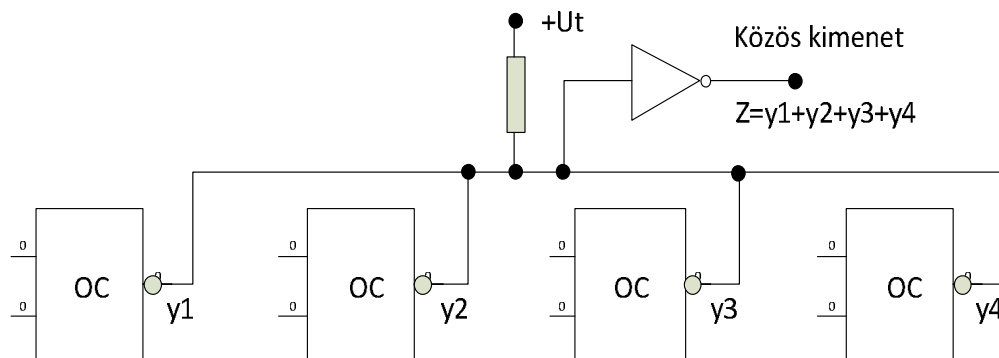


4.3 ábra: Nyitott-kollektórú kimenetek kapcsolása (ÉS kapcsolat)

Mivel a De-Morgan azonosság szabályai szerint:

$$q_1 + q_2 + \dots + q_n = \overline{\overline{q_1} \cdot \overline{q_2} \cdot \dots \cdot \overline{q_n}}$$

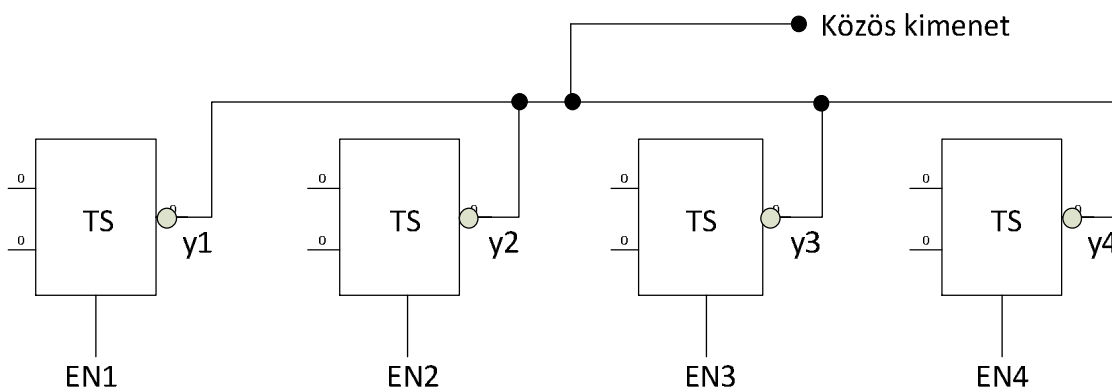
az ábrának megfelelően *huzalozott VAGY* kapcsolat is előállítható.



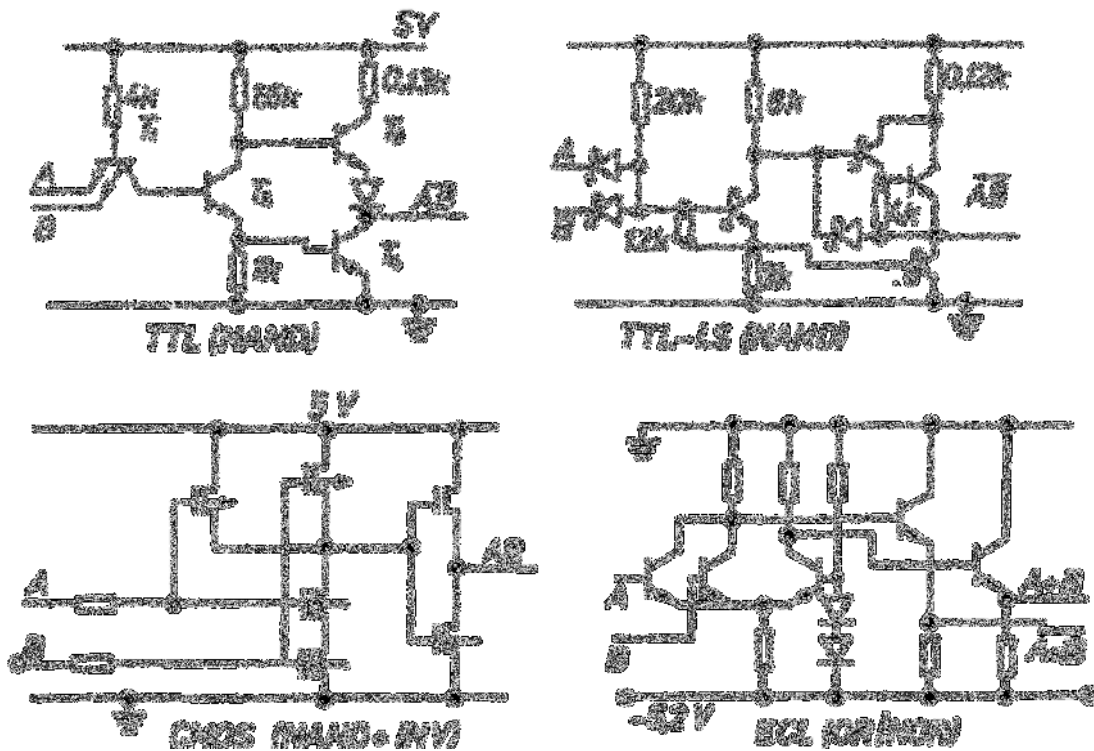
4.4 ábra: Nyitott-kollektoros kimenetek kapcsolása (VAGY kapcsolat)

Háromállapotú (Tri-State, Three-State) kimenetek

Tipikusan számítógépes hardver környezetben sokszor használnak úgynevezett *buszrendszereket*. A buszok olyan vezetékek vagy vezetékcsoportok, amelyeket több eszköz is egyidejűleg használhat, de egy időben csak egyetlen, kiválasztott áramkör hajthatja meg a vonalakat. Ilyenkor háromállapotú kimenetekkel rendelkező áramköröket alkalmazunk. Ezeknél az elemeknél egy járulékos vezérlőbemenettel „kikapcsolható” a kimenetek (nagy-impedanciás módba állíthatók), megengedve egy másik vele párhuzamosan kapcsolódó eszköznek, hogy a vezetékeken az ő általa kiadott adatokat továbbítsa. Mindig ügyelni kell arra, hogy egy buszon egyszerre csak egy eszköz lehet aktív!



4.5 ábra: Háromállapotú kimenetek összekapcsolása



4.6 ábra: NAND ill. NOR kapu különböző áramköri megvalósításokkal

A fenti ábrán különböző logikai áramkörrendszerekkel megvalósított hasonló funkciójú áramköri elemeket láthatunk. Első ránézésre is látható, hogy felépítésüket tekintve nagyon hasonlóak egymáshoz, mindegyik áramköri megoldás a rá jellemző jegyeket viseli. A TTL és TTL-LS áramkörök közötti alapvető különbség a Schottky diódák alkalmazása. Ettől eltekintve a kapcsolástechnika teljesen azonos. A CMOS áramkörök esetén nem alkalmazhatunk multi-emitteres tranzisztorokat. Helyettük a fejezet elején megismert ÉS, VAGY alaplogikákat használják. Az ECL logikai áramkörök is némi hasonlóságot mutatnak a TTL áramkör családokhoz, de itt kizárólag a földelt-kollektoros kapcsolástechnikát alkalmazzák a nagyobb sebesség elérése végett.

5. Logikai függvények egyszerűsítése

Minél egyszerűbb egy kombinációs hálózat logikai függvénye, annál kevesebb áramköri elemmel lehet megvalósítani. Minél kevesebb bemenete van összesen a megvalósításhoz használt áramköri elemeknek, annak annál kisebb lesz a mérete, annál kevesebb áramot fogyaszt. A fejezetben kétszintű logikai hálózatok tervezésének módszerei lesznek ismertetve a bemutatott célfüggvény szerint.

A függvényminimalizálásnál felhasznált alapötlet szerint általában a szomszédos mintermeket (vagy maxtermeket) kell megkeresni. Két minterm (maxterm) akkor szomszédos, ha egymástól egyetlen változóban különböznek, az egyikben ponált, a másikban negált értékkel (a többi változó azonos értékkel szerepel). A szomszédos mintermeket (maxtermeket) össze lehet vonni, azaz egyszerűsíteni lehet a logikai függvényt.

Legyen adott két minterm AB és $A\bar{B}$. Ezek egyetlen változóban (B -ben) különböznek egymástól, azaz szomszédosak. Az alábbi algebrai egyszerűsítés megmutatja, hogy ezt a két mintermet össze lehet vonni úgy, hogy kiesik az a változó, melyben különböznek egymástól. Az azonosságot igazságtábla segítségével le is lehet ellenőrizni.

$$AB + A\bar{B} = A(B + \bar{B}) = A + 1 = A$$

A	B	AB	A \bar{B}	AB + A \bar{B}
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	0	1

5.1 ábra: Az $AB + A\bar{B} = A$ azonosság ellenőrzése igazságtáblával

Legyen adott két maxterm $A + B$ és $A + \bar{B}$. Ezek egyetlen változóban (B -ben) különböznek egymástól, azaz szomszédosak. Az alábbi algebrai egyszerűsítés megmutatja, hogy ezt a két maxtermet össze lehet vonni úgy, hogy kiesik az a változó, melyben különböznek egymástól. A levezetés során fel lett használva az, hogy két szomszédos minterm egyszerűsíthető ($AB + A\bar{B} = A$), valamint ezen azonosság igazságtáblázata is látható.

$$(A + B)(A + \bar{B}) = AA + AB + A\bar{B} + B\bar{B} = A + A + 1 = A$$

A	B	A + B	A + \bar{B}	(A + B)(A + \bar{B})
0	0	0	1	0
0	1	1	0	0
1	0	1	1	1
1	1	1	1	1

5.2 ábra: Az $(A + B)(A + \bar{B}) = A$ azonosság ellenőrzése igazságtáblával

Az összevont és össze nem vont mintermek (maxtermek) neve implikánsok. Ha egy implikáns már nem vonható össze más implikánssal, akkor ezt príimplikánsnak nevezik. A logikai függvény legegyszerűbb alakja príimplikánsokat tartalmaz.

Algebrai módszer

Az algebrai egyszerűsítés során a Boole-algebra azonosságai használhatók fel a logikai függvény legegyszerűbb alakjának eléréséhez.

Boole algebra

A Boole-algebra a logikai operátorok algebrája. George Boole (1815-1864) angol matematikus először mutatott hasonlóságot az általa vizsgált logikai operátorok és a már jól ismert aritmetikai operátorok között. Az általa kidolgozott logikai algebrát Boole algebrának nevezik, mely nagyban hozzájárult a

számítástechnika fejlődéséhez. Elsősorban a hardver tervezés alacsonyabb szintjén van rendkívül fontos szerepe.

A Boole-algebra a kétértékű jelekkel (0 vagy 1) végzett logikai műveletek (ÉS, VAGY, NEM) algebrai leírását teszi lehetővé. Egy Boole algebrai kifejezés kiértékelése során először a NEM, majd az ÉS végül a VAGY műveleteket kell kiértékelni. Zárójelzéssel természetesen módosítható a műveletek elvégzési sorrendje.

Logikai szorzás, ÉS kapcsolat azonosságai:

- $0 \cdot 0 = 0$
- $0 \cdot 1 = 0$
- $1 \cdot 0 = 0$
- $1 \cdot 1 = 1$
- $A \cdot 1 = A$
- $A \cdot 0 = 0$
- $AA = A$ (idempotencia)
- $A\bar{A} = 0$
- $AB = BA$ (kommutativitás)
- $(AB)C = A(BC) = ABC$ (asszociativitás)

Logikai összeadás, VAGY kapcsolat azonosságai:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 1$
- $A + 0 = A$
- $A + 1 = 1$
- $A + A = A$ (idempotencia)
- $A + \bar{A} = 1$
- $A + B = B + A$ (kommutativitás)
- $(A + B) + C = A + (B + C) = A + B + C$ (asszociativitás)

Logikai tagadás, NEM (negáció) azonosságai:

- $\bar{0} = 1$
- $\bar{1} = 0$
- $\bar{\bar{A}} = A$

További azonosságok:

- $A(B + C) = AB + AC$ (disztributivitás)
- $A + BC = (A + B)(A + C)$ (disztributivitás)
- $A + AB = A$ (elnyelési tulajdonság)
- $A(A + B) = A$ (elnyelési tulajdonság)
- $AB + A\bar{B} = A$
- $(A + B)(A + \bar{B}) = A$
- $A + \bar{A}B = A + B$
- $A(\bar{A} + B) = AB$

De-Morgan azonosságok

- $\overline{A + B} = \bar{A}\bar{B}$
- $\overline{A + B + C} = \bar{A}\bar{B}\bar{C}$
- ...
- $\overline{AB} = \bar{A} + \bar{B}$

- $\overline{ABC} = \bar{A} + \bar{B} + \bar{C}$
- ...

Bármelyik azonosságot könnyedén lehet igazolni, például igazságtábla segítségével vagy a logikai kifejezések kibontásával. Igazoljuk például a $A + \bar{A}B = A + B$ azonosságot. Az egyenlet bal oldalának kifejtésekor először meg kell határozni \bar{A} értékét, majd ebből $\bar{A}B$ értékét, végül a teljes kifejezés értéke is meghatározható. Az egyenlet jobb oldala egy VAGY művelet, melynek igazságtáblája már ismert. A két kifejezés kiértékelése alapján kijelenthető, hogy ekvivalensek, mivel egyenlők, bármilyen értéket vesz fel A és B .

A	B	\bar{A}	$\bar{A}B$	$A + \bar{A}B$
0	0	1	0	0
0	1	1	1	1
1	0	0	0	1
1	1	0	0	1

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

5.3 ábra: Az $A + \bar{A}B = A + B$ azonosság ellenőrzése igazságtáblával

Egy logikai kifejezést több féle képen is lehet egyszerűsíteni. Legyen $\overline{AB + \bar{A}B + A\bar{B}}$ az egyszerűsítendő kifejezés. Első lépésben a De-Morgan azonosság segítségével eltüntethető a teljes kifejezés negálása. Eredményül három negált minterm ÉS kapcsolata adódik: $(\overline{AB})(\overline{\bar{A}B})(\overline{A\bar{B}})$. Ahhoz, hogy a mintermek ne legyenek negálva, ezeken is külön-külön alkalmazni kell a De-Morgan azonosságot. Ez a függvény konjunktív normál formáját eredményezi: $(\bar{A} + \bar{B})(A + \bar{B})(\bar{A} + B)$. Következő lépésben a disztributivitás tulajdonságot kihasználva össze lehet vonni a második két zárójelet: $(\bar{A} + \bar{B})(A\bar{A} + AB + \bar{A}\bar{B} + \bar{B}B)$. Ezután kihasználható, hogy egy logikai változó ÉS kapcsolata a változó negáltjával 0-át és egy logikai változó VAGY kapcsolata 1-el a változót adja eredményül: $(\bar{A} + \bar{B})(AB + \bar{A}\bar{B})$. Ötödik lépésben a disztributivitást ismét felhasználva felbonthatóak a zárójelek: $\bar{A}AB + \bar{A}\bar{A}\bar{B} + AB\bar{B} + \bar{A}\bar{B}\bar{B}$. Ezután ismét felhasználható az előző tulajdonság valamint az, hogy bármely kifejezés ÉS kapcsolata 0-val 0: $\bar{A}\bar{A}\bar{B} + \bar{A}\bar{B}\bar{B}$. Az ÉS művelet idempotencia tulajdonsága alapján mind a két szorzatot egyszerűsíthető: $\bar{A}\bar{B} + \bar{A}\bar{B}$. Végül a VAGY művelet idempotencia tulajdonsága alapján ismét egyszerűsíteni lehet: $\bar{A}\bar{B}$. Az egyszerűsítés teljes folyamata alább látható.

$$\begin{aligned}
 \overline{AB + \bar{A}B + A\bar{B}} &= \\
 (\overline{AB})(\overline{\bar{A}B})(\overline{A\bar{B}}) &= \\
 (\bar{A} + \bar{B})(A + \bar{B})(\bar{A} + B) &= \\
 (\bar{A} + \bar{B})(A\bar{A} + AB + \bar{A}\bar{B} + \bar{B}B) &= \\
 (\bar{A} + \bar{B})(AB + \bar{A}\bar{B}) &= \\
 \bar{A}AB + \bar{A}\bar{A}\bar{B} + AB\bar{B} + \bar{A}\bar{B}\bar{B} &= \\
 \bar{A}\bar{A}\bar{B} + \bar{A}\bar{B}\bar{B} &= \\
 \bar{A}\bar{B} + \bar{A}\bar{B} &= \\
 \bar{A}\bar{B} &
 \end{aligned}$$

Ugyanez a logikai függvény másként is egyszerűsíthető, most más sorrendben használva a Boole algebrai azonosságokat. Első lépésben kihasználható a disztributivitás tulajdonság így az első két mintermből kiemelhető B : $(A + \bar{A})B + \bar{A}\bar{B}$. A logikai VAGY kapcsolat idempotencia tulajdonsága alapján az $(A + \bar{A})$ kifejezés kiesik: $B + \bar{A}\bar{B}$. Következő lépésben az $A + \bar{A}B = A + B$ azonosságot felhasználva a kifejezés átalakítható: $B + \bar{A}$. Majd utolsó lépésben egy De-Morgan azonosság segítségével megkapható a végső formát: $\bar{A}\bar{B}$. Az egyszerűsítés teljes menete alább látható.

$$\overline{AB + \bar{A}B + A\bar{B}} =$$

$$\begin{aligned} \overline{(A + \bar{A})B + A\bar{B}} &= \\ \overline{B + A\bar{B}} &= \\ \overline{B + A} &= \\ \bar{A}\bar{B} & \end{aligned}$$

Mind a példából is látszik, ugyanazt az eredményt többféle képen is elérhető algebrai minimalizálás esetén. A fenti példában az első levezetés nyolc lépést igényelt, a második viszont csak fele annyit. Annak eldöntése, hogy a levezetés melyik lépésében melyik azonosságot érdemes használni nagyfokú tapasztalatot igényel, így a módszer nehezen automatizálható.

Ha a fejezet elején bemutatott szomszédos mintermek (maxtermek) módszerét akarjuk használni, akkor minden logikai kifejezést először át kell alakítani diszjunktív (konjunktív) normál formába és utána összevonni őket. Gyakorlatilag ez történt az első levezetés során. Ebben az esetben figyelni kell arra, hogy egy minterm több mintermmel is összevonható és minden lehetséges összevonást el kell végezni, de nem feltétlenül van szükség az összes prímisszorzóra. Ilyen esetekre a későbbiekben látható majd példa.

Kifejtési módszer

Az algebrai módszer komplexebb, több változót tartalmazó logikai kifejezések $F(x_1, x_2, \dots, x_n)$ egyszerűsítése esetén nehezen alkalmazható. Ilyen esetekben alkalmazható a kifejtési módszer. Ekkor egy adott változó értékét először ponálnak $F(1, x_2, \dots, x_n)$, majd negálnak $F(0, x_2, \dots, x_n)$ kell definiálni. Ezután a két egyszerűbb, kevesebb változót tartalmazó függvényt egyszerűsíthető (lehet rekurzívan kifejtési módszert is alkalmazni). Az eredeti függvény egyszerűsített formája kétféleképpen is megkapható.

- A változó ponált rögzítésével kapott függvényhez ÉS logikai kapcsolattal hozzá kell fűzni a változót ponált értékben, a negált rögzítéssel kapott függvényhez pedig negált értékben. Végül az így kiszámított két logikai kifejezést VAGY logikai művelettel kell összekötni. $F(x_1, x_2, \dots, x_n) = x_1 \cdot F(1, x_2, \dots, x_n) + \bar{x}_1 \cdot F(0, x_2, \dots, x_n)$
- A változó negált rögzítésével kapott függvény negáltjához VAGY logikai kapcsolattal kell hozzáfűzni a változót ponált értékben, a ponált rögzítéssel kapott függvény negáltjához pedig negált értékben. Végül az így kiszámított két logikai kifejezést ÉS logikai művelettel kell összekötni és negálni. $F(x_1, x_2, \dots, x_n) = \overline{(x_1 + F(0, x_2, \dots, x_n)) \cdot (\bar{x}_1 + F(1, x_2, \dots, x_n))}$

Legyen adott diszjunktív normál formában egy háromváltozós F függvény, ahol $F(A, B, C) = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$. Legyen az A változó értéke először 1, ekkor $F(1, B, C) = 0B\bar{C} + 0BC + 1\bar{B}\bar{C} + 1B\bar{C} = \bar{B}\bar{C} + B\bar{C} = (\bar{B} + B)\bar{C} = \bar{C}$. Ezek után legyen A értéke 0, azaz $F(0, B, C) = 1B\bar{C} + 1BC + 0\bar{B}\bar{C} + 0B\bar{C} = B\bar{C} + BC = B(\bar{C} + C) = B$.

- Az egyszerűsített logikai függvény meghatározása az első módszer szerint, azaz $F(A, B, C) = A \cdot F(1, B, C) + \bar{A} \cdot F(0, B, C) = A\bar{C} + \bar{A}B$.
- Az egyszerűsített logikai függvény meghatározása a második módszer szerint, azaz $F(A, B, C) = \overline{(A + F(1, B, C)) \cdot (\bar{A} + F(0, B, C))} = \overline{(A + \bar{C})(\bar{A} + B)} = \bar{A}B + A\bar{C}$.

Kifejtési módszer esetében fontos, hogy melyik változó kerül rögzítésre, ennek hatékony kiválasztása az algebrai módszerhez hasonlóan nagy gyakorlatot igényel.

Grafikus módszer

Mint az korábban kiderült, az algebrai egyszerűsítés és a kifejtési módszer sikere nagyban függ a számítást végző gyakorlatától, vagy attól, hogy éppen mennyire tud az adott feladatra koncentrálni. További hátrányuk a fenti módszereknek, hogy nem vagy csak nehezen tudják kezelni a nem teljesen definiált hálózatokat.

A következőkben ismertetett módszer az emberi tényezőket kizárja, teljes mértékben automatizálható és nem teljesen specifikált hálózatok esetében is használható. A módszer szomszédos mintermeket (implikánsokat) keres és azokat összevonja az egyszerűsítés során.

Logikai függvény grafikus minimalizálása

A korábbi fejezetekben megismert Karnaugh táblában minden cella egy mintermnek (maxtermnek) felel meg. Ahol a táblázatban 1 van, az a minterm szerepel (maxterm nem szerepel) az egyszerűsítendő függvényben, ahol 0, az nem (az igen). A tábla szomszédos cellái, a táblázat felépítése alapján, szomszédos mintermeket (maxtermeket) tartalmaznak. Így könnyű felfedezni a szomszédos mintermeket (maxtermeket) és így egyszerűsíteni is a függvényt.

Legyen $\bar{A}\bar{B}C + \bar{A}BC$ logikai függvény adott. A függvényben szereplő két minterm szomszédos mivel csak a B változó értékében különböznek. A függvény egyszerűsítése ez alapján a következőképpen néz ki $\bar{A}\bar{B}C + \bar{A}BC = \bar{A}C(\bar{B} + B) = \bar{A}C$. Karnaugh táblán az összevonni kívánt mintermek egy hurokkal (tömbbel) összeköthetők.

		C			
		B			
A	BC	00	01	11	10
	0	0	1	1	0
A	1	0	0	0	0
		4	5	7	6

5.4 ábra: Példa kettes hurokra

Az ábrából látszik, hogy a kettes hurkok implikánsokat jelölnek. A szomszédos kettes hurkokat is össze lehet vonni négyes hurkokká, a négyes hurkokat nyolcas hurkokká, és így tovább. Az alábbi ábrán az $\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC$ függvény Karnaugh táblája látható. A függvény egyszerűsítése az ábra alapján a következőképpen néz ki $\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC = \bar{A}C(\bar{B} + B) + AC(\bar{B} + B) = \bar{A}C + AC = C(\bar{A} + A) = C$.

		C			
		B			
A	BC	00	01	11	10
	0	0	1	1	0
A	1	0	1	1	0
		4	5	7	6

5.5 ábra: Példa négyes hurokra

A Karnaugh tábla jellegzetessége, hogy a táblázat ugyanabban a sorában (oszlopában) lévő két szélén (alul és felül) lévő cellák is szomszédos termeket tartalmaznak, azaz léteznek olyan hurkok, melyek a táblázat szélén túlnyúlnak. Ez az $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$ függvény egyszerűsítésén keresztül lesz bemutatva. A függvény egyszerűsítésének menete $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C = \bar{A}\bar{C}(\bar{B} + B) = \bar{A}\bar{C}$ az ábrán is látható.

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

5.6 ábra: Példa túlnyúló kettes hurokra

Természetesen a túlnyúló szomszédos kettes hurkokat is össze lehet vonni. Az alábbi ábrán az $\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$ függvény Karnaugh táblája látható. A függvény egyszerűsítése az ábra alapján a következőképpen néz ki $\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} = \bar{A}\bar{C}(\bar{B} + B) + A\bar{C}(\bar{B} + B) = \bar{A}\bar{C} + A\bar{C} = \bar{C}(\bar{A} + A) = \bar{C}$.

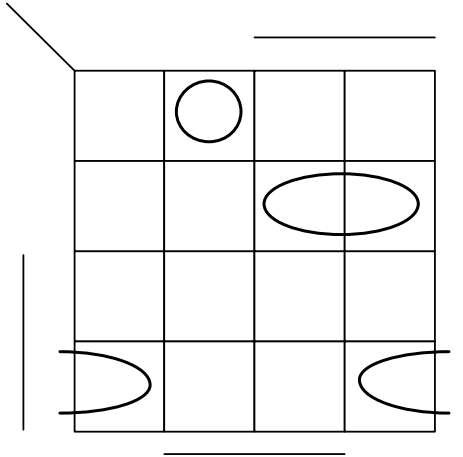
		C			
		B			
A	BC	00	01	11	10
	0	1	0	0	1
1	1	0	0	1	

5.7 ábra: Példa túlnyúló négyes hurokra

Implikánsok kiolvasása a Karnaugh táblából

A Karnaugh táblából az egyszerűsített függvény diszjunktív és konjunktív formájú alakját is ki lehet olvasni. Diszjunktív alak kiolvasásakor az 1-eseket kell hurkolni, konjunktív alak esetén a 0-ásokat. A hurkokat nem szükséges algebrailag is levezetni, mint az előző példákban, hanem a prímisszimplicánsok értéke közvetlenül is leolvasható a Karnaugh táblázatból. A leolvasáshoz a Karnaugh tábla peremmezését lehet használni. Meg kell vizsgálni, hogy az adott hurok mely perem alatt van bent teljes egészében és mely peremek alatt nincs bent egyáltalán. Amely változóhoz tartozó perem teljesen lefedi a hurkot, az diszjunktív esetben ponált formában, konjunktív esetben negált formában szerepel a kifejezésben. Amely változóhoz tartozó perem alatt nincs bent a hurok, az diszjunktív esetben negált, konjunktív esetben ponált formában szerepel a kifejezésben. Azok a változók, melyekhez tartozó perem csak részben fedi le a hurkot, nem szerepelnek a kifejezésben, ezek a változók esnek ki az egyszerűsítés során. A következő példákban különböző méretű hurkokhoz tartozó diszjunktív és konjunktív formájú logikai függvények kiolvasásának bemutatása történik négyváltozós Karnaugh táblából a kisebb hurkoktól a nagyobbak felé haladva.

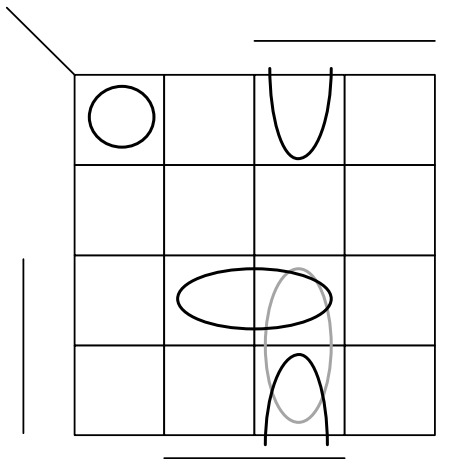
Az alábbi példa az egyes és a kettes hurkok kiolvasását mutatja be és azt, hogy mi lesz az egyszerűsítés eredménye. Az I-es hurok nincs az A, a B és a C perem alatt valamint bent van a D perem alatt, azaz az I-es hurok algebrai alakja $F_I = \bar{A}\bar{B}\bar{C}D$. A II-es hurok nincs az A perem alatt, bent van a B és a C perem alatt valamint a D perem csak részben fedi le, azaz a II-es hurok algebrai alakja $F_{II} = \bar{A}BC$. A III-as hurok nincs a B és a D perem alatt, bent van az A perem alatt valamint a C perem csak részben fedi le, azaz a III-as hurok algebrai alakja $F_{III} = A\bar{B}\bar{D}$. Az egyszerűsített logikai függvény algebrai alakja $F = F_I + F_{II} + F_{III} = \bar{A}\bar{B}\bar{C}D + \bar{A}BC + A\bar{B}\bar{D}$.



$$F = F_I + F_{II} + F_{III} = \bar{A}\bar{B}\bar{C}D + \bar{A}BC + A\bar{B}\bar{D}$$

5.8 ábra: Példa egyes és kettes hurok kiolvasására diszjunktív alakban

A következő példa konjunktív alakok kiolvasását mutatja be. Ebben az esetben nem az 1-esek, hanem a 0-ások vannak hurkolva. Az ábrán az összes lehetséges hurok jelölve van, de a szürkével jelölt IV-es hurokra nincs szükség, mivel az általa lefedett maxtermekeket más hurkok is lefedik (erről később részletesebben lesz szó). Az I-es hurok nincs az A, a B, a C és a D perem alatt sem, azaz az I-es hurok algebrai alakja $F_I = A + B + C + D$. A II-es hurok bent van az A, a B és a D perem alatt valamint a C perem csak részben fedi le, azaz a II-es hurok algebrai alakja $F_{II} = \bar{A} + \bar{B} + \bar{D}$. A III-as hurok nincs a B perem alatt, bent van a C és a D perem alatt valamint az A perem csak részben fedi le, azaz a III-as hurok algebrai alakja $F_{III} = B + \bar{C} + \bar{D}$. Az egyszerűsített logikai függvény algebrai alakja $F = F_I \cdot F_{II} \cdot F_{III} = (A + B + C + D)(\bar{A} + \bar{B} + \bar{D})(B + \bar{C} + \bar{D})$.

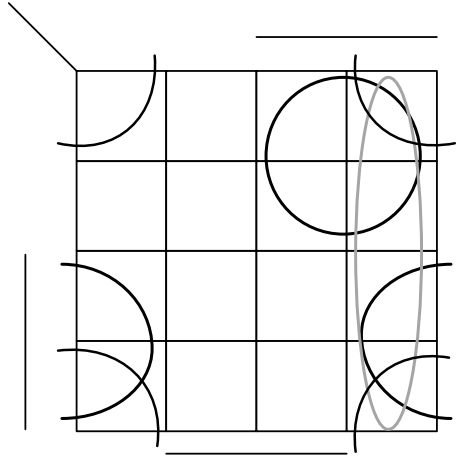


$$F = F_I \cdot F_{II} \cdot F_{III} = (A + B + C + D)(\bar{A} + \bar{B} + \bar{D})(B + \bar{C} + \bar{D})$$

5.9 ábra: Példa egyes és kettes hurok kiolvasására konjunktív alakban

A négyes hurok kiolvasása a következő képen működik diszjunktív alak estén. Az ábrán az összes lehetséges négyes hurok jelölve van, de a szürkével jelölt IV-es hurokra ebben a példában nincs szükség, mivel az általa lefedett maxtermekeket más hurkok is lefedik. Az I-es hurok nincs a B és a D perem alatt valamint az A és a C perem csak részben fedi le, azaz az I-es hurok algebrai alakja

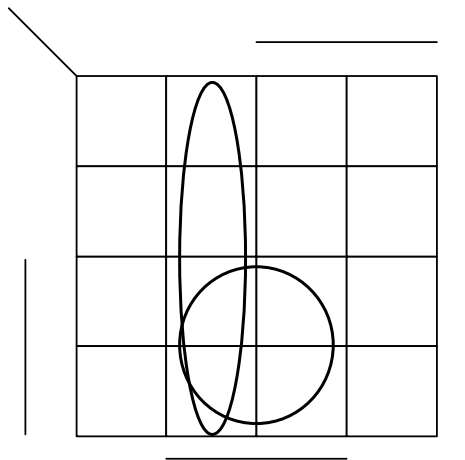
$F_I = \bar{B}\bar{D}$. A II-es hurok nincs az A perem alatt, bent van a C perem alatt valamint a B és a D perem csak részben fedi le, azaz a II-es hurok algebrai alakja $F_{II} = \bar{A}C$. A III-as hurok nincs a D perem alatt, bent van az A perem alatt valamint a B és a C perem csak részben fedi le, azaz a III-as hurok algebrai alakja $F_{III} = A\bar{D}$. Az egyszerűsített logikai függvény algebrai alakja $F = F_I + F_{II} + F_{III} = \bar{B}\bar{D} + \bar{A}C + A\bar{D}$.



$$F = F_I + F_{II} + F_{III} = \bar{B}\bar{D} + \bar{A}C + A\bar{D}$$

5.10 ábra: Példa négyes hurok kiolvasására diszjunktív alakban

Konjunktív esetben hasonlóan történik a négyes hurok kiolvasása. Az I-es hurok nincs a C perem alatt, bent van a D perem alatt, valamint az A és a B perem csak részben fedi le, azaz az I-es hurok algebrai alakja $F_I = C + \bar{D}$. A II-es hurok bent van az A és a D perem alatt valamint a B és a C perem csak részben fedi le, azaz a II-es hurok algebrai alakja $F_{II} = \bar{A} + \bar{D}$. Az egyszerűsített logikai függvény algebrai alakja $F = F_I \cdot F_{II} = (C + \bar{D})(\bar{A} + \bar{D})$.

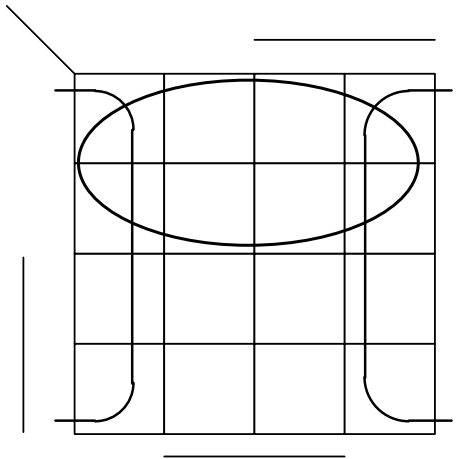


$$F = F_I \cdot F_{II} = (C + \bar{D})(\bar{A} + \bar{D})$$

5.11 ábra: Példa négyes hurok kiolvasására konjunktív alakban

Következzenek a nyolcas hurok, azon belül először a diszjunktív alak kiolvasása. Az I-es hurok nincs a D perem alatt valamint az A, a B és a C perem csak részben fedi le, azaz az I-es hurok algebrai alakja

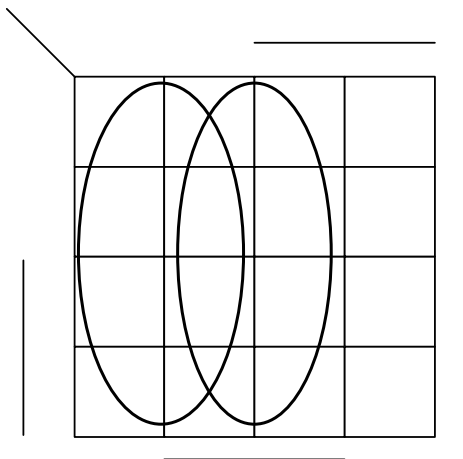
$F_I = \bar{D}$. A II-es hurok nincs az A perem alatt, valamint a B, a C és a D perem csak részben fedi le, azaz a II-es hurok algebrai alakja $F_{II} = \bar{A}$. Az egyszerűsített logikai függvény algebrai alakja $F = F_I + F_{II} = \bar{D} + \bar{A}$.



$$F = F_I + F_{II} = \bar{D} + \bar{A}$$

5.12 ábra: Példa nyolcas hurok kiolvasására diszjunktív alakban

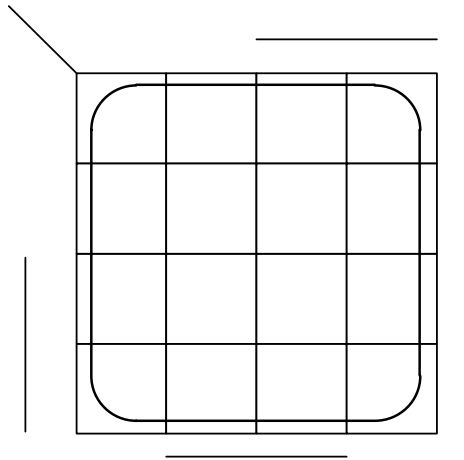
Legyen egy hasonló példa konjunktív esetre is. Az I-es hurok nincs a C perem alatt, valamint az A, a B és a D perem csak részben fedi le, azaz az I-es hurok algebrai alakja $F_I = C$. A II-es hurok bent van a D perem alatt valamint az A, a B és a C perem csak részben fedi le, azaz a II-es hurok algebrai alakja $F_{II} = \bar{D}$. Az egyszerűsített logikai függvény algebrai alakja $F = F_I \cdot F_{II} = (C)(\bar{D})$. A kapott alak egy speciális konjunktív alak, ahol minden maxterm pontosan egy változót tartalmaz, azaz a szorzatuk egy mintermet eredményez. Ha a diszjunktív alakot közvetlenül szeretnénk kiolvasni ugyanebben a példában, akkor azt egy $C\bar{D}$ hurok segítségével tehetnénk meg, azaz az eredmény ugyanaz lenne, mint konjunktív esetben.



$$F = F_I \cdot F_{II} = (C)(\bar{D})$$

5.13 ábra: Példa nyolcas hurok kiolvasására konjunktív alakban

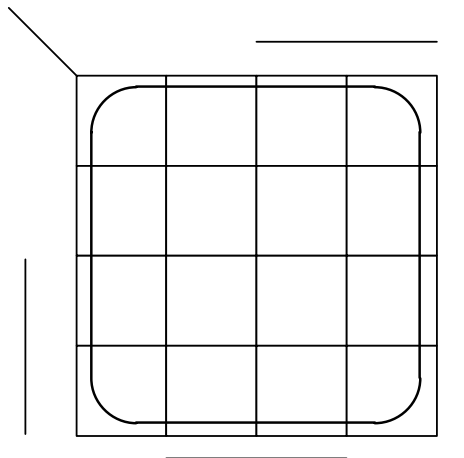
Négyváltozós logikai függvénynél tizenhatos hurok csak akkor alakulhat ki diszjunkt esetben, ha a Karnaugh táblában minden cellában 1 szerepel, azaz konstans 1 a függvény értéke $F = 1$.



$$F = 1$$

5.14 ábra: Példa tizenhatos hurok kiolvasására diszjunktív alakban

Konjunktív esetben a Karnaugh tábla minden cellájában 0 van tizenhatos huroknál, azaz a függvény a konstans 0, $F = 0$.



$$F = 0$$

5.15 ábra: Példa tizenhatos hurok kiolvasására konjunktív alakban

Hurkok felrajzolásának szabályai

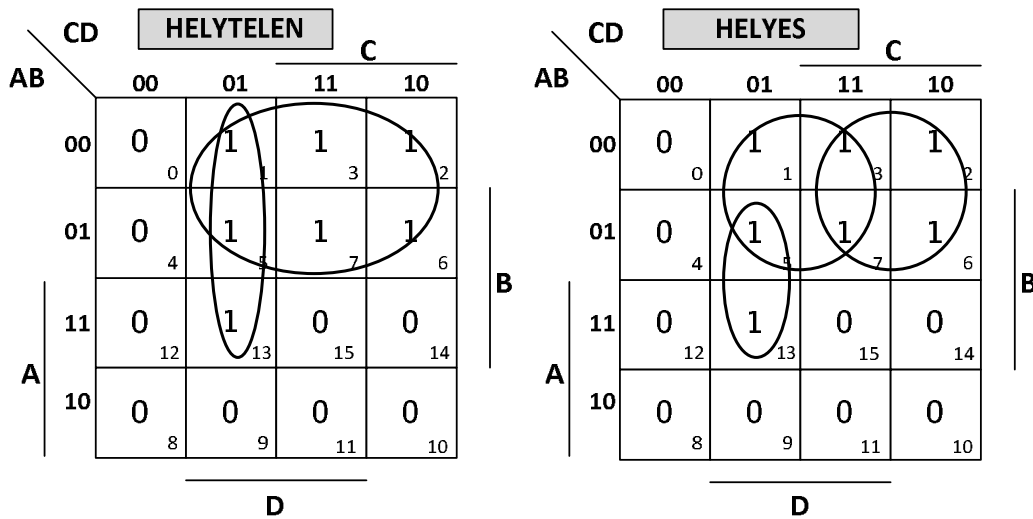
A hurkok optimális felrajzolásának, vagy más néven a tömbösítésnek, vannak szabályai, amelyeket az egyszerűsítés során be kell tartani.

- 2^n ($n = 0, 1, 2, \dots$) darab minterm (maxterm) vonható be egy hurokba.
- A hurkok téglalap alakúak.
- Minden 1-est (maxtermek esetében 0-ást) tartalmaznia kell legalább egy huroknak és 0 (maxterm esetében 1) nem kerülhet hurokba.
- Egyik hurok a másikat nem tartalmazhatja teljes mértékben.

- Egy term több hurokban is szerepelhet.
- A hurok felrajzolásánál törekedni kell a lehető legnagyobb méretű hurok használatára.
- Közömbös ('-') kimeneti függvényértékeket a jobb (optimálisabb) lefedésnek megfelelően kell megválasztani (NTSH).

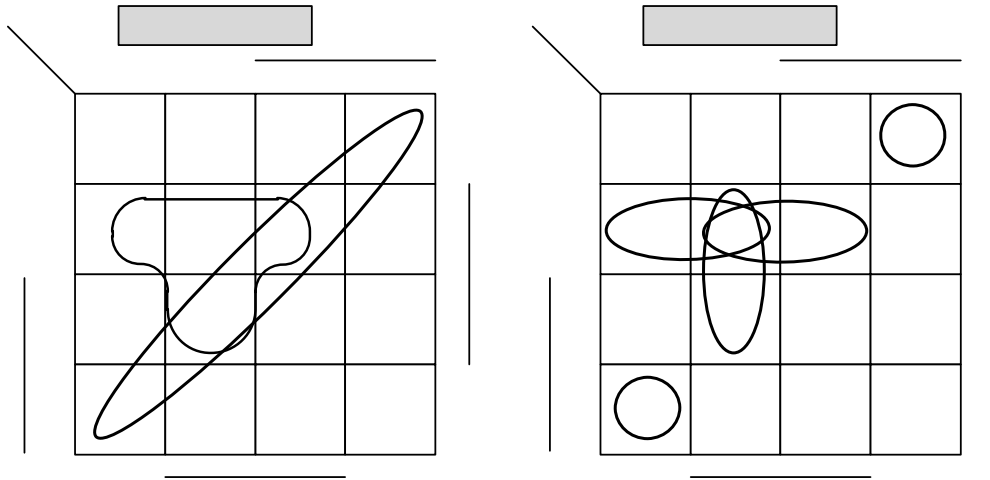
A következőkben a szabályok részletes elemzése történik meg. A használt példákban diszjunktív formát kell kiolvasni a Karnaugh táblákból. A szabályok természetesen konjunktív esetre is vonatkoznak és példákban az 1-ek és 0-ák felcserélésével meg lehet kapni az ezekre az esetekre vonatkozó példákat.

Az első szabály szerint 2^n ($n = 0, 1, 2, \dots$) darab minterm vonható be egy hurokba, azaz nem lehet hármas, ötös, hatos, hetes, kilences, tízes ... hurok a Karnaugh táblában. Ez szabály az egyszerűsítés alapötletéből következik azaz, hogy azon mintermeket lehet párosával összevonni, melyek csak egyetlen változóban különböznek. A korábbi példákból is jól látszik, hogy egyes hurokokról indulva minden összevonásnál az összevont cellák száma duplázódik, így minden hurokban kettő hatványa számú cella lesz. Az alábbi két Karnaugh táblából az elsőben van egy hármas és hatos hurok is, amely természetesen nem megfelelő. A helyes hurkolás a második Karnaugh táblázatban látható.



5.16 ábra: Példa a hurkolás első szabályának alkalmazására

A második szabály szerint a hurok téglalap alakúak. Mint az előző feltétel ez is az összevonás alapötletéből következik. Csak azon mintermek vonhatók össze, melyek szomszédosak és a szomszédos mintermek szomszédos cellákban helyezkednek el, azaz átlós hurokot nem szabad készíteni. Hasonló megfontolások alapján csak azon hurok vonhatók össze nagyobb hurokká, melyek a kapcsolódó felületükön végig szomszédosak. A következő két Karnaugh táblában az első szabályt (2^n darab cella tartozik egy hurokhoz) betartjuk, de a hurok nem téglalap alakúak.



5.17 ábra: Példa a hurkolás második szabályának alkalmazására

A következő szabály szerint minden 1-est tartalmaznia kell legalább egy huroknak és 0 nem kerülhet hurokba. Ha ez a feltétel nem teljesül, akkor az eredő függvény nem lesz ekvivalens az eredeti függvénnyel. Ha létezik olyan 1-es, melyet nem tartalmaz egyetlen hurok sem, akkor az ehhez tartozó minterm nem fog szerepelni az egyszerűsített függvényben, azaz 0 értéket fog felvenni 1 helyett, ha a bemeneti változók az adott mintermhez tartozó értékeket veszik fel. Ha viszont van olyan 0-ás, melyet tartalmaz egy hurok, akkor az eredő függvény ezen bemenethez tartozó értéknél 0 helyett 1-es értéket fog felvenni.

Ha a többi feltétel közül valamelyik (vagy akár több is) nem teljesül, akkor az eredő logikai függvényük ugyan ekvivalens lesz az eredetivel, viszont nem a lehető legegyszerűbb formát kapjuk.

- Ha van olyan hurok, amelyet teljesen egészében tartalmaz egy másik, akkor az redundanciát okoz a rendszerben, mert a befoglalt hurok logikai funkcióját a nagyobb ellátja. Algebrai forma esetében ez azt jelentené, hogy hiába lett összevonva egy implikánst egy másikkal, az továbbra is benne marad a függvényben, azaz például $AB + A\bar{B}$ egyszerűsítése nem A -t, hanem $AB + A$ -t eredményezne. A szabály betartásával az összes eredményül kapott hurok egy prímisszimplikánst fog reprezentálni.
- Ha minden term csak pontosan egy hurokban szerepelhetne, akkor nem lehetne az összes lehetséges összevonást elvégezni. Minden mintermnek annyi szomszédja van, ahány bemeneti változója van a függvénynek, azaz ennyi kettes hurokban is szerepelhet. Például az előző példa második Karnaugh táblájában az 5-ös minterm három kettes hurokban is szerepel. Ha ezt nem lehetne, akkor egy kettes hurok mellett két egyes hurkot kellene felvenni, ami bonyolultabb kapcsolást eredményezne.
- A lehető legnagyobb hurkok felvételének szabálya egyértelmű, mivel minél nagyobbak a hurkok az annál több egyszerűsítési lépést jelent, annál kevesebb a benne szereplő műveletek és változók száma.
- Mint az már korábban látható volt, a nem teljesen specifikált hálózatok esetén a különböző kimeneti függvényértékek megválasztása tetszőleges, mivel feltételezhető, hogy ilyen bemeneti kombináció nem fordul elő a rendszerben. Ez azt eredményezi, hogy ezek a cellák bevonhatók hurokba és ez nagyobb hurokokat eredményez, azaz egyszerűbb függvényalakot. A hurokba bevont mintermekhez tartozó bemenet esetén a megvalósított hálózat 1, a hurokba nem bevontak esetében 0 kimeneti értéket eredményez.

CD

AB

00

0

00

01

1

11

0

A

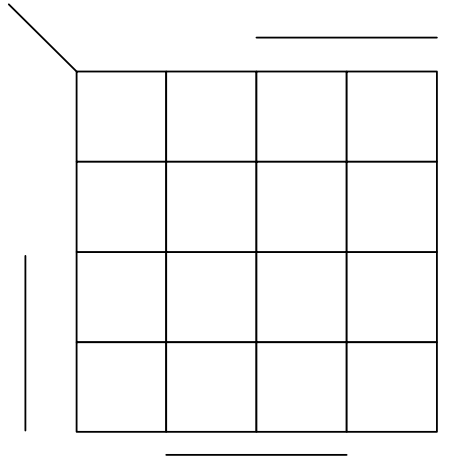
10

1

Prímimplikánsok meghatározása

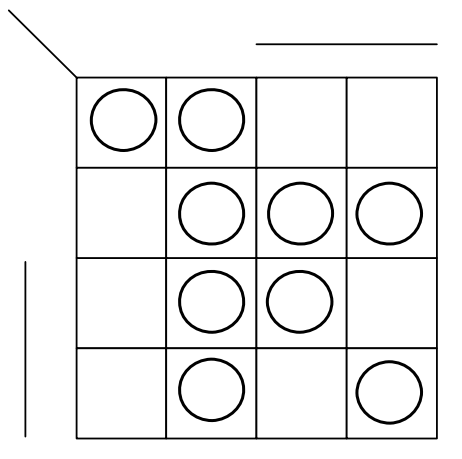
A tömbösítés szabályait figyelembe véve algoritmikusan lehet egyszerűsíteni bármely logikai függvényt. Alapvetően kétfajta megközelítés lehetséges. Az egyikben a legnagyobb hurkokat kell felvenni és utána kisebb hurkokkal kell lefedni a még le nem fedett mintermeket, a másiban pedig a kisebb hurkok felől kell haladni a nagyobbak felé. Algoritmikus és számítógépes megvalósítás szempontból a második módszer használható jobban, ezért a továbbiakban ez lesz részletesen bemutatva.

Az algoritmus egy példán keresztül kerül bemutatásra. A feladat az alábbi Karnaugh táblával adott logikai függvény minimalizálása, ahol az eredményt diszjunktív formában kell megkapni.



5.18 ábra: Példa prímimplikánsok meghatározására: kiinduló Karnaugh tábla

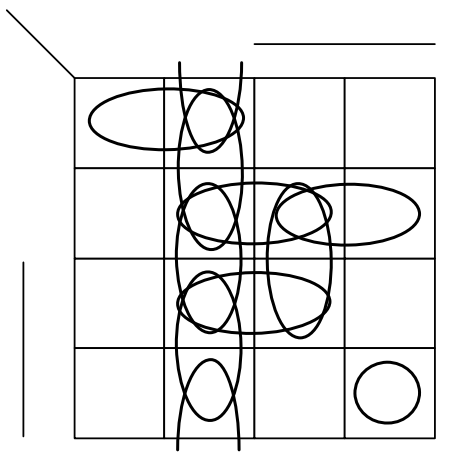
Kezdeként meg kell keresni az összes 1-es hurkot. Ez gyakorlatilag a táblázatban az összes 1-est tartalmazó cella külön-külön való hurkolását jelenti.



5.19 ábra: Példa prímimplikánsok meghatározására: egyes hurkok

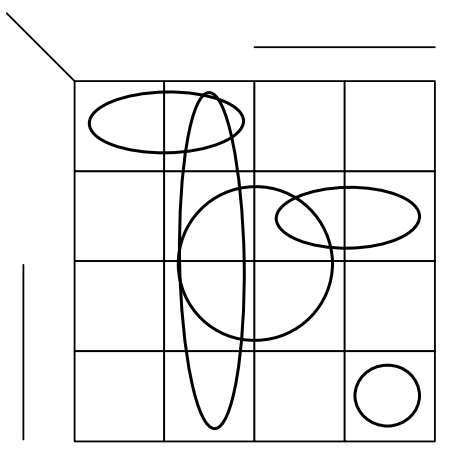
Ezek után következnek egy ciklus. A ciklus i -edik lépésében a 2^i méretű hurkokból elő kell állítani az összes lehetséges 2^{i+1} méretű hurkot. Ha egy hurok összevonásra kerül egy másikkal, akkor az a továbbiakban már nem lesz ábrázolva a Karnaugh táblában, mivel az a hurok nem prímimplikánst reprezentál. A ciklus akkor áll le, ha egy iteráció során nem sikerül új hurkot készíteni.

A ciklus 1. lépésében el kell készíteni az összes kettes hurkot. Például a 0-ás cellához tartozó hurkot össze lehet vonni az 1-eshez tartozóval, az 1-eshez tartozót a 0-áshoz, az 5-öshöz és a 9-eshez tartozóval. Azaz az összes szomszédos hurok estében el kell készíteni egy új kettes hurkot. A 10-es cellához tartozó hurkot kivéve az összes egyes hurok összevonható egy másikkal. Így a ciklus első lépésében 9 darab kettes hurok és 1 darab egyes hurok keletkezik.



5.20 ábra: Példa prímisszimplikánsok meghatározására: kettes hurkok

A 2. lépésben kettes hurkok összevonásával készíthetünk négyes hurkokat. Az (5, 7) hurkot lehet összevonni a (13, 15) hurokkal, valamint az (5, 13) hurkot a (7, 15) hurokkal. Mind a két összevonás az (5, 7, 13, 15) hurkot eredményezi. Ezen kívül összevonható még az (1, 5) a (9, 13) hurokkal, valamint az (1, 9) az (5, 13) hurokkal. Az összevonásban szereplő hurkok eltűnnek (mivel nem prímisszimplikánsok), az összes többi megmarad. Így 2 darab négyes, 2 darab kettes és 1 darab egyes hurok keletkezik.



5.21 ábra: Példa prímisszimplikánsok meghatározására: négyes hurkok

3. lépésben nem lehet négyes hurkokat összevonni nyolcas hurokká, ezért a ciklus befejeződik. A Karnaugh táblában szereplő mind 6 hurok egy-egy prímisszimplikánst jelöl, az egyszerűsített formában ezek szerepelnek. Azaz a függvény egyszerűsített formája $\bar{C}D + BD + \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}\bar{C}\bar{D}$.

A fenti lépéseket követve, a tömbösítés szabályait betartva meg lehet határozni bármely logikai függvény összes prímiimplikánsát. A példában az összes prímiimplikánsra szükség van a legegyszerűbb forma meghatározásához, de könnyen lehet olyan példát mutatni, ahol nincs szükség mindegyikre. A következő fejezetben a prímiimplikánsok kiválasztásának szabályai lesznek bemutatva.

Egyszerűsített alak megadása

A prímiimplikánsok kiválasztásánál figyelembe kell venni az összes a logikai függvényben szereplő mintermet. Azaz úgy kell kiválasztani a prímiimplikánsokat, hogy minden minterm szerepeljen legalább egy prímiimplikánsban. Ez gyakorlatilag megfelel a tömbösítési szabályoknál említett „minden 1-est le kell fedni hurokkal” szabálynak.

Általában vannak olyan prímiimplikánsok, melyeket biztos be kell venni az egyszerűsített alakba. Ehhez meg kell nézni, hogy van-e olyan minterm (maxterm), amely csak egyetlen hurokban szerepel. Ezeket megkülönböztetett mintermnek (maxtermnek) nevezik. Lényeges prímiimplikánsnak nevezik azokat a prímiimplikánsokat, amelyekben legalább egy megkülönböztetett minterm (maxterm) szerepel. A lényeges prímiimplikánsok mindenképpen szerepelnek az egyszerűsített logikai függvényben.

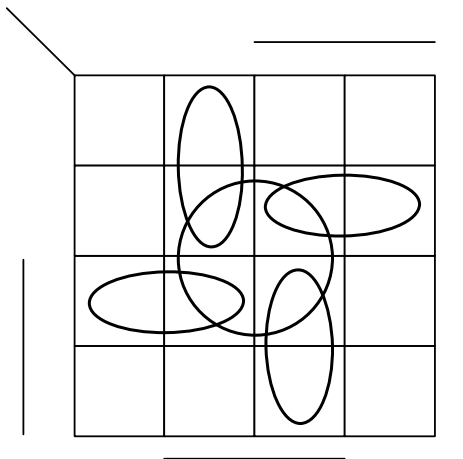
A Karnaugh táblából a minimális számú hurkot kell kiválasztani, amelyek lefedik az összes mintermet. Általában szabad szemmel látható, hogy mely prímiimplikánsok szükségesek a függvény felírásához. A szükséges prímiimplikánsok algoritmikus meghatározásához a prímiimplikáns táblát vagy a segédfüggvényt lehet használni, mely részletesen a számjegyes minimalizálás részben lesz bemutatva.

Az alábbi példában 3 darab kettes hurkot generál a bemutatott módszer. Mind a három kettes hurok egy prímiimplikánst reprezentál, de a (3, 7) hurokhoz tartozó prímiimplikánsra nincs szükség, mivel a 3-as minterm az (1, 3) hurokban, a 7-es minterm a (6, 7) hurokban is szerepel. Így az egyszerűsített függvényalak a következő: $\bar{A}C + AB$.

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

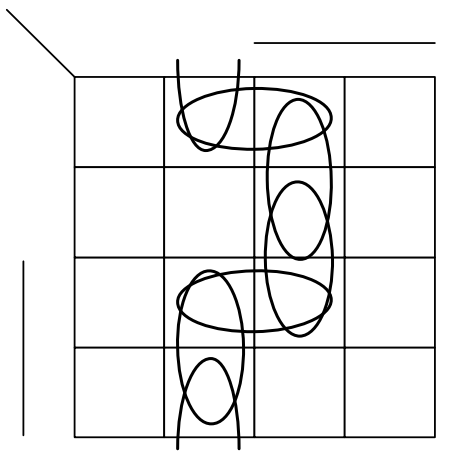
5.22 ábra: Példa redundáns prímiimplikáns esetére

Minimalizálás során olyan eset is előfordulhat, hogy nem a legnagyobb méretű hurkok maradnak meg a kiválasztás után. A következő példában az optimális hurkolás egy négyes és négy kettes hurkot tartalmaz. A Karnaugh táblából jól látszik, hogy az összes kettes hurok lényeges prímiimplikánst jelöl és ezek a kettes hurkok lefedik az összes mintermet, azaz az egyszerűsített függvényalakban a négyes hurok által reprezentált prímiimplikánsra nincs szükség. Így az egyszerűsített függvényalak a következő: $\bar{A}\bar{C}D + \bar{A}BC + AB\bar{C} + ACD$.



5.23 ábra: Példa redundáns prímimplikáns esetére: a legegyszerűbb prímimplikáns redundáns

A következő példa azt az esetet mutatja be, amikor nincs lényeges prímimplikáns, azaz ránézésre nem lehet egyértelműen eldönteni, hogy melyik prímimplikánsokra van szükség a legegyszerűbb alak eléréséhez. Jelen példában két megoldás is létezik: $\bar{A}\bar{B}D + BC\bar{D} + A\bar{C}D$ és $\bar{A}CD + ABD + \bar{B}\bar{C}D$.



5.24 ábra: Példa redundáns prímimplikáns esetére: nem egyértelmű, hogy melyik prímimplikáns redundáns

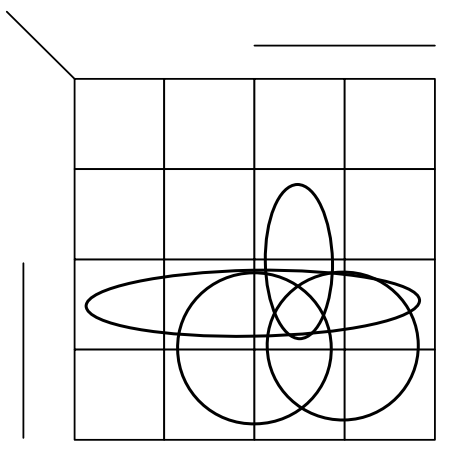
Példa

Készíteni kell egy logikai hálózatot, amelynek négy bemenete (A , B , C és D) van és minden bemenetéhez egy-egy súly tartozik. Az A bemenet súlya legyen 3, a B bemeneté 2, a C és a D bemeneteké pedig 1. A hálózat F kimenetén akkor lesz 1 logikai érték, ha az 1 értékkel rendelkező bemenetek összsúlya nagyobb, mint a 0 bemeneti értékkel rendelkezőké. Azaz ha például az A bemeneten 0 van, B , C és D bemeneteken pedig 1, akkor a kimenet értéke 1, mivel B , C és D összsúlya ($2 + 1 + 1 = 4$) nagyobb, mint A súlya (3). Az igazságtáblázatot ez alapján fel lehet írni.

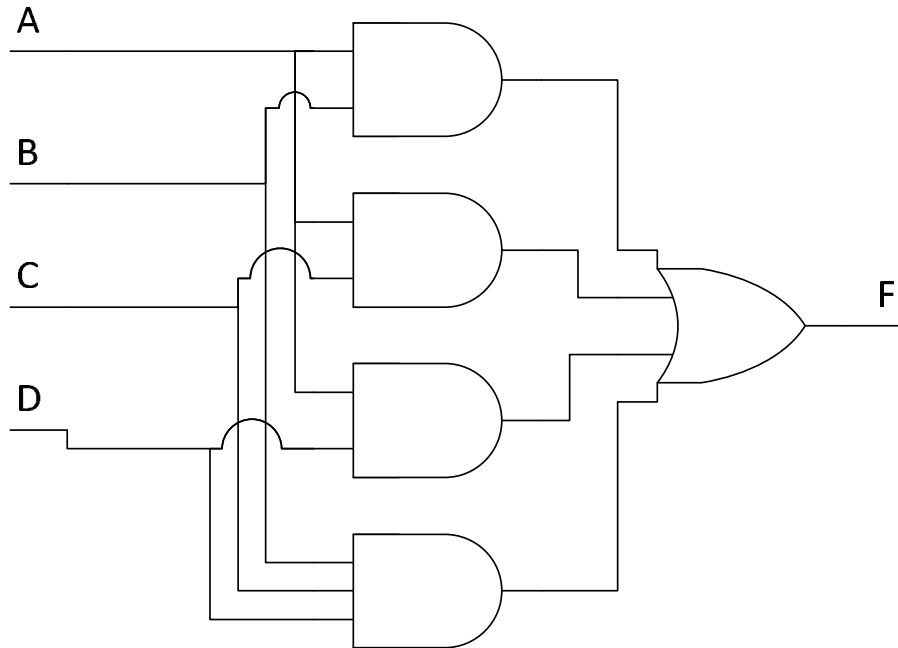
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0

0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

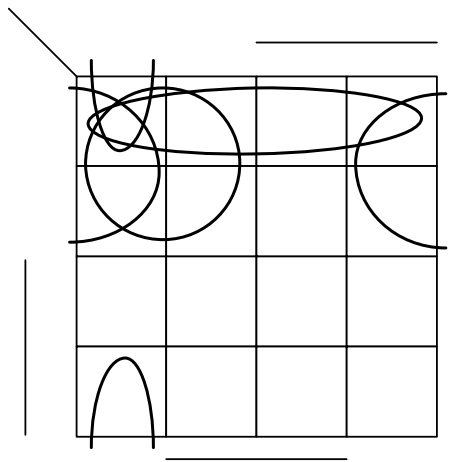
Az igazságtáblázatból fel lehet rajzolni a Karnaugh táblát. Először a diszjunktív alak meghatározása a cél. A primimplikánsok meghatározhatók a Karnaugh tábla segítségével. Ebben az esetben az összes megtalált primimplikáns lényeges primimplikáns.



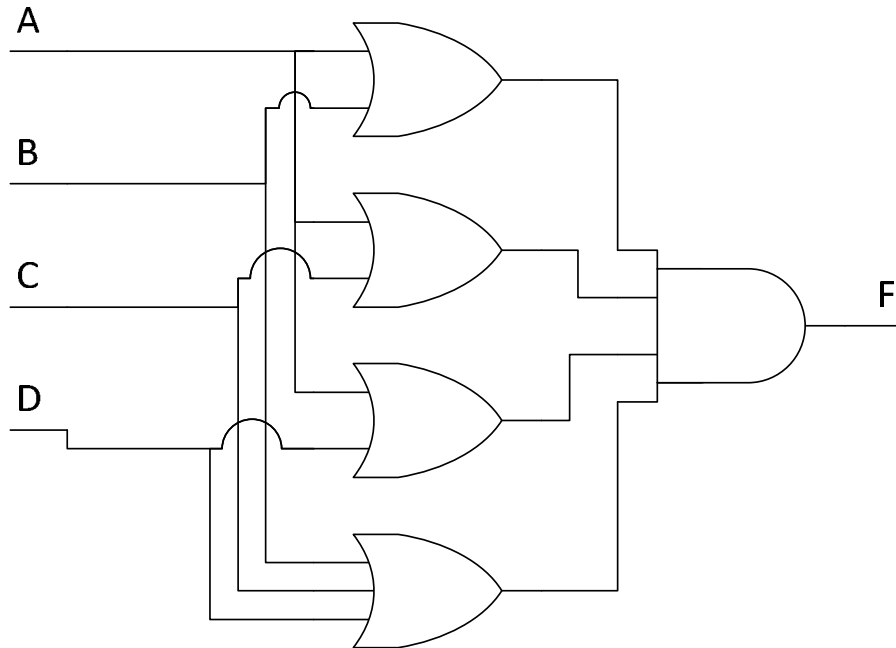
Mivel az összes lényeges primimplikánst tartalmaznia kell az egyszerűsített függvényalaknak, ezért az egyszerűsített függvényalak a következő $F = AB + AC + AD + BCD$, melynek a megvalósítása az alábbi ábrán látható.



Konjunktív alak meghatározása esetén a Karnaugh táblában a 0-kat kell hurkolni a prímmimplikánsok megtalálásához.



Az itt megtalált összes prímmimplikáns is lényeges prímmimplikáns, ezért nem hagyható el a megoldásból. Az eredményül kapott $F = (A + B)(A + C)(A + D)(B + C + D)$ logikai áramköri realizációja alább látható.



A megoldásként kapott diszjunktív és konjunktív alak ugyannyi kapubemenetet tartalmaz, ezért mind a kettő ugyanolyan egyszerűnek számít. Ezek után a rendelkezésre álló konkrét logikai áramkörök tulajdonságai (ár, rendelkezésre állás, méret, ...) határozzák meg, hogy melyiket kell megvalósítani.

Nem teljesen specifikált hálózatok minimalizálása

Nem teljesen specifikált hálózatok esetében a Karnaugh táblában szerepelnek olyan cellák is, melyek sem 0-t sem 1-et nem tartalmaznak, hanem az ezen cellákhoz tartozó bemeneti kombinációra a logikai kapcsolás kimenetén tetszőleges kimenet keletkezhet. Ez tipikusan olyan esetekben fordul elő, amikor egy adott bemeneti kombináció soha nem fordulhat elő. A Karnaugh tábla ilyen bemeneteket reprezentáló celláiban „-” szerepel. A megvalósított logikai áramkör kimenetén természetesen 1 vagy 0 jelenik meg ezekre a bemeneti kombinációkra is, de hogy ezek közül melyik, az egyszerűsítés eredményétől függ. A határozatlan állapotok 1-nek való tekintése növelheti a hurkok méretét, azaz egyszerűbb prímisszimplikánsok találhatók, de ez akár az eredetihez képest további hurkokat is eredményezhet, azaz bonyolíthatja is a függvényt.

Az egyszerűsítés során a határozatlan állapotok 0-nak és 1-nek is tekinthetők. Azaz a hurkok felvétele során le lehet fedni ilyen cellákat is, de a lefedésük nem kötelező. Az előzőekben bemutatott módszer használata esetén a kiinduló lépésben a határozatlan állapotokat is le kell fedni egyes hurkokkal és a ciklusban ezeket a hurkok is felhasználásra kerülnek, azaz az prímisszimplikánsok meghatározása ugyanúgy történik. A különbség az eddigiekhez képest az egyszerűsített alak meghatározásában, azaz a szükséges prímisszimplikánsok kiválasztásában van. Itt a Karnaugh táblából a minimális számú hurkot kell kiválasztani, amelyek lefedik az összes 1-et tartalmazó cellát.

Egy példán keresztül bemutatásra kerül, hogy a határozatlan állapotok 0-nak vagy 1-nek választása mennyiben befolyásolja a megvalósítandó logikai kapcsolási vázlatot. A következő igazságtáblában nyolc darab 1-es és kettő határozatlan állapot szerepel. Karnaugh tábla segítségével meghatározhatók a prímisszimplikánsok.

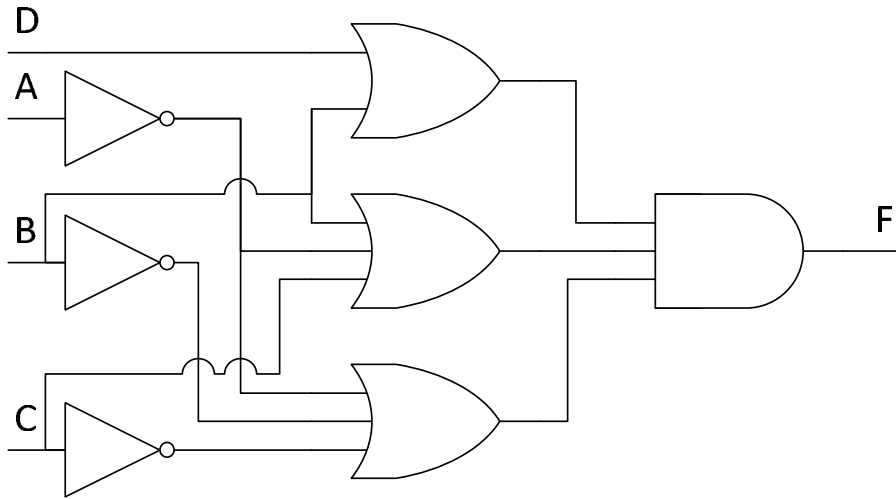
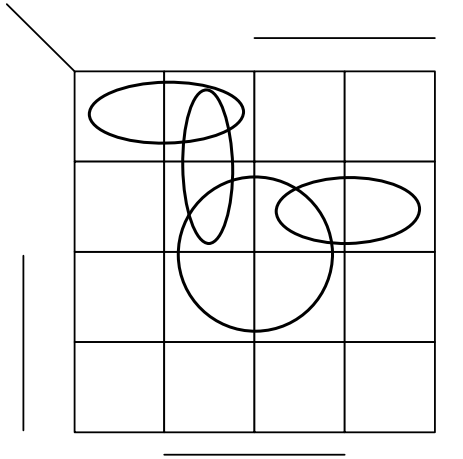
A	B	C	D	F
0	0	0	0	1
0	0	0	1	1

0	0	1	0	0
0	0	1	1	-
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	-
1	1	1	1	1

Először megvizsgálásra kerül, hogy mi történik, ha minden határozatlan állapotot 0. Ekkor az igazságtábla a következő képen módosul.

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

A Karnaugh tábla három darab kettes és egy négyes hurkot eredményez. A Karnaugh táblából a korábbiaknak megfelelően meghatározható az egyszerűsített függvényalak, mely három prímisszűrt tartalmaz és a következő függvényt eredményezi $F = BD + \bar{A}BC + \bar{A}\bar{B}\bar{C}$.

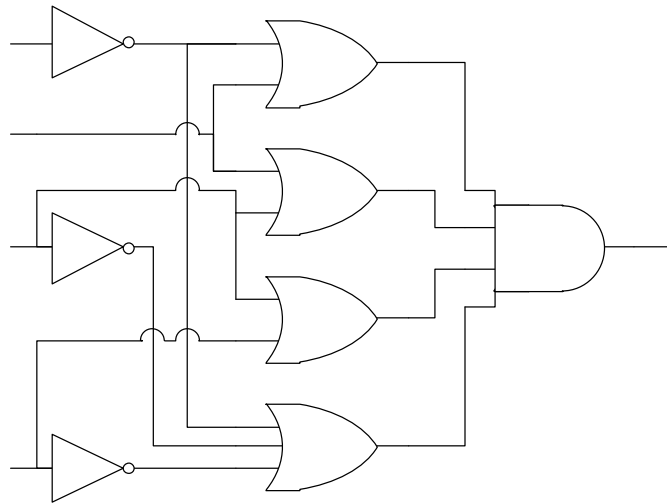
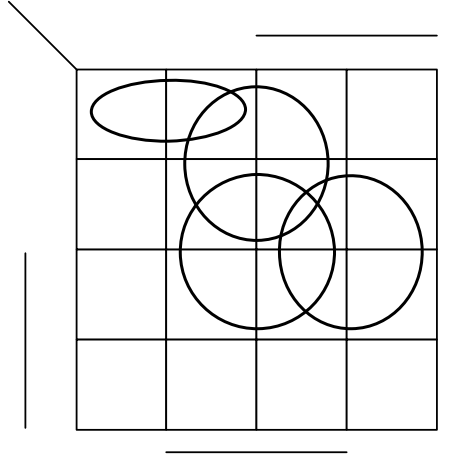


Következő esetben minden határozatlan állapot 1.

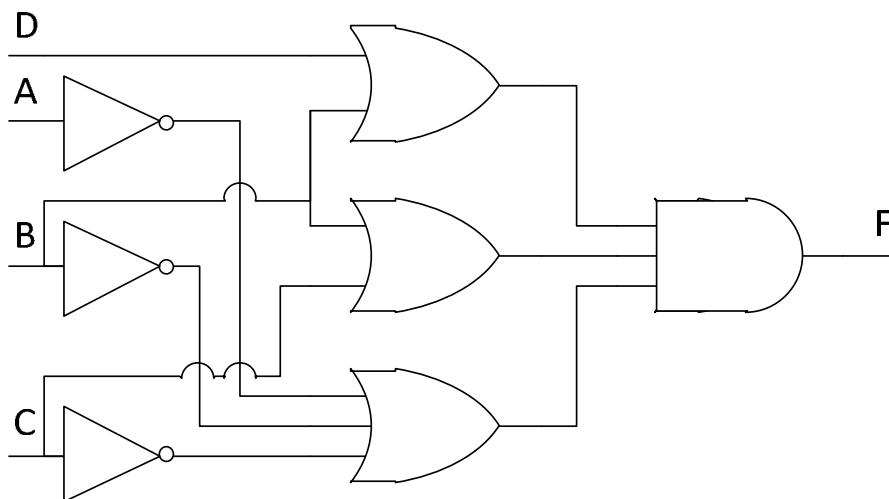
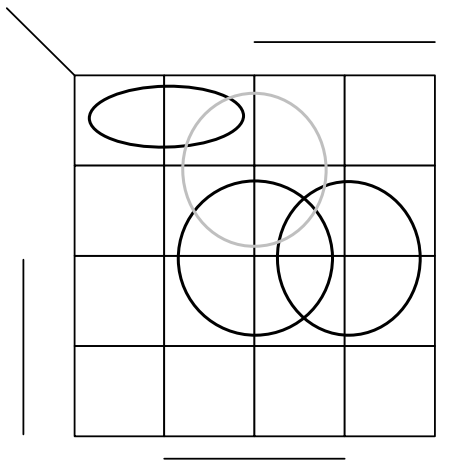
A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1

1	1	1	1	1
---	---	---	---	---

Ebben az esetben egyszerűbb prímiimplikánsokat sikerül elérni, így ez egy darab kettes és három darab négyes hurkot eredményez. Az egyszerűsített függvényalak a következő $F = \bar{A}D + BD + BC + \bar{A}\bar{B}\bar{C}$. A prímiimplikánsok valóban egyszerűbbek, kevesebb mintermet tartalmaznak, de több prímiimplikánusra van szükség a függvény megvalósításához.



Egyik előző eset sem eredményezett optimális megoldást. A fejezet elején leírt módszert alkalmazva ugyanazok a hurkok keletkeznek, abban az esetben, amikor a határozatlan állapotok 1-esek, de most a prímiimplikánsok kiválasztásánál a határozatlan állapotokat ne kelljen feltétlenül lefedni. Így a következő ábrán szürkével jelölt (1, 3, 5, 7) hurkot nem kell az egyszerűsített logikai függvénynek tartalmaznia, mivel annak az 1-es, 5-ös és 7-es cellája le van fedve másik hurokkal is, a negyedik cellája viszont határozatlan állapot. Az egyszerűsített függvényalak az $F = BD + BC + \bar{A}\bar{B}\bar{C}$ függvény, amely kevesebb prímiimplikánst tartalmaz, mint ha minden határozatlan állapot 1-es és egyszerűbb, mint ha 0.



Ez a megvalósítás a következő igazságtáblázatnak felel meg, amely nem mond ellent az eredeti specifikációval.

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

1	1	1	0	1
1	1	1	1	1

A módszer korlátai

A Karnaugh táblák 4 változóig használhatók hatékonyan, 5 változó esetében egy mintermnek (maxtermnek) négy szomszédos minterme (maxterme) van, ami a Karnaugh táblában azt jelentené, hogy minden cellának négy szomszédja van. Ezt két dimenzióban nem lehet ábrázolni, három dimenzióban pedig a papír-ceruza módszer nem hatékony. Ha lehetne háromdimenziós táblázatokat rajzolni, az is legfeljebb 6 változóig oldaná meg a problémát, afelett már a három dimenzió sem lenne elég. A következő fejezetben olyan módszer kerül bemutatásra (számjegyes minimalizálás), amely tetszőleges számú változó esetében használható.

Számjegyes minimalizálás

A számjegyes minimalizálás vagy más néven Quine-McCluskey módszer ugyanazt a minimalizálási alapötletet használja fel, mint a Karnaugh tábla, viszont ebben az esetben nincs korlát a változók számára. A számjegyes minimalizálás a mintermeket egyértelműen azonosító alsó indexet használja fel a számoláshoz. Ez az index egy decimális szám, amely alapján meg kell tudni mondani, hogy két minterm mikor szomszédos egymással.

Két minterm akkor szomszédos, ha csak egy változóban különböznek. Azaz ha bináris számként tekintünk rájuk, akkor csak egy helyiértékben különböznek, tehát egyetlen pozícióban van 1-es helyett 0. Három feltételnek kell teljesülnie a szomszédsághoz. A feltételekben példaként használt két bináris szám különbözik csak az i -edik bitben.

Az első feltétel azt írja elő, hogy a decimális indexek különbsége kettő hatványa kell, hogy legyen. Ha a két bináris számot kivonunk egymásból (a nagyobból a kisebbet), akkor eredményül olyan bináris számot kapunk, amely mindenhol 0, kivéve az i -edik bitet. Ennek decimálisan értéke 2^i . Például az $m_7^3 = ABC$ mintermnek három szomszédja van, az $m_3^3 = \bar{A}BC$, az $m_5^3 = A\bar{B}C$ és az $m_6^3 = AB\bar{C}$. Mind a három mintermmel való különbsége kettő hatványa, konkrétan 4, 2 és 1. Ez a feltétel önmagában nem elegendő, mivel például az $m_2^3 = \bar{A}\bar{B}\bar{C}$ és az $m_1^3 = \bar{A}\bar{B}C$ mintermek decimális indexének különbsége 1, de mégsem szomszédosak.

A második feltétel az indexek bináris súlyának különbséges 1 kell, hogy legyen. A bináris súly azt mondja meg, hogy a számot binárisan felírva hány darab 1-es van benne. Mivel a két bináris szám csak az i -edik bitben különbözik egymástól, ezért amelyikben 1-es van 0 helyett, annak a bináris súlya eggyel nagyobb. Például az $m_7^3 = ABC$ bináris súlya 3, a vele szomszédos $m_3^3 = \bar{A}BC$, $m_5^3 = A\bar{B}C$ és $m_6^3 = AB\bar{C}$ bináris súlya 2. Ez a feltétel sem elegendő önmagában, mivel például az $m_2^3 = \bar{A}\bar{B}\bar{C}$ bináris súlya 1, az $m_3^3 = \bar{A}\bar{B}C$ bináris súlya 2, azaz a bináris súlyuk különbsége 1, de mégsem szomszédosak.

A harmadik feltétel szerint a nagyobb decimális indexnek nagyobb a bináris súlya. Amelyik számban 1-es van 0 helyett, annak decimális értéke 2^i értékkel nagyobb. Ez a feltétel sem elegendő önmagában, mivel például az $m_6^3 = AB\bar{C}$ bináris súlya 2, az $m_0^3 = \bar{A}\bar{B}\bar{C}$ bináris súlya 0, azaz a nagyobb értékűnek nagyobb a bináris súlya, de mégsem szomszédosak.

A három feltétel együtt biztosítja számunkra azt, hogy a két minterm minimális legyen. Azaz összefoglalva két minterm szomszédos, ha a következő feltételek teljesülnek.

- A decimális indexük különbségének értéke 2 hatványa.
- A bináris súlyuk különbsége 1.
- A nagyobb indexű term súlya nagyobb.

A fenti feltételek megfelelően leírják, hogy két minterm mikor szomszédos, azaz mikor lehet őket összevonni. Ez viszont csak az első lépéshez elegendő, mivel az implikánsok nem csak egyetlen mintermet tartalmazhatnak. Az implikánsok összevonásánál a feltétel szerint az implikánsok által

lefedett mintermek páronként szomszédosak. Ez csak abban az esetben fordulhat elő, ha két implikánsnak ugyanazokat a változókat tartalmazza, azaz ugyanazok a változók lettek belőlük kiegészítésre.

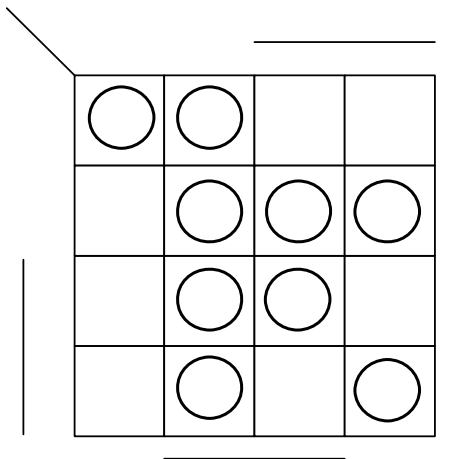
Prímimplikánsok meghatározása

Ezek után gyakorlatilag ugyanaz a módszer használható, mint a Karnaugh tábla esetében. A kevesebb mintermet tartalmazó implikánsokat kell összevonni és nagyobbakat készíteni. Első lépésben az egy mintermet tartalmazó implikánsokat kell összegyűjteni, a Karnaugh táblában ez az egyes hurkokat jelenti. Ezeket egy táblázatba kell rendezni a bináris súlyuk alapján csoportosítva, mivel így a második feltétel ellenőrzésére a későbbiekben nem lesz szükség. A csoportok között aláhúzás szerepel a táblázatban a jobb láthatóság végett. A mintermek összevonása során a szomszédos csoportokban lévő összes mintermet párosítani kell és ellenőrizni, hogy a három korábban felírt feltétel teljesül-e. Amennyiben igen, akkor ez egy új implikánst eredményez, mely egy új táblázatban kerül összegyűjtésre. Ebben a táblázatban az implikánsok mellett fel kell jegyezni, hogy mennyi volt a decimális különbségük. A decimális különbség mutatja, hogy melyik változót lehetett kiegészíteni az összevonással, a decimális különbség 2-es alapú logaritmus jelöli ki az elhagyható változó helyiértékét. Továbbá meg kell jelölni, hogy mely mintermeket lehetett bevonni valamely implikánsba, mivel ezek a mintermek nem lehetnek prímimplikánsok. Ilyen módon el lehet készíteni az összes kettő mintermet tartalmazó implikánst. Érdemes az újonnan kapott implikánsokat is az első mintermjük szerint bináris súlyuk alapján csoportosítani, ezt aláhúzással jelöljük a táblázatban, így csak a szomszédos csoportokban lévő implikánsokat kell megvizsgálni.

A Karnaugh tábla használatához hasonlóan itt is iteratíván addig kell új táblázatokat készíteni az előzőből, amíg keletkezik új implikáns. A ciklus végén azok az implikánsok lesznek a prímimplikánsok, melyeket nem lehetett összevonni másik implikánssal, azaz nem lettek megjelölve az algoritmus során.

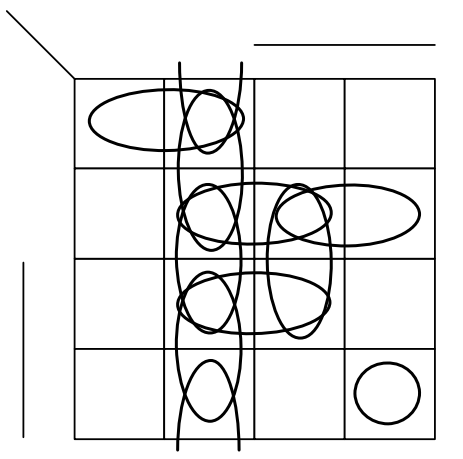
A példában legyen a minimalizálandó függvény ugyanaz, mint a Karnaugh táblás minimalizálás bemutatása során, azaz az $F^{n=4} = \sum_{i=0}^{2^n-1} (0,1,5,6,7,9,10,13,15)$. A könnyebb érthetőség miatt minden lépés után szerepel a Karnaugh tábla is. Első lépés a mintermek csoportosítása a bináris súlyuk szerint.

Minterm	Bináris érték	Bináris súly
0	0000	0
1	0001	1
5	0101	2
6	0110	
9	1001	
10	1010	
7	0111	3
13	1101	
15	1111	4



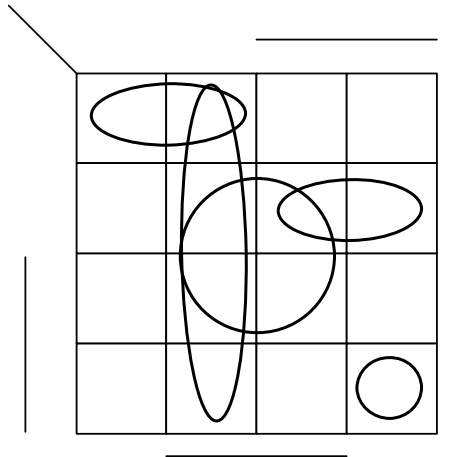
Következő lépésben a szomszédos csoportokban lévő mintermek kell összehasonlítani, és amiket lehet, össze kell vonni új implikánsá. Ahol mind a három, mintermekre vonatkozó szomszédossági feltétel teljesül, azok bekerülnek a következő táblázatba és az összevonható mintermek mellé „+” jel kerül. Például az 5 és 9 mintermre teljesülnek a feltételek, de például a 9 és 7 mintermekre nem teljesül a harmadik tulajdonság, azaz hogy a nagyobb bináris súlyú minterm decimális értéke nagyobb. Az összevonás eredményeképpen ugyanazok az implikánsok adódtak, mint a Karnaugh táblás minimalizálás során a kettős hurkok képzésénél és a 10-es minterm ugyanúgy nem vonható össze semmilyen másik mintermmel, azaz ő maga egy príimplikáns.

Minterm	Bináris érték	Bináris súly	Implikáns	Decimális különbség
0+	0000	0	0, 1	(1)
1+	0001	1	1, 5	(4)
5+	0101	2	1, 9	(8)
6+	0110		5, 7	(2)
9+	1001		5, 13	(8)
10	1010		6, 7	(1)
7+	0111	3	9, 13	(4)
13+	1101		7, 15	(8)
15+	1111	4	13, 15	(2)



Ezek után következik az eredményül kapott implikánsok további összevonása. Ebben az esetben akkor vonhatóak össze az implikánsok, ha páronként teljesülnek a benne lévő mintermekre a szomszédossági feltétel és ebből következően a decimális különbségük megegyezik. Ez alapján össze lehet vonni az 1, 5 implikánst a 9, 13 implikánssal, ami a 1, 5, 9, 13 implikánst eredményezi ugyanúgy, mint az 1, 9 implikáns összevonása az 5, 13 implikánssal. A Karnaugh táblában is minden négyes hurkot két féle képen lehetett előállítani kettes hurkokból. Az összevonás során még egy új implikáns keletkezik, de annak decimális különbsége nem ugyanaz, mint a másiké, ezért ezeket a későbbiekben nem lehet összevonni.

Implikáns	Decimális különbség	Implikáns	Decimális különbség
0, 1	(1)	1, 5, 9, 13	(4, 8)
1, 5+	(4)	5, 7, 13, 15	(2, 8)
1, 9+	(8)		
5, 7+	(2)		
5, 13+	(8)		
6, 7	(1)		
9, 13+	(4)		
7, 15+	(8)		
13, 15+	(2)		



A számjegyes minimalizálás során kapott prímiimplikánsok megegyeznek azokkal a prímiimplikánsokkal, amelyeket a Karnaugh táblás minimalizálás eredményezett.

A következő feladat a decimális indexekből a prímiimplikánsok algebrai alakját meghatározni. Meg kell vizsgálni, hogy mely pozíciókban egyforma az összes minterm algebrai alakja és ez adja meg az prímiimplikáns algebrai alakját. Ez alapján a következő képen kell meghatározni az előző példa prímiimplikánsainak algebrai alakjait.

Decimális alak	Algebrai alakok	Eredő algebrai alak
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}\bar{D}$
0, 1	$\bar{A}\bar{B}\bar{C}\bar{D}$ $\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}\bar{C}$
6, 7	$\bar{A}BC\bar{D}$ $\bar{A}BCD$	$\bar{A}BC$
1, 5, 9, 13	$\bar{A}\bar{B}\bar{C}D$ $\bar{A}B\bar{C}D$ $A\bar{B}\bar{C}D$ $AB\bar{C}D$	$\bar{C}D$

	$\bar{A}\bar{B}\bar{C}D$	
5, 7, 13, 15	$\bar{A}BCD$	BD
	$AB\bar{C}D$	
	$ABCD$	

Egyszerűsített alak megadása

Az egyszerűsített alak meghatározásához elsősorban a prímiimplikáns tábla használható. Ebből kiolvashatók a lényeges prímiimplikánsok és szerencsés esetben ezek az összes mintermet tartalmazzák („lefedik”). Abban az esetben, ha a prímiimplikáns tábla alapján ránézésre nem állapítható meg, hogy mely prímiimplikánsokra van szükség, vagy többváltozós bonyolult függvényt kell minimalizálni, akkor segédfüggvényt használhatunk az eredmény meghatározásához.

A prímiimplikáns táblában minden prímiimplikánsnak tartozik egy sor és minden mintermhez tartozik egy oszlop. Ha egy prímiimplikáns tartalmaz egy mintermet, akkor az adott cellába „*”-ot kell írni. A megkülönböztetett mintermek oszlopaiban egyetlen „x” szerepel, azaz azok a lényeges prímiimplikánsok, amelyben van olyan minterm, melyek oszlopában egyetlen „x” van, ezeket „*”-al megjelöljük.

A $F^{n=4} = \sum_{i=0}^{2^n-1} (0,1,5,7,10,11,13,15)$ függvényhez öt prímiimplikáns tartozik, azaz a prímiimplikáns táblának 5 sora és 8 oszlopa lesz. A fentiek alapján kitölthető a táblázat és látható belőle, hogy 0 mintermet csak a 0, 1 prímiimplikáns fed le, azaz ez lényeges prímiimplikáns. Hasonlóan a 10, 11 és az 5, 7, 13, 15 prímiimplikánsok is lényeges prímiimplikánsok. A példában a lényeges prímiimplikánsok lefedik az összes mintermet, azaz ránézésre látszik a megoldás.

Sor	Prímiimplikáns	0	1	5	7	10	11	13	15
*	0, 1	x	x						
	1, 5		x	x					
*	10, 11					x	x		
	11, 15						x		x
*	5, 7, 13, 15			x	x			x	x

5.25 ábra: Példa prímiimplikáns táblára: a lényeges prímiimplikánsok felfedik az összes mintermet

Olyan eset is előfordulhat, hogy nem a legegyszerűbb prímiimplikánsok maradnak meg a kiválasztás után. A következő példában egy darab négy és négy darab kettő mintermet tartalmazó prímiimplikáns van. A prímiimplikáns táblából jól látszik, hogy az összes kettő mintermet tartalmazó prímiimplikáns lényeges prímiimplikáns jelölésű és ezek lefedik az összes mintermet, azaz az egyszerűsített függvényalakban a négy mintermet tartalmazó prímiimplikánsra nincs szükség.

Sor	Prímiimplikáns	1	5	6	7	11	12	13	15
*	1, 5	x	x						
*	6, 7			x	x				
*	12, 13						x	x	
*	11, 15					x			x
	5, 7, 13, 15		x		x			x	x

5.26 ábra: Példa prímiimplikáns táblára: a lényeges prímiimplikánsok nem fedik le az összes mintermet

A következő példa azt az esetet mutatja be, amikor nincs a rendszerben lényeges prímiimplikáns, azaz ránézésre nem lehet egyértelműen eldönteni, hogy mely prímiimplikánsokra van szükség a

legegyszerűbb alak eléréséhez. Jelen esetben a prímmimplikáns tábla minden oszlopában kettő darab „x” szerepel.

Sor	Prímmimplikáns	1	3	7	9	13	15
1, 3	<i>a</i>	x	x				
1, 9	<i>b</i>	x			x		
3, 7	<i>c</i>		x	x			
7, 15	<i>d</i>			x			x
13, 15	<i>e</i>					x	x
9, 13	<i>f</i>				x	x	

5.27 ábra: Példa prímmimplikáns táblára: nincs lényeges prímmimplikáns

Amikor ránézésre nem állapítható meg a legegyszerűbb alak, vagy amikor sok változó és prímmimplikáns miatt áttekinthetetlen a táblázat, segédfüggvényt kell használni. A segédfüggvény gyakorlatilag logikai függvénye annak a feltételnek, hogy minden mintermet le kell fedni legalább egy prímmimplikánssal. Ennek eléréséhez minden prímmimplikánshoz be kell vezetni egy logikai változót, mely azt reprezentálja, hogy az szerepel-e az egyszerűsített függvényalakban. Ezeket jelöli az előző táblázatban az *a*, *b*, *c*, *d*, *e* és *f*. Azt, hogy egy minterm le van fedve, úgy lehet leírni, hogy az őt lefedni tudó prímmimplikánsokhoz tartozó logikai változókat VAGY kapcsolattal kell összefűzni. Azt, hogy minden minterm le van fedve, úgy lehet leírni, hogy a mintermekhez tartozó logikai kifejezéseket ÉS kapcsolattal kell összekapcsolni. Az eredményül kapott segédfüggvényt *S*-el jelölik.

A prímmimplikáns változóknak egy olyan kombinációját kell megtalálni, hogy a segédfüggvény értéke 1-et vegyen fel. Ehhez át kell alakítani a függvény diszjunktív alakba a szorzások (ÉS kapcsolatok) elvégzésével és meg kell határozni az összes prímmimplikánsát. Diszjunktív alak esetén a segédfüggvény akkor vesz fel 1 értéket, ha van olyan tagja, melynek értéke 1. Azaz ki lehet választani bármelyik tagot a segédfüggvényből, és ha az abban a tagban szereplő változókhoz tartozó prímmimplikánsok segítségével lesz megvalósítva a logikai függvény, akkor minden minterm le lesz fedve. Mivel a legegyszerűbb alak megtalálása a cél, ezért azt a tagot kell kiválasztani, amelyben a legkevesebb prímmimplikáns szerepel. Ha több olyan tag is van, melyek számossága minimális, akkor a választás közöttük tetszőleges.

Az előző példához tartozó segédfüggvény a következő képen néz ki $S = (a + b)(a + c)(c + d)(b + f)(e + f)(d + e)$. Ezt át lehet alakítani diszjunktív alakba a zárójelek felbontásával és egyszerűsíteni az elnyelési tulajdonság felhasználásával $S = (a + bc)(bc + cf + bd + df)(e + df) = (bc + acf + abd + adf)(e + df) = bce + bcdf + acef + acdf + abde + abdf + adef + adf = bce + adf + bcdf + acef + abde$. Az öt kapott prímmimplikáns öt különböző megoldást jelöl, ezek közül az a *bce* és az *adf* a legegyszerűbbek, mivel ezek tartalmazzák a legkevesebb prímmimplikánsot. A *bce* alapján a függvény megvalósításához a *b*, a *c* és az *e* prímmimplikánsokra van szükség, így a függvény egyszerűsített alakja $F = b + c + e = \bar{B}\bar{C}D + \bar{A}CD + ABD$. Az *adf* alapján a függvény megvalósításához az *a*, a *d* és az *f* prímmimplikánsokra van szükség, így a függvény egyszerűsített alakja $F = a + d + f = \bar{A}\bar{B}D + BCD + \bar{A}\bar{C}D$.

Példa

Legyen adott a következő logikai függvény $F^{n=4} = \sum_{i=0}^{2^n-1} 1(4,5,10,11,13,14,15)$. A mintermek alapján fel lehet írni a kiinduló táblázatot.

Minterm	Bináris érték	Bináris súly
4	0100	1
5	0101	2
10	1010	

11	1011	3
13	1101	
14	1110	
15	1111	4

Az első lépésben meg kell határozni a kettő mintermet tartalmazó implikánsokat.

Minterm	Bináris érték	Bináris súly	Implikáns	Decimális különbség
4+	0100	1	4, 5	(1)
5+	0101	2	5, 13	(8)
10+	1010		10, 11	(1)
11+	1011	3	10, 14	(4)
13+	1101		11, 15	(4)
14+	1110		13, 15	(2)
15+	1111	4	14, 15	(1)

Ezek után meghatározhatók a négy mintermet tartalmazó implikánsokat.

Implikáns	Decimális különbség	Implikáns	Decimális különbség
4, 5	(1)	10, 11, 14, 15	(1, 4)
5, 13	(8)		
10, 11+	(1)		
10, 14+	(4)		
11, 15+	(4)		
13, 15	(2)		
14, 15+	(1)		

Mivel csak egyetlen darab négy mintermet tartalmazó implikáns van, ezért azt már nem lehet összevonni másikkal, azaz nincs nyolc mintermet tartalmazó implikáns. A prímiimplikáns tábla segítségével meghatározhatók a lényeges prímiimplikánsok.

Sor	Prímiimplikáns		4	5	10	11	13	14	15
*	4, 5	<i>a</i>	x	x					
	5, 13	<i>b</i>		x			x		
	13, 15	<i>c</i>					x		x
*	10, 11, 14, 15	<i>d</i>			x	x		x	x

A lényeges prímiimplikánsok nem fedik le az összes mintermet, ezért a segédfüggvény segítségével kell meghatározni a szükséges prímiimplikánsokat. $S = a(a + b)dd(b + c)d(c + d) = ad(b + c) = abd + abc$. Azaz a két lényeges prímiimplikáns mellett tetszőleges, hogy a *b* vagy a *c* prímiimplikáns szerepel a megoldásban. Most legyen *b* a kiválasztott prímiimplikáns. Ekkor az eredő függvényhez meg kell határozni a prímiimplikánsok algebrai alakjait.

Decimális alak	Algebrai alakok	Eredő algebrai alak
4, 5	$\bar{A}\bar{B}\bar{C}\bar{D}$ $\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}\bar{C}$
5, 13	$\bar{A}B\bar{C}\bar{D}$ $AB\bar{C}\bar{D}$	$B\bar{C}\bar{D}$
10, 11, 14, 15	$A\bar{B}\bar{C}\bar{D}$ $A\bar{B}CD$ $AB\bar{C}\bar{D}$ $ABCD$	AC

Ebből az eredő függvényalak $F = \bar{A}\bar{B}\bar{C} + B\bar{C}\bar{D} + AC$.

A következő példában legyen $F^{n=5} = \sum_{i=0}^{2^n-1} (0,1,6,7,14,15,17,21,22,23,24,30,31)$ egy ötváltozós logikai függvény. A mintermek alapján fel lehet írni a kiinduló táblázatot.

Minterm	Bináris érték	Bináris súly
0	00000	0
1	00001	1
6	00110	2
17	10001	
24	11000	
7	00111	3
14	01110	
21	10101	
22	10110	
15	01111	4
23	10111	
30	11110	
31	11111	5

Az első lépésben meg kell határozni a kettő mintermet tartalmazó implikánsokat.

Minterm	Bináris érték	Bináris súly	Implikáns	Decimális különbség
0+	00000	0	0,1	(1)
1+	00001	1	1,17	(16)
6+	00110	2	6,7	(1)
17+	10001		6,14	(8)
24	11000		6,22	(16)
7+	00111	3	17,21	(4)
14+	01110		7,15	(8)
21+	10101		7,23	(16)
22+	10110		14,15	(1)
15+	01111	4	14,30	(16)
23+	10111		21,23	(2)
30+	11110		22,23	(1)
31+	11111	5	22,30	(8)
			15,31	(16)
			23,31	(8)
			30,31	(1)

Ezek után meghatározhatók a négy mintermet tartalmazó implikánsokat.

Implikáns	Decimális különbség	Implikáns	Decimális különbség
0, 1	(1)	6, 7, 14, 15	(1, 8)
1, 17	(16)	6, 7, 22, 23	(1, 16)
6, 7+	(1)	6, 14, 22, 30	(8, 16)
6, 14+	(8)	7, 15, 23, 31	(8, 16)
6, 22+	(16)	14, 15, 30, 31	(1, 16)
17, 21	(4)	22, 23, 30, 31	(1, 8)
7, 15+	(8)		
7, 23+	(16)		
14, 15+	(1)		
14, 30+	(16)		
21, 23	(2)		
22, 23+	(1)		
22, 30+	(8)		

15, 31+	(16)
23, 31+	(8)
30, 31+	(1)

Ezek után meghatározhatók a nyolc mintermet tartalmazó implikánsokat.

Implikáns	Decimális különbség	Implikáns	Decimális különbség
6, 7, 14, 15+	(1,8)	6, 7, 14, 15, 22, 23, 30, 31	(1, 8, 16)
6, 7, 22, 23+	(1,16)		
6, 14, 22, 30+	(8,16)		
7, 15, 23, 31+	(8,16)		
14, 15, 30, 31+	(1,16)		
22, 23, 30, 31+	(1,8)		

Mivel csak egyetlen darab nyolc mintermet tartalmazó implikáns van, ezért azt már nem lehet összevonni másikkal, azaz nincs tizenhat mintermet tartalmazó implikáns. A primimplikáns tábla segítségével meghatározhatók a lényeges primimplikánsok.

Sor	Prímimplikáns	0	1	6	7	14	15	17	21	22	23	24	30	31
*	24	<i>a</i>										x		
*	0, 1	<i>b</i>	x	x										
	1, 17	<i>c</i>		x				x						
	17, 21	<i>d</i>						x	x					
	21, 23	<i>e</i>							x		x			
*	6, 7, 14, 15, 22, 23, 30, 31	<i>f</i>			x	x	x	x			x	x	x	x

A lényeges primimplikánsok nem fedik le az összes mintermet, ezért a segédfüggvény segítségével kell meghatározni a szükséges primimplikánsokat. $S = b(b + c)ffff(c + d)(d + e)f(e + f)aff = bf(c + d)(d + e)a = abf(cd + ce + d + de) = abf(ce + d) = abcef + abdf$. A két lehetséges megvalósításból az *abdf* tartalmaz kevesebb primimplikáns, ezért ez az optimális megoldás. Az eredő függvényhez meg kell határozni a primimplikánsok algebrai alakjait.

Decimális alak	Algebrai alakok	Eredő algebrai alak
24	$ABC\bar{D}\bar{E}$	$ABC\bar{D}\bar{E}$
0, 1	$\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$ $\bar{A}\bar{B}\bar{C}\bar{D}E$	$\bar{A}\bar{B}\bar{C}\bar{D}$
17, 21	$A\bar{B}\bar{C}\bar{D}E$ $A\bar{B}C\bar{D}E$	$A\bar{B}\bar{D}E$
6, 7, 14, 15, 22, 23, 30, 31	$\bar{A}\bar{B}CDE$ $\bar{A}BCDE$ $\bar{A}BC\bar{D}E$ $\bar{A}BCDE$ $ABCDE$ $ABC\bar{D}\bar{E}$ $ABCDE$	CD

Ebből az eredő függvényalak $F = ABC\bar{D}\bar{E} + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{D}E + CD$.

Nem teljesen specifikált hálózat minimalizálása

Nem teljesen specifikált esetben is hasonló az optimalizálás, mint a Karnaugh táblánál, azaz a határozatlan állapotokat is fel kell venni implikánsnak a kiinduló táblázatban és használni őket az összevonásoknál. És ugyanúgy, mint a Karnaugh táblánál a primimplikánsok kiválasztásánál nem kell

őket lefedni, a prímiimplikáns táblázatból ki lehet hagyni őket. Így a segédfüggvényben sem szerepelnek a határozatlan mintermekhez tartozó logikai kifejezések.

Legyen adott a következő logikai függvény $F^{n=4} = \sum_{i=0}^{2^n-1} (0,1,3,6,7,8,9) + (2,12)$. A mintermek alapján fel lehet írni a kiinduló táblázatot, ahol a határozatlan állapotok is szerepelnek.

Minterm	Bináris érték	Bináris súly
0	0000	0
1	0001	1
2	0010	
8	1000	
3	0011	2
6	0110	
9	1001	
12	1100	
7	0111	3

Az első lépésben meg kell határozni a kettő mintermet tartalmazó implikánsokat.

Minterm	Bináris érték	Bináris súly	Implikáns	Decimális különbség
0+	0000	0	0, 1	(1)
1+	0001	1	0, 2	(2)
2+	0010		0, 8	(8)
8+	1000		1, 3	(2)
3+	0011	2	1, 9	(8)
6+	0110		2, 3	(1)
9+	1001		2, 6	(4)
12+	1100		8, 9	(1)
7+	0111	3	8, 12	(4)
			3, 7	(4)
			6, 7	(1)

Ezek után meghatározhatók a négy mintermet tartalmazó implikánsokat.

Implikáns	Decimális különbség	Implikáns	Decimális különbség
0, 1+	(1)	0, 1, 2, 3	(1, 2)
0, 2+	(2)	0, 1, 8, 9	(1, 8)
0, 8+	(8)	2, 3, 6, 7	(1, 4)
1, 3+	(2)		
1, 9+	(8)		
2, 3+	(1)		
2, 6+	(4)		
8, 9+	(1)		
8, 12	(4)		
3, 7+	(4)		
6, 7+	(1)		

Mivel mind a három implikánsnál különböznek a decimális különbségek, ezért egyiket sem lehet összevonni másikkal, azaz nincs nyolc mintermet tartalmazó implikáns. A prímiimplikáns tábla segítségével meghatározhatók a lényeges prímiimplikánsok.

Sor	Prímiimplikáns	0	1	3	6	7	8	9
	8, 12	<i>a</i>					x	
	0, 1, 2, 3	<i>b</i>	x	x	x			
*	0, 1, 8, 9	<i>c</i>	x	x			x	x

*	2, 3, 6, 7	d	x	x	x
---	------------	-----	-----	-----	-----

A lényeges prímiimplikánsok nem fedik le az összes mintermet, ezért a segédfüggvény segítségével kell meghatározni a szükséges prímiimplikánsokat. $S = (b + c)(b + c)(b + d)dd(a + c)c = cd$. Azaz az optimális megoldásban a c és a d prímiimplikáns szerepel. Az eredő függvényhez meg kell határozni a prímiimplikánsok algebrai alakjait.

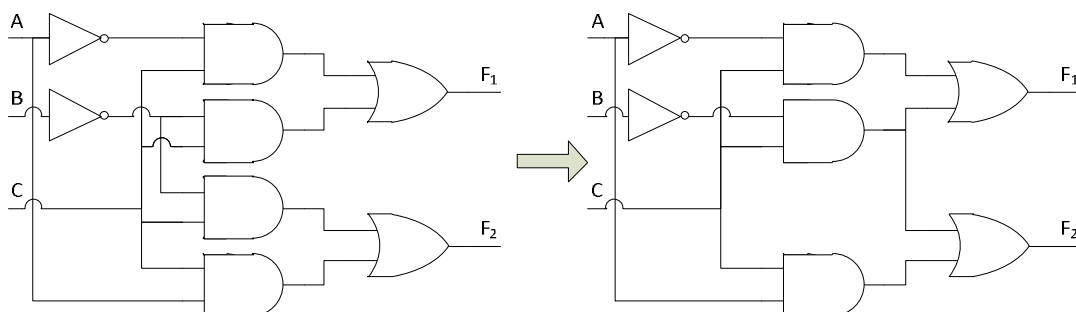
Decimális alak	Algebrai alakok	Eredő algebrai alak
0, 1, 8, 9	$\overline{A}\overline{B}\overline{C}\overline{D}$	$\overline{B}\overline{C}$
	$\overline{A}\overline{B}\overline{C}D$	
	$\overline{A}\overline{B}C\overline{D}$	
	$\overline{A}\overline{B}CD$	
2, 3, 6, 7	$\overline{A}\overline{B}C\overline{D}$	$\overline{A}C$
	$\overline{A}\overline{B}CD$	
	$\overline{A}BC\overline{D}$	
	$\overline{A}BCD$	

Ebből az eredő függvényalak $F = \overline{B}\overline{C} + \overline{A}C$.

6. Többkimenetű logikai függvények egyszerűsítése

Többkimenetű kombinációs hálózatok esetén minden kimenethez külön logikai függvény tartozik. A logikai függvények egyszerűsítését végre lehet hajtani külön-külön egyszerűsítve őket így jutva el a legegyszerűbb diszjunktív vagy konjunktív alakhoz. Mivel mindegyik függvényhez ugyanazok a bemeneti változók tartoznak, ezért további egyszerűsítésre is lehetőség nyílik, a közös részeket nem kell többször megvalósítani.

Legyen egy három bemenetű két kimenetű logikai hálózat, ahol a kimeneti függvények legegyszerűbb diszjunktív alakja a következő $F_1 = \bar{A}C + \bar{B}C$ és $F_2 = AC + \bar{B}C$. A két függvényt külön-külön is meg lehet valósítani, de észrevehető, hogy mind a kettő tartalmazza a $\bar{B}C$ prímmimplikánst, azaz ezt a logikai kapcsolatot elegendő egyszer megvalósítani. Mind a két megvalósítás logikai áramköri realizációja az alábbi ábrán látható. A második megvalósítás természetesen egyszerűbb, azt kell megvalósítani.



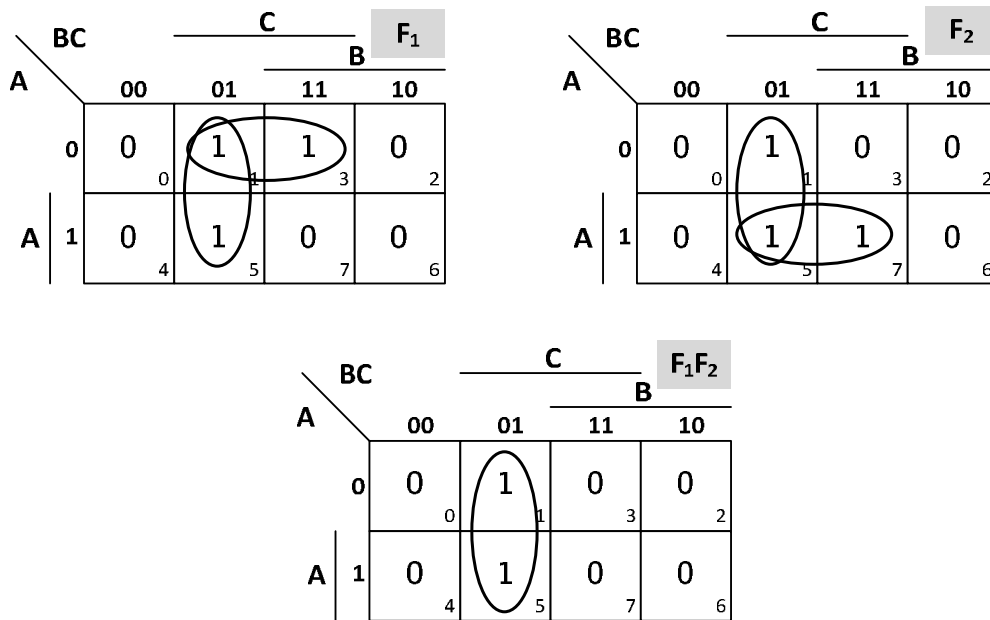
6.1 ábra: Közös prímmimplikáns használata

A cél a továbbiakban a közös prímmimplikánsok megtalálása, hogy azokat felhasználva kevesebb áramköri elemmel legyen megvalósítható a hálózat.

Grafikus minimalizálás

A minimalizálási eljárást módosítani kell úgy, hogy a közös prímmimplikánsokat csak egyszer kelljen megvalósítani. Az egyik legegyszerűbb megoldás a szorzatfüggvény vizsgálata. A szorzatfüggvény ott vesz fel 1 értéket, ahol mind a két függvény 1 értéket vesz fel, azaz a szorzatfüggvény Karnaugh táblájában ott lesz 1, ahol mind a két Karnaugh táblában 1 volt.

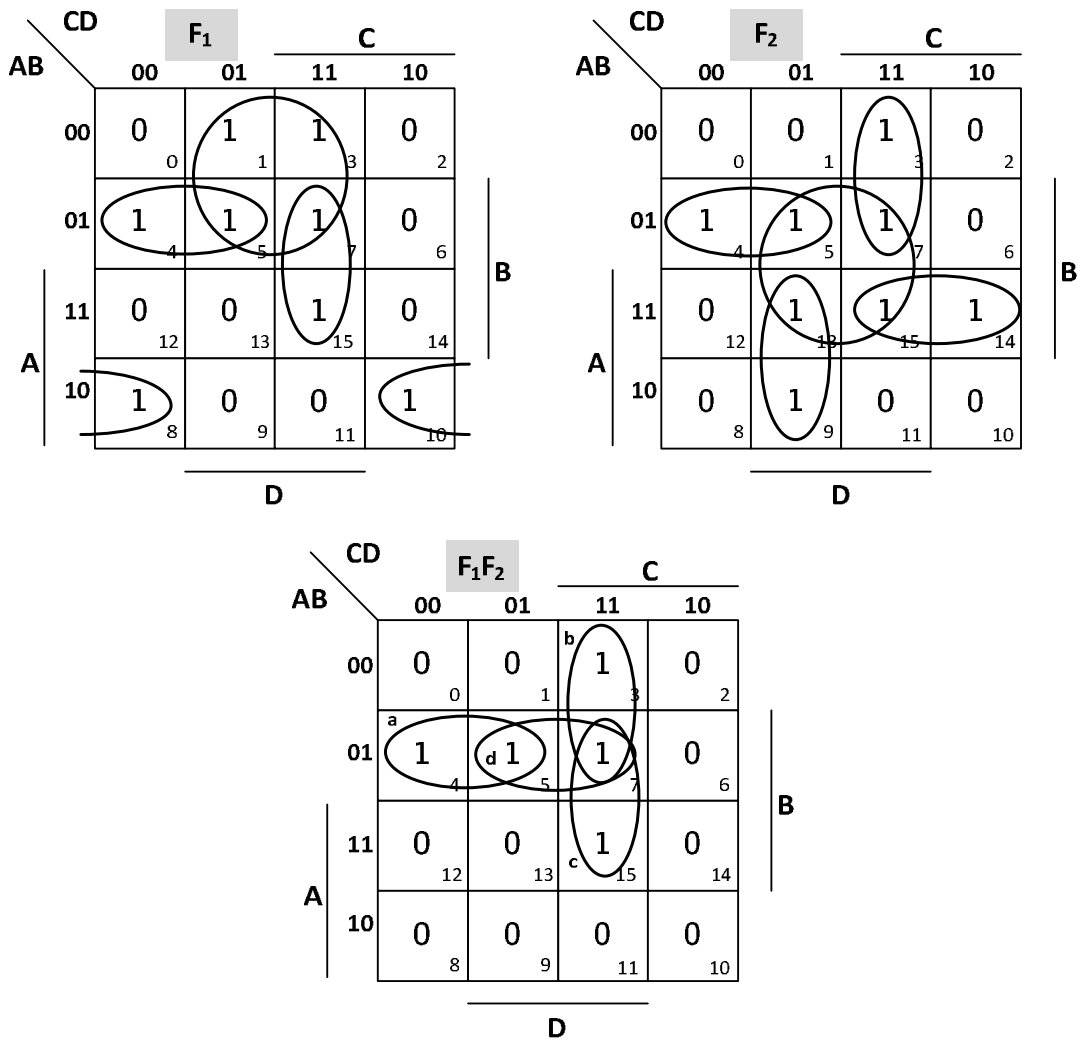
Az előző példát nézve a szorzatfüggvény Karnaugh táblájában egyetlen hurok szerepel, mégpedig a közös $\bar{B}C$ hurok.



6.2 ábra: Függvények és a szorzatfüggvény Karnaugh táblázatai

A szorzatfüggvénynek természetesen lehet olyan prímiimplikánsa is, melyek nem prímiimplikánsa valamely vagy mindegyik eredeti függvényeknek. Az ilyen prímiimplikánsokat nem lehet figyelmen kívül hagyni a szorzatfüggvény optimális lefedéséhez.

A következő példa legyen egy négy bemenetű két kimenetű logikai hálózat, melynek Karnaugh táblái az alábbiakban adóttak. Továbbá adótt a szorzatfüggvény Karnaugh táblája is, ahol az implikánsok betűkkel vannak megjelölve a későbbi könnyebb hivatkozás érdekében.



Megvizsgálva a Karnaugh táblákat megállapítható, hogy a szorzatfüggvény prímiimplikánsai között vannak közös és nem közös prímiimplikánsok. Az **a** prímiimplikáns közös, mivel F_1 -ben és F_2 -ben is szerepel. A **b** prímiimplikáns nem közös, mivel F_2 -ben ugyan prímiimplikáns, de F_1 -ben nem prímiimplikáns, hanem egy másik prímiimplikánsnak csak egy része, azaz tovább egyszerűsíthető implikánsként szerepel. A **c** prímiimplikáns szintén nem közös, mivel F_1 -ben ugyan prímiimplikáns, de F_2 -ben nem prímiimplikáns, hanem egy másik prímiimplikánsnak csak egy része, azaz tovább egyszerűsíthető implikánsként szerepel. A **d** prímiimplikáns nem közös, mivel sem F_1 -ben sem F_2 -ben nem prímiimplikáns, hanem egy másik prímiimplikánsnak csak egy része, azaz tovább egyszerűsíthető implikánsként szerepel.

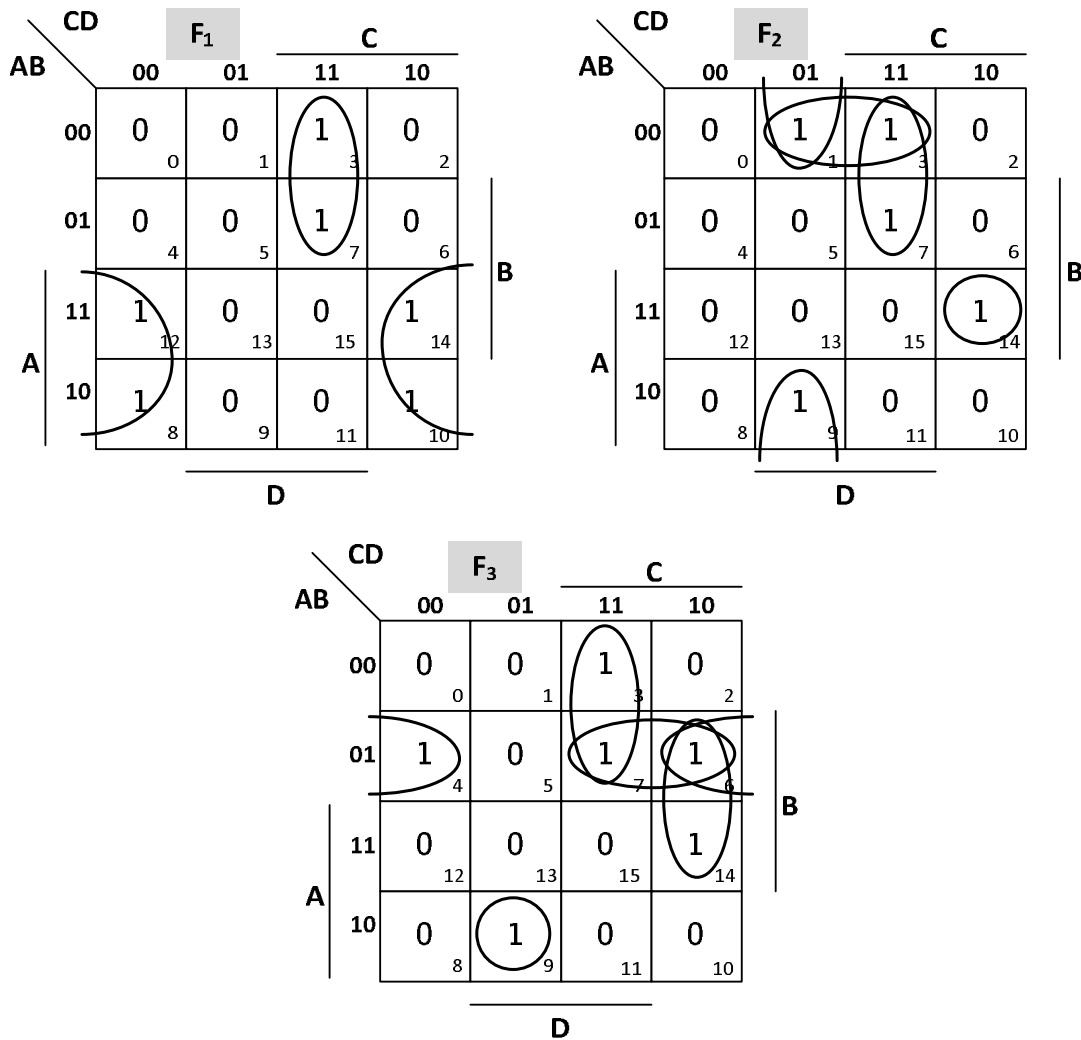
Az egyszerűsítés során a hálózat kimeneti függvényeinek összes lehetséges kombinációjára képezni kell a szorzatfüggvényt, majd ezek után minden függvényhez és szorzatfüggvényhez meg kell határozni a prímiimplikánsokat. A minimalizált függvények felírásához az összes generált prímiimplikáns figyelembe kell venni a függvények és a szorzatfüggvények prímiimplikánsait is.

Mivel a Karnaugh táblákat nehéz lehet áttekinteni, ezért a prímiimplikáns tábla valamint segédfüggvény segítségével kell kiválasztani a szükséges prímiimplikánsokat. A prímiimplikáns tábla tartalmazza az összes megtalált prímiimplikáns a függvények és függvénytársítások szerint csoportosítva. A táblázat kitöltését a szorzatfüggvények prímiimplikánsaival kell kezdeni. Ha egy szorzatfüggvénynek vagy egy függvénynek már van olyan prímiimplikánsa, amely szerepel a táblázatban, akkor azt nem kell még egyszer felvenni. Ha már minden prímiimplikánsa szerepel, akkor

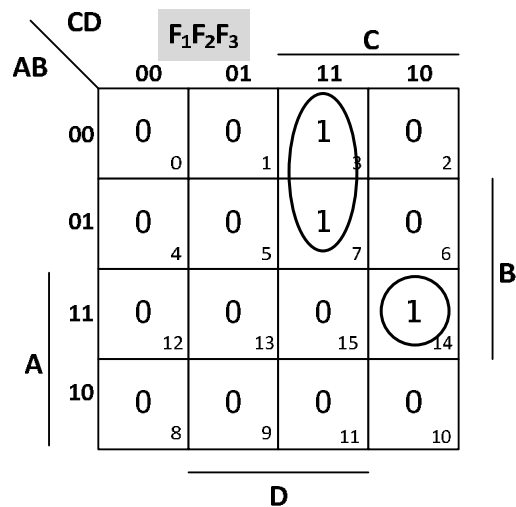
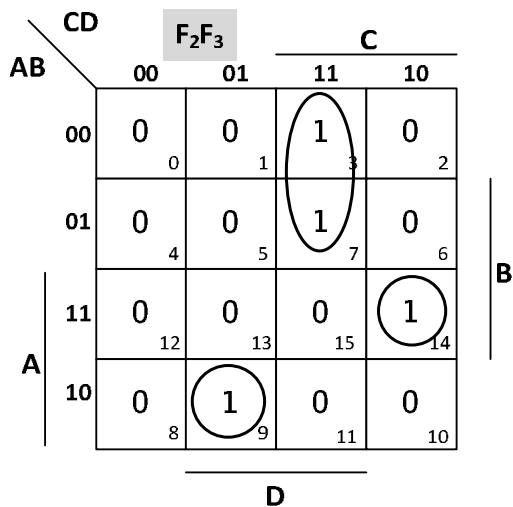
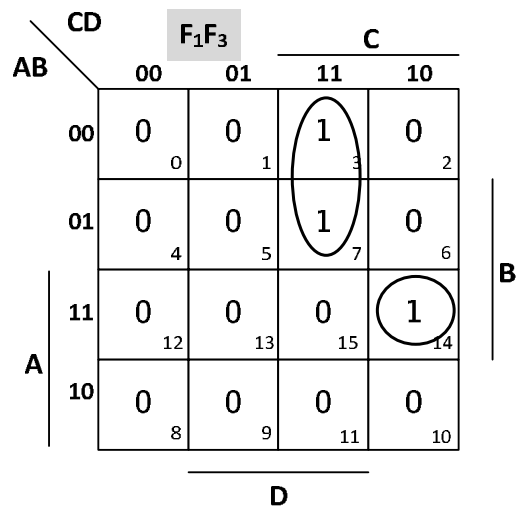
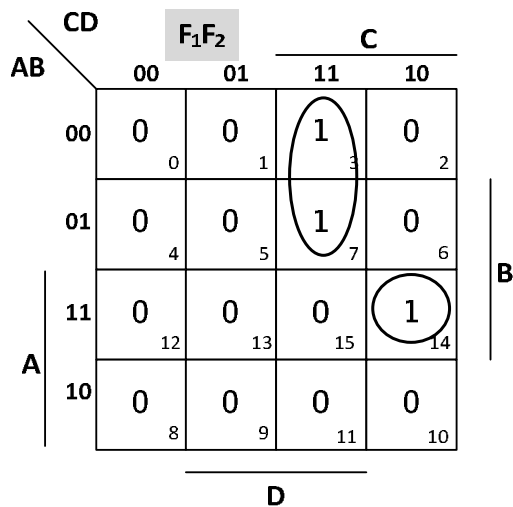
az a függvény vagy szorzatfüggvény nem jelenik meg külön a táblázatban. Így minden prímiimplikáns csak egyszer fog szerepelni a táblázatban.

Amennyiben a szorzatfüggvény alapján több olyan megoldás létezik, melyekben ugyanannyi, minimális számú prímiimplikáns szerepel, akkor ezek közül próbálgatással lehet kiválasztani az optimálist. A próbálgatás során csak a legkevesebb tényezőt tartalmazó szorzatokat kell figyelembe venni. A kiválasztott szorzatban szereplő prímiimplikánsok az elvi logikai rajzban ÉS kapukkal lesznek megvalósítva, majd ezeknek az ÉS kapuknak a kimeneteit egy VAGY kapuk bemeneteire kell vezetni. Mivel azonos számú prímiimplikáns szerepel mindegyik szorzatban, ezért a VAGY szinten ugyanannyi kapubemenet van, tehát az ÉS szinten kell minimalizálni a kapubemenetek számát.

A következő példában a feladat egy négy bemenetű és három kimenetű kombinációs hálózat minimális realizációjának meghatározása diszjunktív formában. Legyen a három kimeneti függvény diszjunktív normál formában adott a következőképpen. $F_1^4 = \sum_{i=0}^{2^n-1} (3,7,8,10,12,14)$, $F_2^4 = \sum_{i=0}^{2^n-1} (1,3,7,9,14)$ és $F_3^4 = \sum_{i=0}^{2^n-1} (3,4,6,7,9,14)$. Első lépésben meg kell határozni a kimeneti függvények prímiimplikánsait Karnaugh táblák segítségével, mely alapján $F_1 = A\bar{D} + \bar{A}CD$, $F_2 = \bar{A}\bar{B}D + \bar{A}CD + \bar{B}CD + ABC\bar{D}$ és $F_3 = \bar{A}CD + \bar{A}BC + \bar{A}B\bar{D} + BCD + A\bar{B}C\bar{D}$.



Második lépésben az összes lehetséges szorzatfüggvény prímiimplikánsainak meghatározása következik, mely alapján $F_1F_2 = \bar{A}CD + ABC\bar{D}$, $F_1F_3 = \bar{A}CD + ABC\bar{D}$, $F_2F_3 = \bar{A}CD + ABC\bar{D} + A\bar{B}C\bar{D}$ és $F_1F_2F_3 = \bar{A}CD + ABC\bar{D}$.



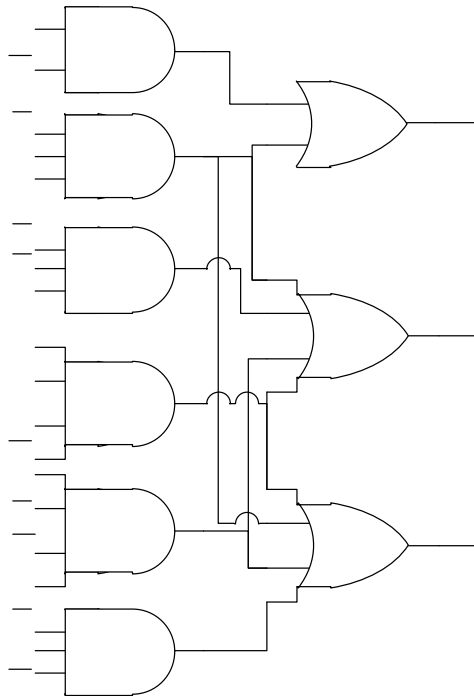
A generált prímisszimplikánsok alapján felírható a prímisszimplikáns tábla. Például az $\bar{A}CD$ prímisszimplikáns szerepel az $F_1F_2F_3$ szorzatfüggvényben így bekerül a táblázatban, de hiába szerepel ugyanez a prímisszimplikáns az összes többi függvényben és szorzatfüggvényben, nem kerül be többször a táblázatba. Valamint például az F_1F_2 szorzatfüggvény nem szerepel a táblázatban, mivel az összes prímisszimplikánsa már korábban bekerült.

Sor	Prímisszimplikáns	a	F_1					F_2					F_3						
			3	7	8	10	12	14	1	3	7	9	14	3	4	6	7	9	14
$F_1F_2F_3$	* $\bar{A}CD$	a	x	x					x	x			x				x		
	* $ABC\bar{D}$	b					x				x								x
F_2F_3	* $\bar{A}\bar{B}\bar{C}D$	c								x								x	
F_1	* $A\bar{D}$	d			x	x	x	x											
F_2	$\bar{A}\bar{B}D$	e							x	x									
	$\bar{B}\bar{C}D$	f							x			x							
F_3	$\bar{A}BC$	g													x	x			

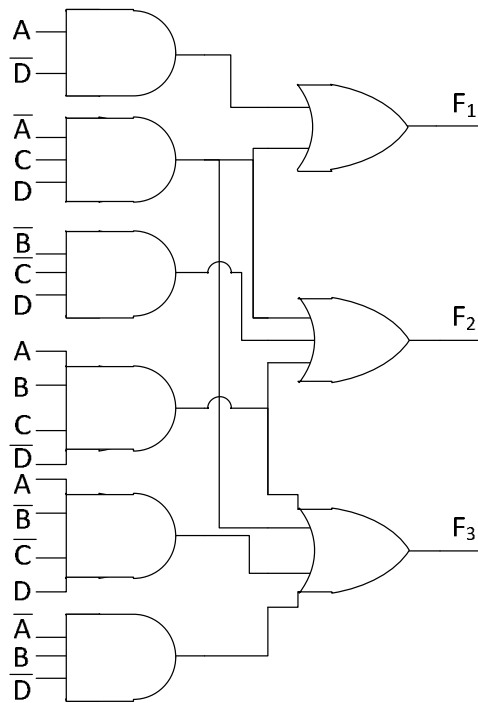
*	$\bar{A}B\bar{D}$	h			x	x	
	$BC\bar{D}$	i				x	x

A táblázatból leolvashatóak a lényeges prímiplikánsok, de ezek nem fedik le az összes mintermet, ezért mindenképpen segédfüggvényre van szükség. Mind a három függvényhez fel kell írni a segédfüggvényt, ahol $S_1 = aadd(b + d)$, $S_2 = (e + f)(a + e)a(c + f)b$ és $S_3 = ah(g + h + i)(a + g)c(b + i)$. Az eredő segédfüggvény a segédfüggvények ÉS kapcsolata, azaz $S = S_1S_2S_3 = aadd(b + d)(e + f)(a + e)a(c + f)bah(g + h + i)(a + g)c(b + i)$. Az elnyelési tulajdonság segítségével a függvény egyszerűsíthető $S = abcdh(e + f) = abcdeh + abcdfh$. A segédfüggvény két lehetséges megvalósítást eredményez, ahol mind a kettőben ugyanannyi prímiplikáns szerepel. A legegyszerűbb hálózat eléréséhez próbálgatásra van szükség, azaz mind a két lehetséges megoldást elemezni kell.

Az $abcdeh$ megvalósítása esetén mind a három függvény megvalósításához csak az adott prímiplikánsokat szabad felhasználni. Mind a három függvényhez fel kell írni egy segédfüggvényt és megvizsgálni, hogy melyik függvényhez melyik prímiplikánsok kellenek, de a nem felhasználható prímiplikánsok helyett 0 értékkel kell számolni. Így $S_1 = aadd(b + d) = ad$, $S_2 = (e + 0)(a + e)a(c + 0)b = e(a + e)acb = abce$ és $S_3 = ah(0 + h + 0)(a + 0)c(b + 0) = ahacab = abch$. Ebből meghatározhatók a kimenetek függvényei, $F_1 = \bar{A}CD + A\bar{D}$, $F_2 = \bar{A}CD + ABC\bar{D} + A\bar{B}\bar{C}D + \bar{A}\bar{B}D$ és $F_3 = \bar{A}CD + ABC\bar{D} + A\bar{B}\bar{C}D + \bar{A}\bar{B}D$, melynek logikai áramköri realizációja lejjebb látható és a bementi negálókat nem számolva összesen 29 kapubemenetet tartalmaz.



Az $abcdfh$ megvalósítása esetén mind a három függvény megvalósításához csak az adott prímiplikánsokat szabad felhasználni. Mind a három függvényhez fel kell írni egy segédfüggvényt és megvizsgálni, hogy melyik függvényhez melyik prímiplikánsok kellenek, de a nem felhasználható prímiplikánsok helyett 0 értékkel kell számolni. Így $S_1 = aadd(b + d) = ad$, $S_2 = (0 + f)(a + 0)a(c + f)b = faa(c + f)b = abf$ és $S_3 = ah(0 + h + 0)(a + 0)c(b + 0) = ahacab = abch$. Ebből meghatározhatók a kimenetek függvényei, $F_1 = \bar{A}CD + A\bar{D}$, $F_2 = \bar{A}CD + ABC\bar{D} + \bar{B}\bar{C}D$ és $F_3 = \bar{A}CD + ABC\bar{D} + A\bar{B}\bar{C}D + \bar{A}\bar{B}D$, melynek logikai áramköri realizációja lejjebb látható és a bementi negálókat nem számolva összesen 28 kapubemenetet tartalmaz.



A két kapcsolás csak az F_2 függvény megvalósításában különbözik. A második megvalósításban egyel kevesebb prímisszorzó kell a függvény megvalósításához. Az első megvalósításhoz képest egy három bemenetű ÉS kapu helyett egy másik három bemenetű kell, ez az ÉS kapuk szintjén nem jelent egyszerűsítést, de a VAGY kapuk szintjén az F_2 megvalósításához négy bemenetű VAGY kapu helyett elég egy három bemenetű is, azaz egy kapubemenetet lehet megtakarítani.

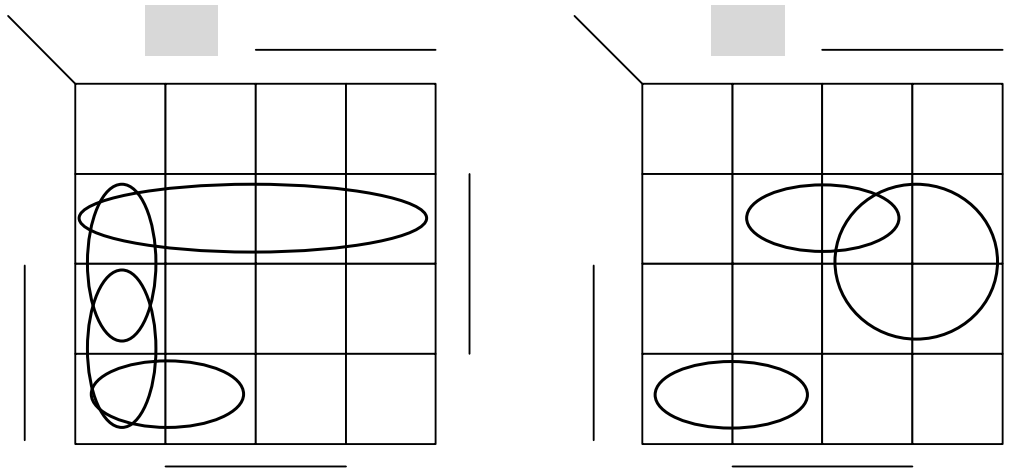
Nem teljesen specifikált hálózatok minimalizálása

Több kimenttel rendelkező, nem teljesen specifikált hálózatok esetén ugyanúgy kell eljárni, mint egy kimenttel rendelkező hálózatoknál. Azaz a prímisszorzók generálásához fel kell használni a határozatlan állapotokat, de az egyszerűsített alak megadásánál ezeket nem kell figyelembe venni. Mivel többkimenetű hálózatok esetén az egyszerűsített alak meghatározásához prímisszorzó táblát kell használni, ezért a határozatlan állapotokat nem kell felvenni a táblázatba, ugyanúgy, mint az egy kimenttel rendelkező, nem teljesen specifikált hálózatok esetében. Természetesen ebben az esetben is szükség van az összes lehetséges szorzatfüggvény előállítására és a hozzájuk tartozó prímisszorzók generálására.

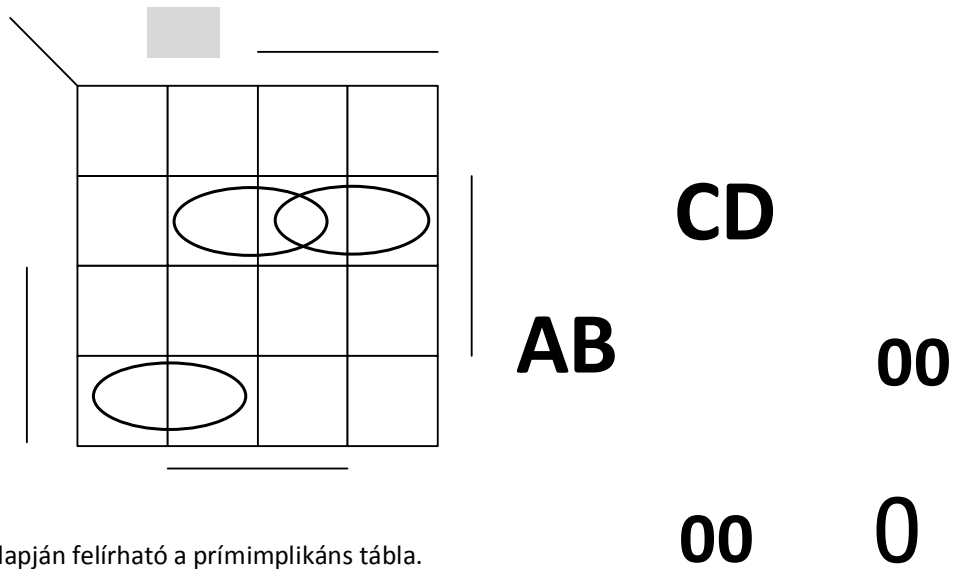
Egy fontos tisztázandó kérdés marad csupán a szorzatfüggvények előállításával kapcsolatban, a határozatlan állapotok ÉS kapcsolatának meghatározása. A határozatlan állapotot lehet egy logikai változónak tekinteni, amely bármely logikai értéket felvehet, és így egyértelmű lesz a szorzás eredménye. Az ÉS kapcsolat azonosságai között szerepel, hogy $A \cdot 0 = 0$, azaz ha 0 logikai értéket bármivel összefűzünk ÉS kapcsolat segítségével, akkor az 0 értéket eredményez, azaz a határozatlan állapot ÉS kapcsolat 0-val 0-t eredményez. Valamint az $A \cdot 1 = A$ azonosságból következik, hogy az 1 értékkel való ÉS kapcsolat eredménye csak a logikai változó értékétől függ, azaz a határozatlan állapot ÉS kapcsolata 1-el határozatlan állapot marad. Végül a határozatlan állapot ÉS kapcsolata egy másik határozatlan állapottal határozatlan állapotot eredményez.

A példa egy négy bemenetű és kettő kimenetű nem teljesen specifikált hálózat minimalizálást mutatja be. Legyen a három logikai függvény a következő $F_1^{n=4} = \sum_{i=0}^{2^n-1} (4,5,7,8,12) + (6,9)$ és $F_2^{n=4} = \sum_{i=0}^{2^n-1} (5,6,9,14,15) + (7,8)$. Ez alapján fel kell rajzolni a Karnaugh táblákat és megkeresni az

összes prímmimplikánst Karnaugh táblák segítségével, mely alapján $F_1 = \bar{A}B + B\bar{C}\bar{D} + A\bar{C}\bar{D} + A\bar{B}\bar{C}$,
 $F_2 = BC + \bar{A}BD + A\bar{B}\bar{C}$.



Második lépésben az összes lehetséges szorzatfüggvény prímmimplikánsainak meghatározása következik, mely alapján $F_1F_2 = \bar{A}BC + \bar{A}BD + A\bar{B}\bar{C}$.



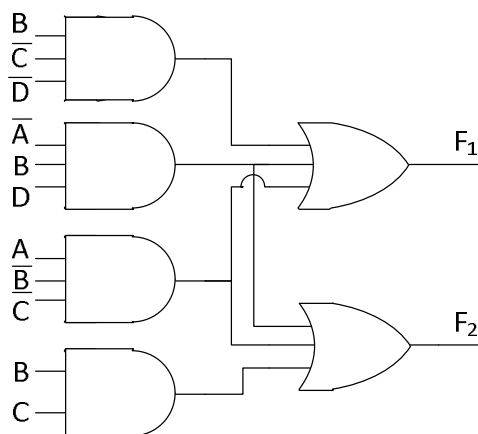
A generált prímmimplikánsok alapján felírható a prímmimplikáns tábla.

Sor	Prímmimplikáns	F_1				F_2					
		4	5	7	8	12	5	6	9	14	15
*	$\bar{A}BC$	a		x				x			
F_1F_2	$\bar{A}BD$	b	x	x			x				
	$A\bar{B}\bar{C}$	c			x			x			
F_1	$\bar{A}B$	d	x	x	x						
	$B\bar{C}\bar{D}$	e	x							x	
	$A\bar{C}\bar{D}$	f			x	x					

F_2	*	BC	g		x	x	x
-------	---	------	-----	--	-----	-----	-----

A táblázatból leolvashatóak a lényeges prímiimplikánsok, de ezek nem fedik le az összes mintermet, ezért mindenképpen segédfüggvényre van szükség. Mind a kettő függvényhez fel kell írni a segédfüggvényt, ahol $S_1 = (d + e)(b + d)(a + b + d)(c + f)(e + f)$ és $S_2 = b(a + g)cg$. Az eredő segédfüggvény a segédfüggvények ÉS kapcsolata, azaz $S = S_1 S_2 = (d + e)(b + d)(a + b + d)(c + f)(e + f)b(a + g)cg$. Az elnyelési tulajdonság és a zárójelek felbontásának segítségével a függvény egyszerűsíthető $S = bcg(d + e)(e + f) = bcg(e + df) = bcge + bcgdf$. Ez két lehetséges megvalósítást eredményez, ahol a $bcge$ megvalósításban kevesebb prímiimplikáns szerepel, ezért a legegyszerűbb hálózat eléréséhez ezt kell használni.

Mind a kettő függvény megvalósításához csak az adott prímiimplikánsokat szabad felhasználni. Mind a kettő függvényhez fel kell írni egy segédfüggvény és megvizsgálni, hogy melyik függvényhez melyik prímiimplikánsok kellenek, de a nem felhasználható prímiimplikánsok helyett 0 értékkel kell számolni. Így $S_1 = (0 + e)(b + 0)(0 + b + 0)(c + 0)(e + 0) = bce$ és $S_2 = b(0 + g)cg = bcg$. Ebből meghatározhatók a kimenetek függvényei, $F_1 = \bar{A}BD + A\bar{B}\bar{C} + B\bar{C}\bar{D}$ és $F_2 = \bar{A}BD + A\bar{B}\bar{C} + BC$, melynek logikai áramköri realizációja lejjebb látható.



Számjegyes minimalizálás

Quine-McCluskey módszer esetén fel kell készíteni az algoritmust arra, hogy minden mintermről, implikánsról és prímiimplikánsról tudni lehessen, hogy melyik kimeneti függvényben fordul elő. Ennek eléréséhez annyi jelzőbitet kell rendelni minden mintermhez, ahány kimenetű a hálózat. A bit értéke 1, ha az adott minterm előfordul az adott kimeneti függvényben. Ezeket a biteket a táblázatban tároljuk a mintermek mellett.

Az egy kimenettel rendelkező hálózatok esetében három feltétel volt a mintermek szomszédosságára, amelyek továbbra is igazak. Azaz két minterm szomszédos, ha a következő feltételek teljesülnek.

- A decimális indexük különbségének értéke 2 hatványa.
- A bináris súlyuk különbsége 1.
- A nagyobb indexű term súlya nagyobb.

Több kimenetű hálózatok esetén arra is figyelni kell, hogy csak akkor vonható össze két minterm (implikáns), ha van olyan jelzőbit, amelyik mind a kettőnél 1. Az eredő implikáns jelzőbitjei ott lesznek 1-ek, ahol mind a két minterm (implikáns) jelzőbitje 1 volt, mindenhol máshol 0 érték szerepel. Gyakorlatilag egy ÉS logikai műveletet (szorzást) kell elvégezni a jelzőbiteken. Ha az összevont mintermek (implikánsok) jelzőbitjei megegyeznek, akkor meg lehet őket jelölni, mint

összevont tagokat, de ha valahol különböznek, akkor nem, mert az valamely kimeneti függvény prímiplikánsa lehet.

A példa legyen ugyanaz, mint a teljesen specifikált hálózat Karnaugh táblás egyszerűsítés esetében, azaz egy négy bemenetű és három kimenetű kombinációs hálózat minimális realizációjának kell meghatározni diszjunktív formában. A három kimeneti függvény diszjunktív normál formában adott a következő képen. $F_1^4 = \sum_{i=0}^{2^n-1}(3,7,8,10,12,14)$, $F_2^4 = \sum_{i=0}^{2^n-1}(1,3,7,9,14)$ és $F_3^4 = \sum_{i=0}^{2^n-1}(3,4,6,7,9,14)$. Ebből fel lehet írni a kiindulási táblázatot.

Minterm	Bináris érték	Bináris súly	F_1	F_2	F_3
1	0001	1	0	1	0
4	0100		0	0	1
8	1000		1	0	0
3	0011	2	1	1	1
6	0110		0	0	1
9	1001		0	1	1
10	1010		1	0	0
12	1100		1	0	0
7	0111	3	1	1	1
14	1110		1	1	1

A következő lépésben elkészíthetők a kettő mintermet tartalmazó implikánsok. A 8 és 9 mintermekre teljesül ugyan a szomszédsági feltétel, de nem lehet őket összevonni mivel nincsenek benn ugyanabban a függvényben. Az 1 és 9 mintermek szomszédosak és van közös jelzőbitjük, ezért össze lehet őket vonni, de mivel nem egyezik meg az összes jelző bit, ezért az eredő implikánsban csak az F_2 jelzőbitje lesz 1 és egyik minterm sem lesz megjelölve, mint egyszerűsíthető minterm.

Minterm	Bináris érték	Bináris súly	F_1	F_2	F_3	Implikáns	Decimális különbség	F_1	F_2	F_3
1	0001	1	0	1	0	1, 3	(2)	0	1	0
4+	0100		0	0	1	1, 9	(8)	0	1	0
8+	1000		1	0	0	4, 6	(2)	0	0	1
3+	0011	2	1	1	1	8, 10	(2)	1	0	0
6+	0110		0	0	1	8, 12	(4)	1	0	0
9	1001		0	1	1	3, 7	(4)	1	1	1
10+	1010		1	0	0	6, 7	(1)	0	0	1
12+	1100		1	0	0	6, 14	(8)	0	0	1
7+	0111	3	1	1	1	10, 14	(4)	1	0	0
14	1110		1	1	1	12, 14	(2)	1	0	0

A négy mintermet tartalmazó implikánsok generálása hasonló módon történik, mint a kettő mintermet tartalmazóké. Egyetlen négy mintermet tartalmazó implikáns van, ami természetesen két féle módon is generálható ugyanúgy, mint egy kimenettel rendelkező hálózatok esetében.

Implikáns	Decimális különbség	F_1	F_2	F_3	Implikáns	Decimális különbség	F_1	F_2	F_3
1, 3	(2)	0	1	0	8, 10, 12, 14	(2, 4)	1	0	0
1, 9	(8)	0	1	0					
4, 6	(2)	0	0	1					
8, 10+	(2)	1	0	0					
8, 12+	(4)	1	0	0					
3, 7	(4)	1	1	1					
6, 7	(1)	0	0	1					
6, 14	(8)	0	0	1					
10, 14+	(4)	1	0	0					

12, 14+ (2) 1 0 0

Mivel egyetlen négy mintermet tartalmazó implikáns van, ezért további egyszerűsítésre nincs mód. A grafikus minimalizáláshoz hasonlóan fel kell írni a primimplikáns táblát a szükséges primimplikánsok kiválasztásához. Itt is azokat a primimplikánsokat érdemes felírni először, melyek több kimeneti függvényhez tartoznak, azaz amelyeknek a legtöbb jelzőbitje 1. A példában a 3, 7 és a 14 primimplikáns 111 jelzőbit sorrendje azt jelenti, hogy mind a három kimenet tartalmazza a primimplikánsot, azaz a szorzatfüggvényük is. A 9 primimplikáns 011 jelzőbit sorozata szerint a primimplikáns az F_2 és F_3 kimeneti függvényekben és ezek szorzatfüggvényében szerepel. A Karnaugh táblás módszerrel szemben itt nincs probléma azzal, hogy egy primimplikáns többször is generálva lenne az egyszerűsítés során, ezért a sorrend megkötése inkább az egységesítés érdekében történik.

Sor	Prímimplikáns	F_1						F_1					F_1					
		3	7	8	10	12	14	1	3	7	9	14	3	4	6	7	9	14
$F_1 F_2 F_3$	* 3, 7	a	x	x					x	x			x				x	
	* 14	b					x				x							x
$F_2 F_3$	* 9	c								x							x	
F_1	* 8, 10, 12, 14	d		x	x	x	x											
F_2	1, 3	e						x	x									
	1, 9	f						x			x							
F_3	6, 7	g												x	x			
	* 4, 6	h											x	x				
	6, 14	i												x				x

A primimplikáns tábla megegyezik a grafikus minimalizálásnál kapott primimplikáns táblával, azzal a különbséggel, hogy itt a primimplikánsok nem algebrai alakban vannak beírva a táblázatba. Könnyen ellenőrizhető, hogy a primimplikánsok megegyeznek.

Decimális alak	Algebrai alakok	Eredő algebrai alak
3, 7	$\bar{A}\bar{B}CD$ $\bar{A}BCD$	$\bar{A}CD$
14	$ABC\bar{D}$	$ABC\bar{D}$
9	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}\bar{D}$
8, 10, 12, 14	$A\bar{B}\bar{C}\bar{D}$ $A\bar{B}C\bar{D}$ $AB\bar{C}\bar{D}$ $ABC\bar{D}$	$A\bar{D}$
1, 3	$\bar{A}\bar{B}\bar{C}D$ $\bar{A}\bar{B}CD$	$\bar{A}\bar{B}D$
1, 9	$\bar{A}\bar{B}\bar{C}D$ $A\bar{B}\bar{C}\bar{D}$	$\bar{B}\bar{C}D$
6, 7	$\bar{A}BC\bar{D}$ $\bar{A}BCD$	$\bar{A}BC$
4, 6	$\bar{A}\bar{B}\bar{C}\bar{D}$ $\bar{A}\bar{B}C\bar{D}$	$\bar{A}\bar{B}\bar{D}$
6, 14	$\bar{A}BC\bar{D}$ $ABC\bar{D}$	$BC\bar{D}$

Mivel a prímiplikáns táblázat ugyanaz, mint az előző esetben, ezért ugyanaz a megoldás mind a két módszerrel. A számjegyes minimalizálás előnye, hogy több változó esetén is alkalmazható és nem generálja ugyanazokat a prímiplikánsokat többször.

Nem teljesen specifikált hálózatok minimalizálása

Több kimenettel rendelkező, nem teljesen specifikált hálózatok esetén ugyanúgy kell a minimalizálást elvégezni, mint egy kimenettel rendelkezők esetén. A határozatlan állapotokat is fel kell venni implikánsnak a kiinduló táblázatban és használni őket az összevonásoknál. Valamint a prímiplikánsok kiválasztásánál nem kell őket lefedni, a prímiplikáns táblázatból ki lehet hagyni őket. Így a segédfüggvényben sem szerepelnek a határozatlan mintermekhez tartozó logikai kifejezések.

A példa legyen ugyanaz, mint a teljesen specifikált hálózat Karnaugh táblás egyszerűsítés esetében, azaz egy négy bemenetű és kettő kimenetű kombinációs hálózat minimális realizációjának kell meghatározni diszjunktív formában. A kettő kimeneti függvény diszjunktív normál formában adott a következőképpen. $F_1^{n=4} = \sum_{i=0}^{2^n-1} (4,5,7,8,12) + (6,9)$ és $F_2^{n=4} = \sum_{i=0}^{2^n-1} (5,6,9,14,15) + (7,8)$. Ebből fel lehet írni a kiindulási táblázatot.

Minterm	Bináris érték	Bináris súly	F_1	F_2
4	0100	1	1	0
8	1000		1	1
5	0101	2	1	1
6	0110		1	1
9	1001		1	1
12	1100		1	0
7	0111	3	1	1
14	1110		0	1
15	1111	4	0	1

A következő lépésben elkészíthetők a kettő mintermet tartalmazó implikánsok.

Minterm	Bináris érték	Bináris súly	F_1	F_2	Implikáns	Decimális különbség	F_1	F_2
4+	0100	1	1	0	4, 5	(1)	1	0
8+	1000		1	1	4, 6	(2)	1	0
5+	0101	2	1	1	4, 12	(8)	1	0
6+	0110		1	1	8, 9	(1)	1	1
9+	1001		1	1	8, 12	(4)	1	0
12+	1100		1	0	5, 7	(2)	1	1
7+	0111	3	1	1	6, 7	(1)	1	1
14+	1110		0	1	6, 14	(8)	0	1
15+	1111	4	0	1	7, 15	(8)	0	1
					14, 15	(1)	0	1

A négy mintermet tartalmazó implikánsok generálása hasonló módon történik, mint a kettő mintermet tartalmazóké. Egyetlen négy mintermet tartalmazó implikáns van, ami természetesen két féle módon is generálható ugyanúgy, mint egy kimenettel rendelkező hálózatok esetében.

Implikáns	Decimális különbség	F_1	F_2	Implikáns	Decimális különbség	F_1	F_2
4, 5+	(1)	1	0	4, 5, 6, 7	(1, 2)	1	0
4, 6+	(2)	1	0	6, 7, 14, 15	(1, 8)	0	1
4, 12	(8)	1	0				
8, 9	(1)	1	1				
8, 12	(4)	1	0				

5, 7	(2)	1	1
6, 7	(1)	1	1
6, 14+	(8)	0	1
7, 15+	(8)	0	1
14, 15+	(1)	0	1

Mivel a kettő darab négy mintermet tartalmazó implikánshoz különböző decimális különbségek valamint különböző jelzőbitek tartoznak, ezért további egyszerűsítésre nincs mód. A grafikus minimalizáláshoz hasonlóan fel kell írni a prímiimplikáns táblát a szükséges prímiimplikánsok kiválasztásához.

Sor	Prímiimplikáns	F_1					F_2				
		4	5	7	8	12	5	6	9	14	15
*	6, 7	a			x			x			
$F_1 F_2$	5, 7	b		x	x		x				
	8, 9	c				x			x		
F_1	4, 5, 6, 7	d	x	x	x						
	4, 12	e	x							x	
	$A\bar{C}\bar{D}$	f				x	x				
F_2	* 6, 7, 14, 15	g						x		x	x

Mivel a prímiimplikáns táblázat ugyanaz, mint az előző esetben, ezért ugyanaz a megoldás mind a két módszerrel.

7. Szimmetrikus logikai függvények

Bevezetés, definíciók

A korábbi fejezetekben (5.-6.) megismert függvényminimalizálási eljárások, akkor lehetnek hatékonyak, hogyha a minimalizálni kívánt függvény(ek) szomszédos mintermeket (maxtermeket) tartalmaz(nak). Amennyiben nincsenek szomszédos mintermek (maxtermek), akkor ezek az alakok prímisszorzatok is egyben, és ekkor a DNF (KNF) alak a legegyszerűbb megvalósítást jelenti.

Adott a következő teljesen specifikált 3-változós logikai függvény (DNF alakban):

$$F^{n=3}(A, B, C) = \sum_{i=0}^{2^n-1} (1,2,4) = m_1 + m_2 + m_4 = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\overline{C}$$

A függvényegyszerűsítéshez a Karnaugh-táblát felírva, azt tapasztalható, hogy a korábbi fejezetekben megismert összevonási módszerek alkalmazásával sincs lehetőség az egyszerűbb alak megadására (ún. „elszeparált” logikai '1'-es változók szerepelnek a Karnaugh táblán). Az ilyen esetekben, ha lehetséges, más módszert kell alkalmazni a függvényegyszerűsítésekre. Karnaugh tábla és az ekvivalens kanonikus igazságtábla a következő:

		BC		C		F
				B		
A	0	00	01	11	10	
	1	0	1	0	1	
		0	1	3	2	
		4	5	7	6	

m _i	A	B	C	F
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	0

Def. Szimmetrikus függvény: amennyiben az F függvény érzéketlen (változtatlan) marad a független bemeneti változók tetszőleges páronkénti felcserélésére (**permutációjára**), szimmetrikus függvénynek nevezik. Tekintsük például a 'C' → 'A' független változók párcseréjét, ekkor:

$$F^{n=3}(A, B, C) = \overline{C}BA + \overline{C}B\overline{A} + C\overline{B}A = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\overline{C} = m_1 + m_2 + m_4$$

Az előző szemléltető példában szereplő F függvény tehát szimmetrikus függvény. Az is látható, hogy $F(A,B,C) = '1'$ pontosan akkor, ha az 'A', 'B', 'C' független változók közül pontosan csak az egyik logikai értéke '1'.

Def. Szimmetria szám: Minden n-változós szimmetrikus függvényhez megadható legalább egy, vagy több olyan pozitív egész szám ($a_1 \dots a_k$ ahol $k \in \mathbb{N}$, és $0 \leq a_k \leq n$), amelyet szimmetria számnak nevezünk. Ez azt jelenti, hogy $F(A,B,C) = '1'$ függvényérték mellett hány független változó '1'-es logikai (ponált) értéke esetén áll elő az adott szimmetria számmal megadható szimmetrikus függvény. Jele:

$$S_{a_1, \dots, a_k}^n(x_1, \dots, x_n)$$

Az előző példa esetén a szimmetrikus függvény szimmetria száma 1. Jele:

$$S_1^{n=3}(A, B, C)$$

A szimmetria számok létezése szükséges-, és elégséges feltétele annak, hogy egy függvény szimmetrikus legyen. A szimmetria számok között a legnagyobb értéket (a_k) a függvény változóinak száma (n) jelentheti. A legkisebb szimmetriaszám a 0 lehet, amely azt jelenti, hogy a függvény értéke akkor $F(A,B,C)=1$, ha egyetlen változójának az értéke sem '1' (mindegyik negáltan szerepel).

Def. Azokat a szimmetrikus függvényeket, amelyeknek csak egyetlen szimmetriaszámuk van, **kanonikusan szimmetrikus függvényeknek** nevezik. Előző példánkban szereplő szimmetrikus függvénynek egyetlen szimmetria száma volt ($a_k=1$), ezért kanonikusan szimmetrikus függvény is egyben.

Műveletek, transzformációk szimmetrikus függvényekkel

Célunk, hogy a korábbi fejezetekben megismert logikai függvényminimalizálási módszerekkel nem, vagy csak nagyon nehezen egyszerűsíthető feladatokat a szimmetrikus függvények segítségével próbáljuk meg felírni. A rendelkezésre álló szimmetrikus függvények halmazát egy olyan elemkészletnek (adatbázisnak) kell tekinteni, amelyek felhasználásával gazdaságosabban lefedhetők, akár helyettesíthetők a nehezen egyszerűsíthető logikai függvények.

A szimmetrikus függvények a következő 4 műveleti osztályra nézve zártak tekinthetők:

1. Logikai összeadás
2. Logikai szorzás
3. Szimmetrikus függvény negáltja
4. Szimmetrikus függvény független változóinak negáltja (független-változó, vagy 'n-k' transzformáció)

Az ismertetésre kerülő (1.-4.) műveleti tulajdonságok felhasználásával, valamint szimmetria számok értelmezése alapján adott szimmetrikus függvényekből egyszerűen előállíthatók akár más szimmetrikus függvények is. Így, ha az alap építőelem-készlet, mint egy adatbázis rendelkezésre áll, ezekből hierarchikusan építkezve sok más szimmetrikus függvény egyszerűen megvalósítható, anélkül hogy annak minimalizálása gondot okozna.

1.) Logikai összeadás

Két, azonos független (bemeneti) változókon értelmezett szimmetrikus függvény logikai összege (VAGY kapcsolata) egy olyan szimmetrikus logikai függvény lesz, amelynek a szimmetria számai az eredeti két függvény szimmetria számainak az **egyesítéséből (unió)** adódnak.

$$S_{1,3}^{n=3}(A, B, C) + S_{0,3}^{n=3}(A, B, C) = S_{0,1,3}^{n=3}(A, B, C)$$

2.) Logikai szorzás

Két, azonos a független (bemeneti) változón értelmezett szimmetrikus logikai függvény logikai szorzata (ÉS kapcsolata) olyan szimmetrikus logikai függvény, amelynek a szimmetria számai az eredeti két függvény szimmetria számainak a **közös értékeiből (metszet)** adódnak.

$$S_{1,3}^{n=3}(A, B, C) \times S_{0,3}^{n=3}(A, B, C) = S_3^{n=3}(A, B, C)$$

De például az eredmény lehet 0 is, a szimmetria számok értékétől függően:

$$S_{1,3}^{n=4}(A, B, C, D) \times S_{2,4}^{n=4}(A, B, C, D) = 0$$

Megjegyzések: A szorzatfüggvény értéke akkor '1', ha mindkét un. tényezőfüggvény értéke is '1'. Ha a tényezőfüggvényeknek nincsenek közös szimmetria számai, akkor nem létezik olyan változókombináció, amely mellett a tényezőfüggvény '1' lenne, ezáltal a szorzatfüggvény értéke '0'.

3.) Szimmetrikus függvény negáltja

Egy n-változós szimmetrikus logikai függvény negáltja szintén szimmetrikus. A tagadott függvény szimmetria számai (k) az összes olyan n-nél (változók számánál) nem nagyobb természetes számok,

amelyek a kiindulási függvény szimmetria számai között nem szerepelnek ($k \leq n$, ahol $n, k \in N$).
Tehát a tagadás művelete a szimmetria-szám halmazának **kiegészítő (komplementer)** halmaza.

$$\overline{S_{1,2,3}^{n=4}(A, B, C, D)} = S_{0,4}^{n=4}(A, B, C, D)$$

4.) Szimmetrikus függvény változóinak negáltja

Az n -változós szimmetrikus logikai függvényt a változóinak a negáltján értelmezve egy olyan szimmetrikus függvényt kapunk, amelynek a szimmetria számai (k) rendre a kiindulási függvény szimmetria számaiból képezhetők n -ből történő kivonás segítségével („ $n-k$ ” szabály), $k \leq n$, ahol $n, k \in N$.

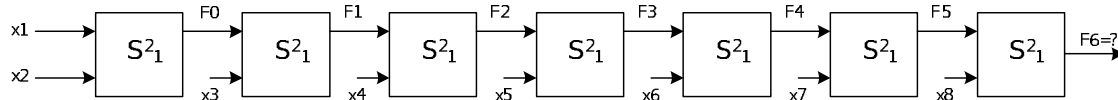
$$S_{2,3}^{n=3}(\overline{A}, \overline{B}, \overline{C}) = S_{0,1}^{n=3}(A, B, C)$$

Példa:

Építőelem egy adatbázisból, legyen a XOR/Kizáró-VAGY kapu, mely rendelkezésre áll.

$$F_0^{n=2}(XOR) = \overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2} \Rightarrow S_1^{n=2}(x_1, x_2)$$

Ebből az elemi kapuból néhányat sorba kapcsolva (lineáris modell) adjuk meg a kimeneti függvényt a következő ábra szerint:



7.1 ábra: Sorba kapcsolt Kizáró-VAGY kapukat definiáló szimmetrikus logikai függvények

Adjuk meg mit kapunk az F_6 kimenetén (egy hét-szintű hálózatban):

$$\begin{aligned} F_1 &= S_1^{n=2}[S_1^{n=2}(x_1, x_2), x_3] = S_{1,3}^{n=3}(x_1, x_2, x_3) \\ F_2 &= S_1^{n=2}[S_{1,3}^{n=3}(x_1, x_2, x_3), x_4] = S_{1,3}^{n=4}(x_1, x_2, x_3, x_4) \\ F_3 &= S_1^{n=2}[S_{1,3}^{n=4}(x_1, x_2, x_3, x_4), x_5] = S_{1,3,5}^{n=5}(x_1, x_2, x_3, x_4, x_5) \\ &\dots \\ F_6 &= S_{1,3,5,7}^{n=8}(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \end{aligned}$$

Ebben az esetben természetesen, ha kiindulási feltételként nemcsak a Kizáró-VAGY kapcsolatnak megfelelő $S_1^{n=2}(x_1, x_2)$ szimmetrikus logikai függvény alkotja az alkatrész adatbázist, akkor a több független változót és szimmetria számot tartalmazó függvényekkel egyszerűbben is előállítható a kívánt szimmetrikus logikai függvény.

Példa: De-Morgan azonosságok bizonyítása szimmetrikus logikai függvényekkel

Bizonyítsa be kétváltozós $S_{k_i}^{n=2}(A, B)$ szimmetrikus logikai függvények szabályainak segítségével a De-Morgan Boole-algebra azonosságait (ahol $k_i, i = 0, \dots, n$ szimmetria számok, amelyeket szintén meg kell határozni az azonosságok teljesüléséhez).

Tudjuk, hogy

$$A \cdot B = S_2^{n=2}(A, B)$$

ÉS kapcsolat kifejezése, illetve

$$A + B = \overline{\overline{A} \cdot \overline{B}} + A \cdot \overline{B} + A \cdot B = S_{1,2}^{n=2}(A, B)$$

VAGY kapcsolat kifejezése

Bizonyítás:

1.) Alak:

A	B	A + B	A · B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

$$\overline{A \cdot B} = \overline{S_2^{n=2}(A, B)} \stackrel{3.tul.}{=} S_{0,1}^{n=2}(A, B) \stackrel{4.tul.}{=} S_{1,2}^{n=2}(\overline{A}, \overline{B}) = \overline{A} + \overline{B}$$

$m-k'$

2.) Alak:

$$\overline{A + B} = \overline{S_{1,2}^{n=2}(A, B)} \stackrel{3.tul.}{=} S_0^{n=2}(A, B) \stackrel{4.tul.}{=} S_2^{n=2}(\overline{A}, \overline{B}) = \overline{A} \cdot \overline{B}$$

$m-k'$

Példa: Szimmetrikus függvény algebrai alakjának felírása

$S_{2,3}^{n=4}(A, B, C, D)$ szimmetrikus logikai függvény algebrai alakját úgy célszerű felírni, hogy rendre képezni kell az összes lehetséges olyan mintermet, amelyben *két* változó ponált értékével fordul elő, majd pedig az összes lehetséges olyan mintermet írjuk fel, amelyben *három* változó szerepel ponált értékével:

$$\begin{aligned} S_{2,3}^{n=4}(A, B, C, D) = & \\ = & \underbrace{\overline{ABCD} + \overline{A\overline{B}CD} + \overline{AB\overline{C}D} + \overline{ABC\overline{D}}}_{2} + \\ & \underbrace{\overline{A\overline{B}C\overline{D}} + \overline{A\overline{B}C\overline{D}} + \overline{A\overline{B}C\overline{D}} + \overline{A\overline{B}C\overline{D}}}_{3} \end{aligned}$$

Megjegyzés: Segítség a felíráshoz

Ha n változó van és a_k a felírni kívánt szimmetria szám, akkor a kombinatorikában megismert kombináció képletét lehet alkalmazni azért, hogy egyetlen minterm tagot se hagyjunk el (azaz mennyi minterm képezhető a_k ponált esetén):

$$C_{a_k}^n = \binom{n}{a_k} = \frac{n!}{a_k!(n-a_k)!}$$

$$S_{2,3}^{n=4}(A, B, C, D)$$

A következő módon 6 lehetséges minterm képezhető, amelyben $a_k = 2$ ponált változó szerepel:

$$C_{a_1=2}^{n=4} = \binom{4}{2} = \frac{4!}{2!(4-2)!} = 6$$

A következő módon 4 lehetséges minterm képezhető, amelyben $a_k = 3$ ponált változó szerepel:

$$C_{a_2=3}^{n=4} = \binom{4}{3} = \frac{4!}{3!(4-3)!} = 4$$

Példa: Szimmetrikus függvény algebrai alakjának felírása

$S_{0,2}^{n=4}(A, B, C, D)$ szimmetrikus logikai függvény algebrai alakját úgy célszerű felírni, hogy rendre képezzük az összes lehetséges olyan mintermet, amelyben *egyetlen* változó *sincs* ponált értékével, majd pedig az összes lehetséges olyan mintermet írjuk fel, amelyben *két* változó szerepel ponált értékével:

$$\begin{aligned} S_{0,2}^{n=4}(A, B, C, D) = & \\ = & \overline{ABCD} + \\ & \overline{A\overline{B}CD} + \overline{A\overline{B}C\overline{D}} + \overline{A\overline{B}C\overline{D}} + \overline{A\overline{B}C\overline{D}} + \overline{A\overline{B}C\overline{D}} \end{aligned}$$

Példa: Szimmetrikus függvények logikai összege:

A mintermek algebrai egyszerűsítései (nevezetes azonosságai) révén a következő alakot kapjuk:

$$\begin{aligned}
& S_{2,3}^{n=4}(A, B, C, D) + S_{0,2}^{n=4}(A, B, C, D) = \\
& \left. \begin{aligned}
& \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + \\
& \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}BC\overline{D} + \overline{A}BC\overline{D} + \overline{A}BC\overline{D} + \overline{A}BC\overline{D} + \\
& \overline{A}BCD + \\
& \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BC\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D}
\end{aligned} \right\} S_{2,3}^{n=4} \\
& = S_{0,2,3}^{n=4}(A, B, C, D)
\end{aligned}$$

Szimmetrikus függvények egyszerűsítési szabályai

Korábbi példák között voltak olyan szimmetrikus függvények, amelyek kanonikus normálalakja egyben a legegyszerűbb DNF alak is (például Kizáró-VAGY). Azonban a szimmetria számoktól függően előfordulhat, hogy a szimmetrikus függvény „bizonyos mértékben” egyszerűsíthető, mivel a szimmetria szám azt is kifejezi egyben, hogy a függvény tartalmaz-e szomszédos mintermeket.

Def.: A **kanonikus szimmetrikus** függvények nem tartalmaznak szomszédos mintermeket, mert csak egy szimmetria számuk van, tehát nem egyszerűsíthetők.

Def.: **Nem egyszerűsíthetők** azok a szimmetrikus függvények sem, amelynek bármely két szimmetria száma közötti különbsége nagyobb 1-nél.

Def.: (előző következménye). Ha egy függvénynek van legalább két olyan szimmetria száma, amelyek közötti különbség 1, akkor ezeknek megfelelő mintermek között biztosan van szomszédos, tehát **egyszerűsíthető**.

Példa: Nem egyszerűsíthető függvényre:

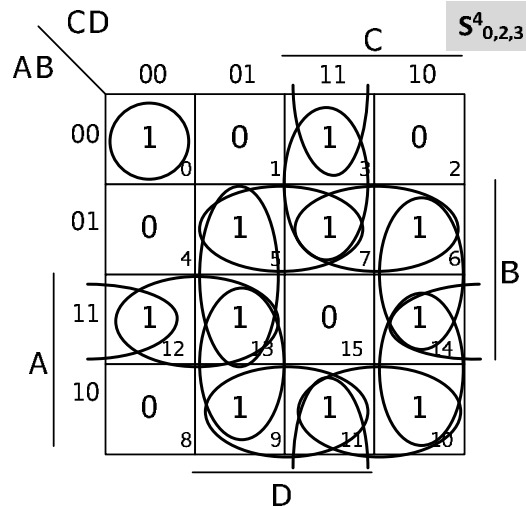
$S_{0,2}^{n=4}(A, B, C, D)$ szimmetrikus függvény Karnaugh táblájának megadása a következő:

		CD		C		$S_{0,2}^4$
		00	01	11	10	
A	B	00	1 ₀	0 ₁	1 ₃	0 ₂
		01	0 ₄	1 ₅	0 ₇	1 ₆
	11	1 ₁₂	0 ₁₃	0 ₁₅	0 ₁₄	
	10	0 ₈	1 ₉	0 ₁₁	1 ₁₀	
		D				

Szimmetria számok különbsége 2, ezért a szimmetrikus függvény nem egyszerűsíthető.

Példa:

$S_{0,2,3}^4(A, B, C, D)$ szimmetrikus függvény Karnaugh táblájának felírása a következő:



Mivel a szimmetria számok között (0,2,3) van legalább kettő olyan (2,3), amelyek között a különbség 1, ezért a függvény egyszerűsíthető: kettes lefedő hurkok képezhetők. A fenti ábrán az összes lehetséges kételemű összevonás jelölve van.

Szimmetrikussá tehető F függvény minimalizálása egy példán keresztül TSH – Teljesen Specifikált Hálózat esetén

Ez a módszer egy kevésbé szisztematikus eljárás alapul. Intuitív módon kell megvizsgálni az S szimmetrikus függvényhez pótlólagosan hozzáadandó, illetve elvenni szükséges mintermekkel kifejezett függvényeket. Ezzel szemben viszont egy mindig célra vezető eljárást kapunk az adatbázisban lévő megadott S szimmetrikus függvény(ek)től függően.

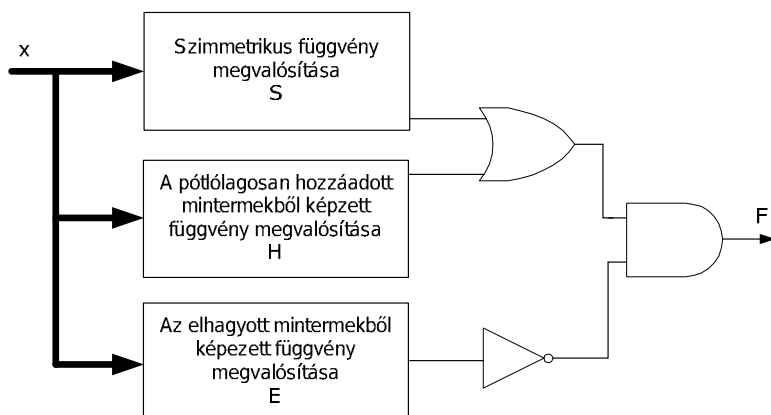
Felmerülő kérdések:

- Ha adott egy F függvény, amely nehezen, vagy egyáltalán nem összevonható mintermeket tartalmaz, akkor kifejezhető-e szimmetrikus S függvény segítségével (akár F -nek pótlólagos megváltoztatásával is)?
- Az így megadott S szimmetrikus függvénnyel nem egyszerűbb-e kifejezni az F -függvényt, mint az F legegyszerűbb DNF alakjával?

Feladat: Fejezzük ki F -et a szimmetrikus S függvény (építőelem) segítségével úgy, hogy S -hez pótlólagosan mintermeket adunk hozzá (H), illetve mintermeket veszünk el (E).

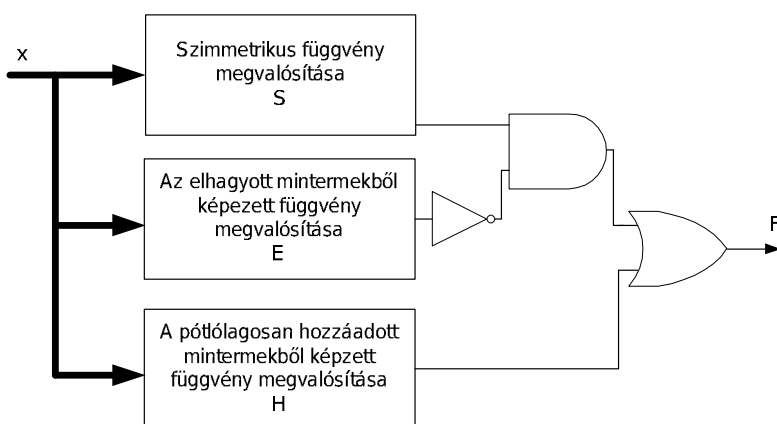
Általánosan: szimmetrikussá tehető TSH/NTSH függvények minimalizálása a következő két módszer szerint történik.

I. módszer: $F = (S + H) \cdot \overline{E}$



7.2 ábra: Függvények minimalizálása szimmetrikus logikai függvénnyel: I. módszer

II. módszer: $F = S \cdot \bar{E} + H$



7.3 ábra: Függvények minimalizálása szimmetrikus logikai függvénnyel: II. módszer

Mint a későbbi példákban látható, a két módszer (I-II.) nem feltétlenül vezet azonos megoldáshoz. Ez azt is jelenti egyben, hogy az F függvény kialakításakor az S szimmetrikus függvényből elvett (E), illetve hozzáadott (H) mintermek műveletének sorrendje kötött lehet.

Példa: Teljesen specifikált hálózatot (TSH) leíró F függvény minimalizálása szimmetrikus S függvénnyel történő kifejezéssel.

Legyen adott az alábbi F függvény

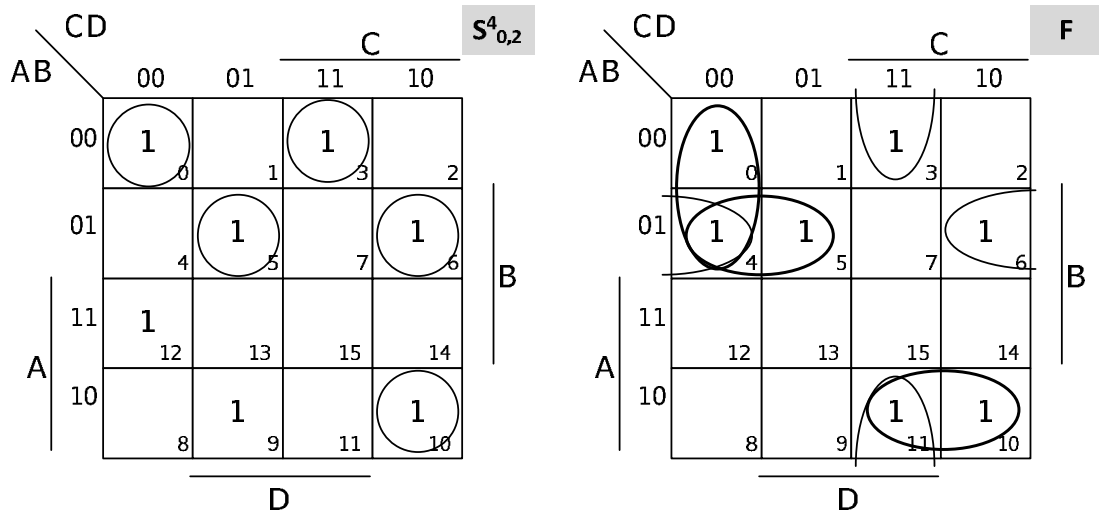
$$F^{n=4}(A, B, C, D) = \sum_{i=0}^{2^n-1} (0,3,4,5,6,10,11)$$

Kérdések:

Adott F függvény kifejezhető-e az adott $S_{0,2}^{n=4}(A, B, C, D)$ szimmetrikus logikai függvény segítségével?

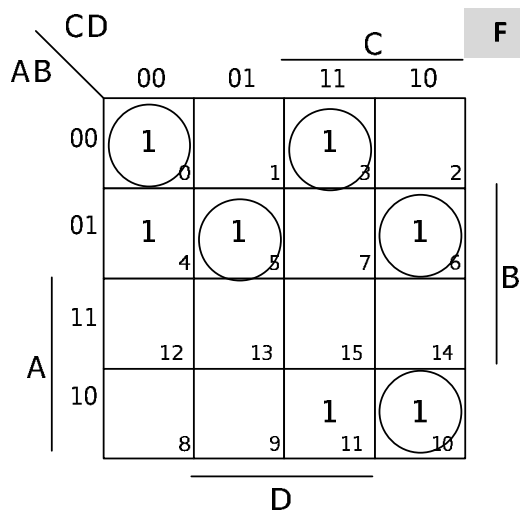
- Az így adott S szimmetrikus függvénnyel nem egyszerűbb-e kifejezni az F -függvényt, mint az F legegyszerűbb DNF alakjával?

Az F függvény, illetve a megadott szimmetrikus S függvény Karnaugh táblái a következő ábrán látható (jelölve az összes lehetséges összevonásokat):

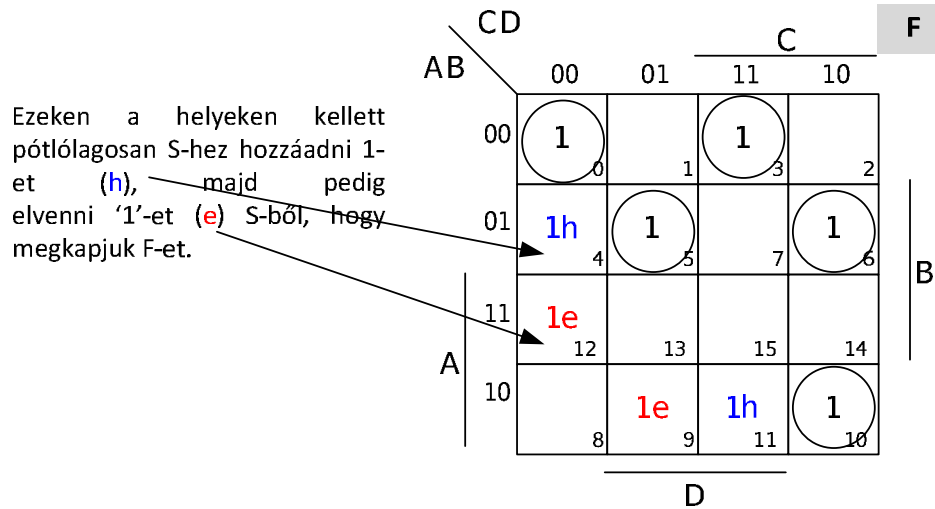


7.4 ábra: a.) F függvény, és b.) S szimmetrikus függvény Karnaugh táblái

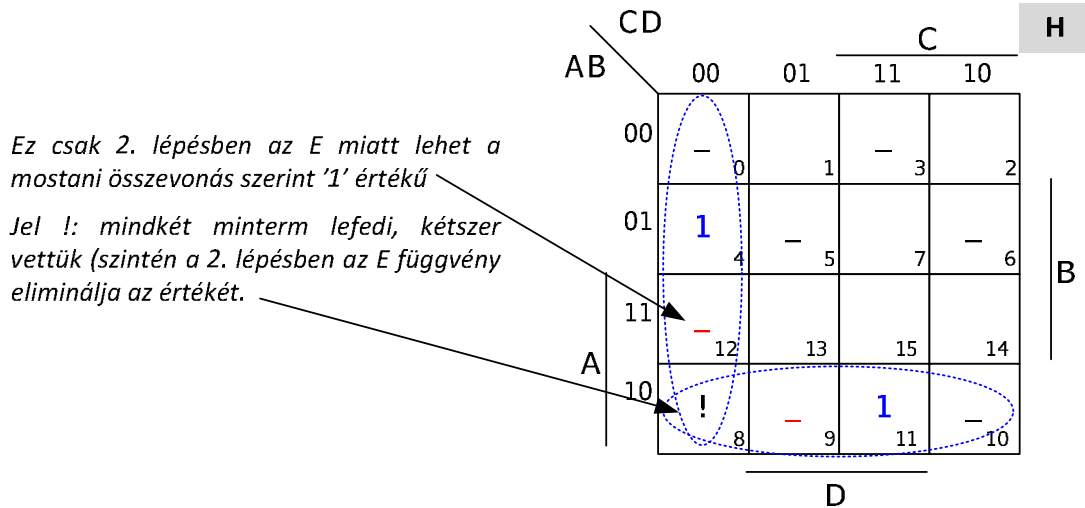
Elsőként rögzítsük '1'-es értékkel azokat a cellákat az F függvény Karnaugh táblájában, amelyeknek van közös része az S szimmetrikus függvénnyel:



Jelölés: O kialakítható közös '1'-esek (F és S között)

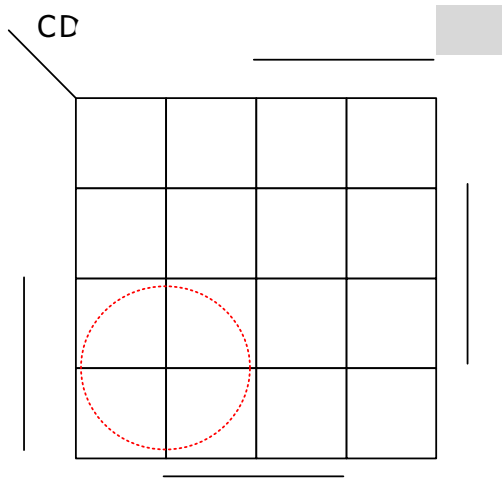


I. módszer szerint: $F = (S + H) \cdot \bar{E}$



Az első lépésben az S-hez pótlólagosan hozzáadandó mintermeket tartalmazó H függvényben a létező legnagyobb lefedéseket vesszük (amelyben az F értéke '1h' volt). Ahol F = '1' volt, oda jelölésként '-'-t tettünk.

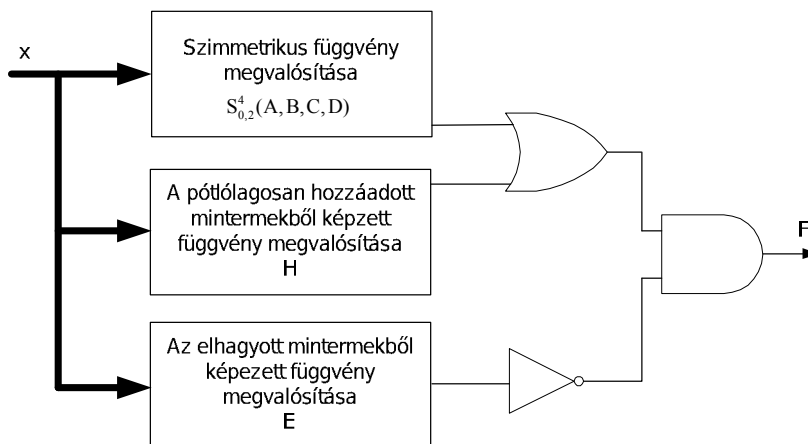
A második lépésben S-ből elvenni szükséges mintermeket tartalmazó E függvényben a létező legnagyobb lefedéseket kell választani (ahol F értéke '1e' volt). Ahol F = '0' volt, oda jelzésként '-'-t tettünk.



I. módszer megoldása: F kifejezése S-el (számít a sorrend $H \rightarrow E$)

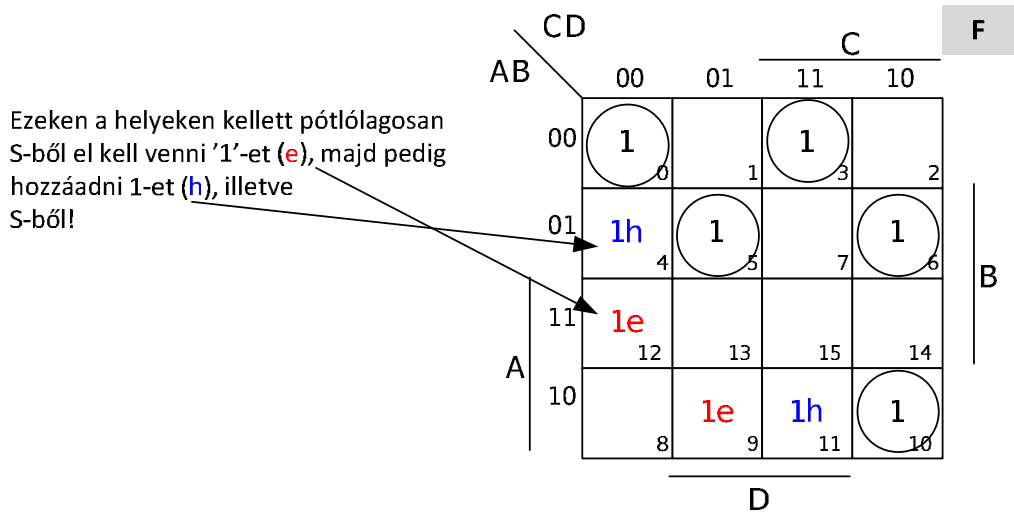
$$F = (S + H) \cdot \bar{E} = [S_{0,2}^{n=4}(A, B, C, D) + (\overline{AB} + \overline{CD})] \cdot \overline{AC}$$

Elvi logikai rajz, I. módszer esetén: $F = (S + H) \cdot \bar{E} = [S_{0,2}^4(A, B, C, D) + (\overline{AB} + \overline{CD})] \cdot \overline{AC}$

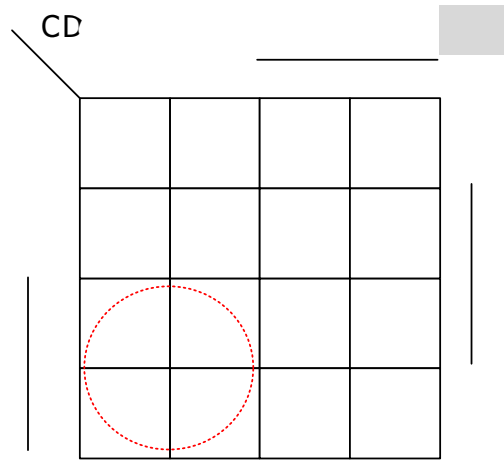


7.5 ábra: mintermek pótlólagos hozzáadásával (H), majd elhagyásával (E) szimmetrikus függvényként kifejezett F függvény megvalósításának felépítése I. módszer esetében

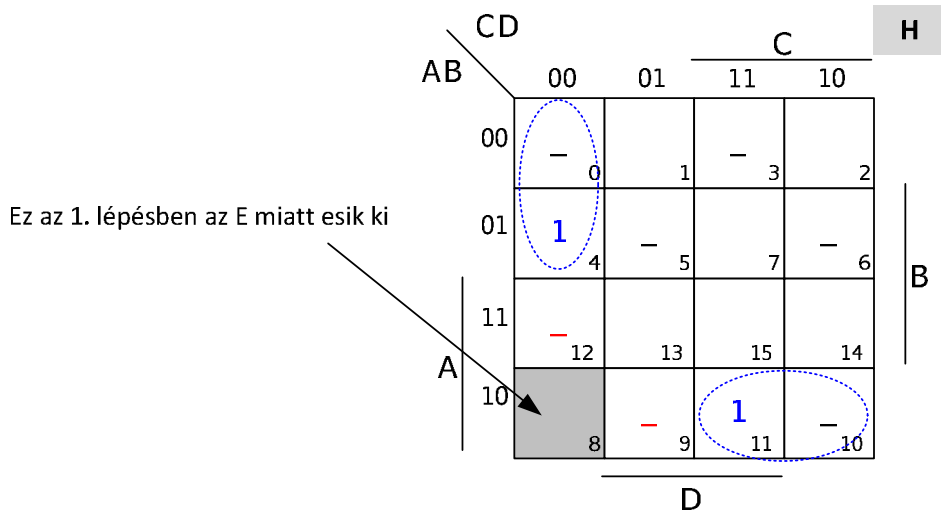
II: módszer szerint: $F = S \cdot \bar{E} + H$



Az első lépésben S-ből elvenni kívánt mintermeket tartalmazó E függvényben a létező legnagyobb lefedéseket kell venni (ahol F értéke '1e' volt). Ahol F = '0' volt, oda jelzésként '_'-t tettünk.



Az második lépésben az S-hez pótlólagosan hozzáadandó mintermeket tartalmazó H függvényben más módon, de azokat a kételemű lefedéseket válasszuk ki, amelyben az F értéke '1h' volt. Ahol F = '1' volt, oda jelölésésként '_'-t tettünk.



II. módszer megoldása: F kifejezése S-el (számít a sorrend $E \rightarrow H$)

$$\begin{aligned}
 F^{n=4} &= [S_{0,2}^{n=4}(A, B, C, D) \cdot \overline{E}] + H = \\
 &= [S_{0,2}^{n=4}(A, B, C, D) \cdot \overline{AC}] + (\overline{ABC} + \overline{ACD})
 \end{aligned}$$

Szimmetrikussá tehető F függvény minimalizálása „bináris súlyok” eljárásának használatával egy példán keresztül bemutatva

Ez az egyszerű eljárás nem intuitív kereséssel, hanem szisztematikus módon megadja azt, hogy az F függvény kifejezhető, egyszerűsíthető-e S szimmetrikus logikai függvény segítségével. Előnye, hogy sok változó esetén ($n > 4$) is jól alkalmazható a szimmetrikus függvény megkeresésére, míg a korábbi eljárás során nehézkessé válik a Karnaugh táblák kezelése.

Az eljárás hátránya viszont a korábban vizsgált intuitív módszerrel szemben, hogy nem minden esetben ad egyértelmű megoldást: pontosabban, ha a bináris súlyok használatával nem lehet minimalizálni az F függvényt (és megadni a szimmetrikus függvényt), attól annak még létezhet a korábban megismert intuitív eljárással egyszerűsíthető megoldása.

Ennek az új eljárásnak az alapja a **szimmetria szám, mint bináris súly**.

A szimmetria számok az F logikai függvényben szereplő **mintermeknek** megfelelő bináris kombinációk **bináris súlyait** definiálják.

Vizsgálati módszer (**bináris súly előfordulása**):

Def.: Ha egy S függvény **szimmetrikus**, akkor az adott szimmetria számnak megfelelő **bináris súlyt** jelentő összes képezhető mintermet tartalmazza.

Ha változók száma 'n', és a szimmetria szám 'k' ($k \leq n$, ahol $n, k \in \mathbb{N}$), akkor a szimmetrikus függvény tartalmazza az összes képezhető 'k'-értékű bináris súlyt képviselő 'n'-változós mintermet, azaz (kombinációjukat), amely a következő módon számítható ki:

$$C_k^n = S_k^n = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Példa:

Adott a következő F logikai függvény DNF alakban:

$$F^{n=5}(A, B, C, D, E) = \sum_{i=0}^{2^n-1} (3,4,8,12,13,14,17,18,23,27,28)$$

Kérdés: felírható-e 'F' függvény szimmetrikus függvény alakban?

a.) Bináris-súly táblázat: változók és bináris súlyaik felírása a DNF alapján. A következő lépésben pedig a táblázat alapján a bináris súly előfordulásának feltételét kell megvizsgálni (táblázat mellett). Ekkor kell alkalmazni a korábban már megismert kombináció képletét a szimmetria számokra, és a ponált értékek előfordulására

minterm	A	B	C	D	E	Bináris súly
3	0	0	0	1	1	#2
4	0	0	1	0	0	#1
8	0	1	0	0	0	#1
12	0	1	1	0	0	#2
13	0	1	1	0	1	#3
14	0	1	1	1	0	#3
17	1	0	0	0	1	#2
18	1	0	0	1	0	#2
23	1	0	1	1	1	#4
27	1	1	0	1	1	#4
28	1	1	1	0	0	#3
#1/ #0	5/6	6/5	6/5	5/6	5/6	

Bináris súlyok előfordulása (kombináció):

#0 bináris súly: -

#1 bináris súly: **nem OK**

$$C_1^5 = \binom{5}{1} = \frac{5!}{1!(5-1)!} = 5 - \text{ször}$$

#2 bináris súly: **nem OK**

$$C_2^5 = \binom{5}{2} = \frac{5!}{2!(5-2)!} = 10 - \text{szer}$$

#3 bináris súly: **nem OK**

$$C_3^5 = \binom{5}{3} = \frac{5!}{3!(5-3)!} = 10 - \text{szer}$$

#4 bináris súly: **nem OK**

$$C_4^5 = \binom{5}{4} = \frac{5!}{4!(5-4)!} = 5 - \text{ször}$$

#5 bináris súly: -

Ahogy látható, egyik előfordulás sem teljesül a táblázat alapján: a kombinációs képlettel kiszámolt előfordulások mindegyikének egyeznie kellene a táblázatban kiszámolt bináris súlyok előfordulásának számával (a táblázat alján az #1 – egyesek számát, #0 – nullák számát jelölik „x/y”-os alakban meghatározva.). Ez azonban még nem jelenti, hogy a függvény nem tehető szimmetrikussá.

b.) Lehetőség: bármely független változó(ka)t „felcserélve” a kimeneti függvény változatlan marad.

Ilyen esetben még egy további feltételt kell megvizsgálni, nevezetesen, hogy a „x/y” alakok kismértékű változtatásával („x/y” reciprokával → „y/x”) lehetséges-e, hogy az bináris súlyok előfordulásai a táblázat és a kombinációs képletek eredményét összehasonlítva teljesülnek. Ekkor a megváltoztatott alakok oszlopában lévő logikai értékek ellentettjét (negált értékét) kell venni. A változtatás során azt kell megvizsgálni, hogy lehetséges-e tisztán „x/y” vagy „y/x”-es alakokat előállítani. Ez azt is jelenti egyben, hogy szimmetria esetén minden (n) változónak azonos számúszor kell ponáltan, illetve negáltan előfordulnia a DNF alakban.

Az előző táblázatban a törtszámok, azaz az '1'-esek (ponált), illetve '0'-ások (negált) értékek száma nem azonosak (#1/#0). De a 6/5 felírható az 5/6 reciproka-ként, ami a bináris súlyokat tekintve annyit tesz, hogy tagadjuk a megfelelő független változókat, azért, hogy az „x/y” alakok azonossága fennálljon.

Változtassuk, azaz negáljuk meg B és C független változók értékét: mivel változik a bináris súly, ezért a kombinációkat újból ki kell számolni, és meg kell vizsgálni az egyes bináris súlyok előfordulásait.

minterm	A	\bar{B}	\bar{C}	D	E	Bináris súly
3	0	1	1	1	1	#4
4	0	1	0	0	0	#1
8	0	0	1	0	0	#1
12	0	0	0	0	0	#0
13	0	0	0	0	1	#1
14	0	0	0	1	0	#1
17	1	1	1	0	1	#4
18	1	1	1	1	0	#4
23	1	1	0	1	1	#4
27	1	0	1	1	1	#4
28	1	0	0	0	0	#1
#1/ #0	5/6	5/6	5/6	5/6	5/6	

Bináris súlyok előfordulása (kombináció):

#0 bináris súly: **OK**

$$C_0^5 = \binom{5}{0} = \frac{5!}{0!(5-0)!} = 1 - \text{szer}$$

#1 bináris súly: **OK**

$$C_1^5 = \binom{5}{1} = \frac{5!}{1!(5-1)!} = 5 - \text{ször}$$

#2 bináris súly: -

#3 bináris súly: -

#4 bináris súly: **OK**

$$C_4^5 = \binom{5}{4} = \frac{5!}{4!(5-4)!} = 5 - \text{ször}$$

#5 bináris súly: -

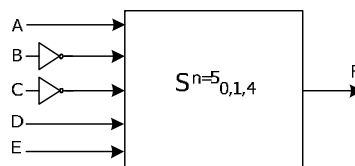
Ahogy a fenti táblázat alapján is látható, a B és C független változók negálásával az egyes mintermekhez tartozó logikai értékek ellentettjét kell képezni. Ez alapján a bináris súlyokat újból ki kell számolni, végül pedig a kombinációs képletbe behelyettesíteni. Így a #0-ás, #1-es illetve #4 es bináris súlyokból a mintermek között rendre, 1, 5, illetve 5 az előfordulás.

Ekkor azt lehet mondani, hogy a bináris súlyok módszerével az F függvény megvalósítható S szimmetrikus logikai függvény segítségével, amelyeknek szimmetria számai a bináris súlyok lesznek!

F felírása a szimmetria alapján pedig a következő:

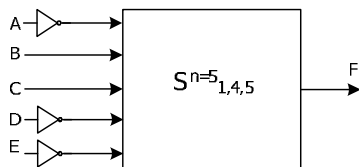
$$F^{n=5}(A, B, C, D, E) = S_{0,1,4}^{n=5}(A, \bar{B}, \bar{C}, D, E)$$

Azzal a feltételezéssel élve, hogy a szimmetrikus függvény rendelkezésre áll az adatbázisban, realizálható az F logikai függvény a következő ábrával:



Megjegyzés: a fenti táblázatban lehetett volna a B,C változók helyett az A,D,E változókat is tagadni természetesen ($x/y = 5/6 \rightarrow y/x = 6/5$ átalakítással). Ekkor, ahogy várjuk a kapott függvény negáltját kapjuk (a korábban megismert 4. szimmetria tulajdonság – „független változók negáltja” alapján):

$$F^{n=5}(A, B, C, D, E) = S_{1,4,5}^{n=5}(\bar{A}, B, C, \bar{D}, \bar{E}) \Leftrightarrow_{m=kt} S_{0,1,4}^{n=5}(A, \bar{B}, \bar{C}, D, E)$$



Példa:

Ha lehetséges, tervezze meg az $S_{0,1}^{n=4}(A, B, C, D)$ szimmetrikus függvény felhasználásával, a bináris súlyok vizsgálata alapján azt az $n=4$ bemenetű 1-kimenetű $F^{n=4}(A, B, C, D)$ kombinációs hálózatot, amelynek a kimenete akkor '1'-es, ha az A és B bemenetei kevesebb számú '1'-est tartalmaznak, mint a C és D bemenetei.

Megoldás (szöveges feladat felírása logikai igazság táblázattal, #1-esek száma esetén, ha $A, B < C, D$):

minterm	A	B	C	D	Bináris súly
1	0	0	0	1	#1
2	0	0	1	0	#1
3	0	0	1	1	#2
7	0	1	1	1	#3
11	1	0	1	1	#3
#1/ #0	1/4	1/4	4/1	4/1	

Bináris súlyok előfordulása (kombináció):

#0 bináris súly: -

#1 bináris súly: **nem OK**

$$C_1^4 = \binom{4}{1} = \frac{4!}{1!(4-1)!} = 4 - \text{szer}$$

#2 bináris súly: **nem OK**

$$C_2^4 = \binom{4}{2} = \frac{4!}{2!(4-2)!} = 2 - \text{szer}$$

#3 bináris súly: **nem OK**

$$C_3^4 = \binom{4}{3} = \frac{4!}{3!(4-3)!} = 4 - \text{szer}$$

Ez megfelel a következő logikai függvénynek:

$$F^{n=4}(A, B, C, D) = \sum_{i=0}^{15} (1,2,3,7,11)$$

(a táblázat alján az #1 – egyesek számát, #0 – nullák számát jelölik „x/y”-os alakban meghatározva.).

I. lehetséges módszer: „x/y” alakra hozás

Ahogy a fenti táblázat alapján is látható, akár az A és B független változók negálásával is lehet az egyes mintermekhez tartozó logikai értékek ellentettjét képezni, úgy hogy minden oszlopban a kiszámolt #1/#0 megoszlása megegyezzen. A negálás alapján a bináris súlyokat újból ki kell számolni, végül pedig a kombinációs képletbe behelyettesíteni, megvizsgálva a szimmetria számok előfordulását.

minterm	\bar{A}	\bar{B}	C	D	Bináris súly
1	1	1	0	1	#3
2	1	1	1	0	#3
3	1	1	1	1	#4
7	1	0	1	1	#3
11	0	1	1	1	#3
#1/ #0	4/1	4/1	4/1	4/1	

Bináris súlyok előfordulása (kombináció):

#0 bináris súly: -

#1 bináris súly: -

#2 bináris súly: -

#3 bináris súly: **OK**

$$C_3^4 = \binom{4}{3} = \frac{4!}{3!(4-3)!} = 4 - \text{szer}$$

#4 bináris súly: **OK**

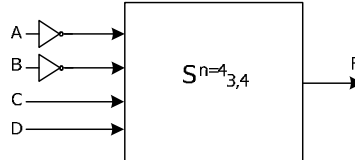
$$C_4^4 = \binom{4}{4} = \frac{4!}{4!(4-4)!} = 1 - \text{szer}$$

Ekkor azt lehet mondani, hogy a bináris súlyok módszerével az F függvény megvalósítható S szimmetrikus logikai függvény segítségével, amelyeknek szimmetria számai a bináris súlyok lesznek!

F felírása a szimmetria alapján pedig a következő:

$$F^{n=4}(A, B, C, D) = S_{3,4}^{n=4}(\bar{A}, \bar{B}, C, D)$$

Azzal a feltételezéssel élve, hogy az S szimmetrikus függvény rendelkezésre is áll az adatbázisban, realizálható az F logikai függvény a következő ábrával:



II. lehetséges módszer: „y/x” alakra hozás

Ahogy a fenti táblázat alapján is látható, akár a C és D független változók negálásával is lehet az egyes mintermekhez tartozó logikai értékek ellentettjét képezni, úgy hogy minden oszlopban a kiszámolt #1/#0 megoszlása megegyezzen. A negálás alapján a bináris súlyokat újból ki kell számolni, végül pedig a kombinációs képletbe behelyettesíteni, megvizsgálva a szimmetria számok előfordulását.

minterm	A	B	\bar{C}	\bar{D}	Bináris súly
1	0	0	1	0	#1
2	0	0	0	1	#1
3	0	0	0	0	#0
7	0	1	0	0	#1
11	1	0	0	0	#1
#1/ #0	1/4	1/4	1/4	1/4	

Bináris súlyok előfordulása (kombináció):

#0 bináris súly: **OK**

$$C_0^4 = \binom{4}{0} = \frac{4!}{0!(4-0)!} = 1 - \text{szer}$$

#1 bináris súly: **OK**

$$C_1^4 = \binom{4}{1} = \frac{4!}{1!(4-1)!} = 4 - \text{szer}$$

#2 bináris súly: -

#3 bináris súly: -

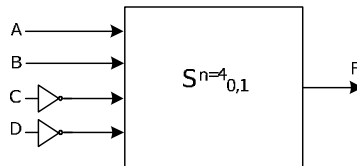
#4 bináris súly: -

Ebben az esetben is azt lehet mondani, hogy a bináris súlyok módszerével az F függvény megvalósítható S szimmetrikus logikai függvény segítségével, amelyeknek szimmetria számai a bináris súlyok lesznek!

F felírása a szimmetria alapján pedig a következő:

$$F^{n=4}(A, B, C, D) = S_{0,1}^{n=4}(A, B, \bar{C}, \bar{D})$$

Azzal a feltételezéssel élve, hogy az S szimmetrikus függvény megtalálható a rendelkezésre álló adatbázisunkban, realizálható az F logikai függvény a következő ábrával:



Következtetés: Az I. és II. módszer összehasonlítása alapján pedig még a következő összefüggést is meg lehet adni, felhasználva a szimmetrikus logikai függvények szimmetria tulajdonságait:

$$F^{n=4}(A, B, C, D) = S_{0,1}^{n=4}(A, B, \bar{C}, \bar{D}) \stackrel{\substack{m-k \\ \text{szabály} \\ (4.tul.)}}{\Leftrightarrow} S_{3,4}^{n=4}(\bar{A}, \bar{B}, C, D)$$

8. Két- és többszintű NEM-ÉS és NEM-VAGY hálózatok tervezése

Egy logikai hálózat megvalósítása során több fajta elemi logikai függvényt is fel lehet használni. Ha csak a két bemenetű logikai függvényeket nézzük, akkor 16 különböző logikai függvény áll rendelkezésre. Ahhoz, hogy tetszőleges logikai függvényt meg lehessen valósítani, nincs szükség az összes alapműveletre.

Funkcionálisan teljes rendszerek

Logikai alapműveletek azon halmazát, melyekkel bármely logikai függvény megvalósítható funkcionálisan teljes rendszernek nevezik. A gyakorlatban három legelterjedtebb funkcionálisan teljes rendszer a NÉV, a NEM-ÉS és a NEM-VAGY rendszerek.

NÉV rendszer

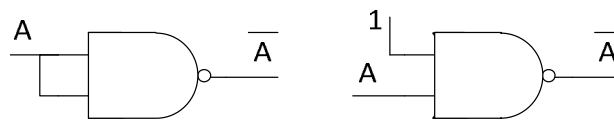
Mint már korábban szó volt róla, minden logikai függvény felírható diszjunktív és konjunktív normál alakban. Ezek az alakok kizárólag ÉS, VAGY valamint NEM logikai alapműveleteket tartalmaznak, azaz ez a három alapművelet együtt funkcionálisan teljes rendszert alkot.

Ezt a rendszert a gyakorlatban nem használják a hardveres korlátok miatt. A chipok általában adott számú egyforma kaput tartalmaz. Ezek után nehéz úgy megtervezni a hardvert, hogy minden chip minden kapuja ki legyen használva. Ha viszont nincs kihasználva, akkor feleslegesen növeli a kapcsolat méretét.

NEM-ÉS rendszer

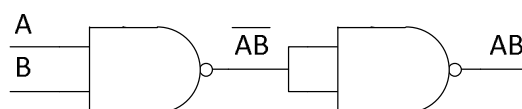
A NEM-ÉS rendszer, mint a neve is mutatja, csak NEM-ÉS kapukat tartalmaz. Ez is funkcionálisan teljes rendszer, azaz minden logikai függvény megvalósítható benne. Ennek bizonyításához a továbbiakban be lesz mutatva, hogy NEM-ÉS rendszerben megvalósítható a NEM, az ÉS valamint a VAGY logikai függvény is.

A NEM logikai függvény kétféleképpen valósítható meg NEM-ÉS kapukkal. Az egyik esetben az ÉS kapcsolat idempotencia tulajdonságát lehet kihasználni, azaz $AA = A$, a másik esetben pedig azt, hogy az 1 értékkel való ÉS kapcsolat eredménye csak a logikai változó értékétől függ, azaz $A \cdot 1 = A$. Ezek alapján az azonosságok mind a két oldalának negálása eredményezi a bizonyítást. $\bar{A} = \overline{AA} = \overline{A \cdot 1}$. A két esethez tartozó logikai áramköri realizációk a következő ábrán láthatóak.



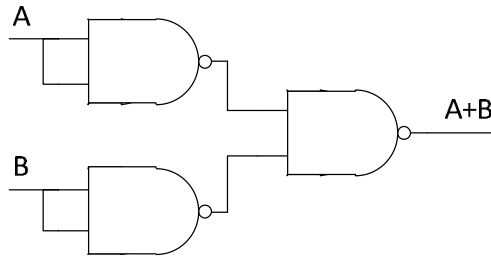
8.1 ábra: \bar{A} megvalósítása NEM-ÉS kapukkal

Ezek alapján az ÉS logikai kapcsolatot megvalósítása NEM-ÉS rendszerben egyszerűen meg lehet valósítani. Azt kell kihasználni, hogy egy logikai kifejezés kétszeres tagadása az eredeti logikai kifejezést eredményezi, azaz $\overline{\bar{A}} = A$. Ez alapján A és B logikai változók ÉS kapcsolata a következő módon állítható elő $AB = \overline{\overline{AB}}$. A logikai áramköri realizációban a NEM függvény megvalósításához az első eset lett felhasználva, de természetesen a másik is használható lett volna.



8.2 ábra: AB megvalósítása NEM-ÉS kapukkal

A VAGY logikai kapcsolat megvalósításához a De-Morgan azonosságra van szükség, mely szerint $\overline{A + B} = \overline{A} \overline{B}$. Mind a két oldalt invertálása egy olyan logikai kifejezést eredményez, melynek bal oldalán VAGY kapcsolat, jobb oldalán pedig ÉS valamint NEM kapcsolat található, azaz $A + B = \overline{\overline{A} \overline{B}}$. Mivel ÉS valamint NEM kapcsolatot meg lehet valósítani NEM-ÉS rendszerben, ezért a VAGY kapcsolat is megvalósítható, melynek logikai áramköri realizációja a következő ábrán látható.

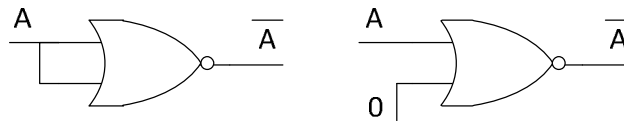


8.3 ábra: $A + B$ megvalósítása NEM-ÉS kapukkal

NEM-VAGY rendszer

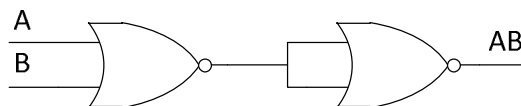
A NEM-VAGY rendszer, mint a neve is mutatja, csak NEM-VAGY kapukat tartalmaz. Ez is funkcionálisan teljes rendszer, azaz minden logikai függvény megvalósítható benne. Ennek bizonyításához a továbbiakban be lesz mutatva, hogy NEM-VAGY rendszerben megvalósítható a NEM, az ÉS valamint a VAGY logikai függvény is.

A NEM logikai függvény kétféleképpen valósítható meg NEM-VAGY kapukkal. Az egyik esetben a VAGY kapcsolat idempotencia tulajdonságát lehet kihasználni, azaz $A + A = A$, a másik esetben pedig azt, hogy a 0 értékkel való VAGY kapcsolat eredménye csak a logikai változó értékétől függ, azaz $A + 0 = A$. Ezek alapján az azonosságok mind a két oldalának negálása eredményezi a bizonyítást. $\overline{A} = \overline{A + A} = \overline{A + 0}$. A két esethez tartozó logikai áramköri realizációk a következő ábrán láthatóak.



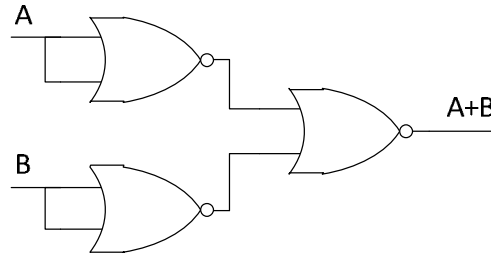
8.4 ábra: \overline{A} megvalósítása NEM-VAGY kapukkal

Ezek alapján a VAGY logikai kapcsolatot megvalósítása NEM-VAGY rendszerben egyszerűen meg lehet valósítani. Azt kell kihasználni, hogy egy logikai kifejezés kétszeres tagadása az eredeti logikai kifejezést eredményezi, azaz $\overline{\overline{A}} = A$. Ez alapján A és B logikai változók VAGY kapcsolata a következő módon állítható elő $A + B = \overline{\overline{A + B}}$. A logikai áramköri realizációban a NEM függvény megvalósításához az első eset lett felhasználva, de természetesen a másik is használható lett volna.



8.5 ábra: AB megvalósítása NEM-VAGY kapukkal

Az ÉS logikai kapcsolat megvalósításához a De-Morgan azonosságra van szükség, mely szerint $\overline{AB} = \overline{A} + \overline{B}$. Mind a két oldalt invertálása egy olyan logikai kifejezést eredményez, melynek bal oldalán ÉS kapcsolat, jobb oldalán pedig VAGY valamint NEM kapcsolat található, azaz $AB = \overline{\overline{A} + \overline{B}}$. Mivel VAGY valamint NEM kapcsolatot meg lehet valósítani NEM-VAGY rendszerben, ezért az ÉS kapcsolat is megvalósítható, melynek logikai áramköri realizációja a következő ábrán látható.

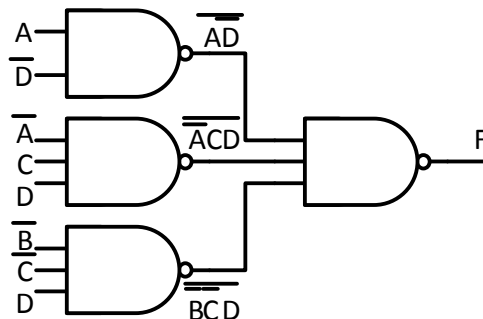


8.6 ábra: $A + B$ megvalósítása NEM-VAGY kapukkal

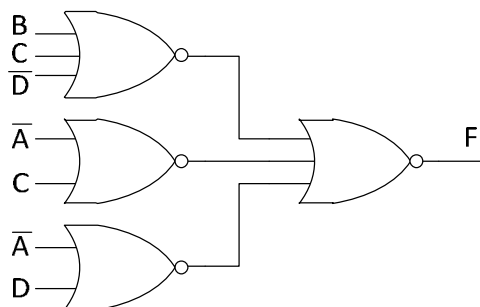
Kétszintű hálózat tervezése NEM-ÉS és NEM-VAGY rendszerben

A szisztematikus függvényegyszerősítő eljárások eredménye kétszintű ÉS-VAGY illetve VAGY-ÉS hálózatként valósítható meg. Ezek a hálózatok két logikai műveleti szinttel rendelkeznek, mivel a bemenő változók a kimenetre két kapun keresztül jutnak el. A szinteket a hálózat kimenetétől a bemenet felé haladva kell számozni. Kétszintű hálózat tervezése előtt meg kell vizsgálni, hogy egy kétszintű NEM-ÉS valamint egy kétszintű NEM-VAGY hálózat hogyan működik.

Először egy NEM-ÉS hálózat vizsgálatát kell elvégezni az alábbi példán keresztül, ahol a logikai függvény a logikai áramkörrel adott. A függvény algebrai alakját fel lehet írni a kapcsolás alapján, ahol $F = \overline{AD} \cdot \overline{ACD} \cdot \overline{BCD}$. De-Morgan azonosságot használva a legmagasabb szintű negálás eltüntethető, azaz $F = \overline{AD} + \overline{ACD} + \overline{BCD} = \overline{AD} + \overline{ACD} + \overline{BCD}$. Ez azt jelenti, hogy az első szinten lévő NEM-ÉS kapu VAGY kapcsolatot a második szinten lévő NEM-ÉS kapuk pedig ÉS kapcsolatot realizálnak. Azaz egy kétszintű NEM-ÉS hálózat egy diszjunktív normál formát ad.



Következő lépésben egy NEM-VAGY hálózat vizsgálatát kell elvégezni. Itt is egy logikai áramkör a kiindulási pont. A függvény algebrai alakját fel lehet írni a kapcsolás alapján, ahol $F = B + C + \overline{D} + \overline{A} + C + \overline{A} + D$. De-Morgan azonosságot használva a legmagasabb szintű negálás eltüntethető, azaz $F = \overline{\overline{B + C + \overline{D}} \cdot \overline{\overline{A} + C} \cdot \overline{\overline{A} + D}}$. Ez azt jelenti, hogy az első szinten lévő NEM-VAGY kapu ÉS kapcsolatot a második szinten lévő NEM-VAGY kapuk pedig VAGY kapcsolatot realizálnak. Azaz egy kétszintű NEM-VAGY hálózat egy konjunktív normál formát ad.



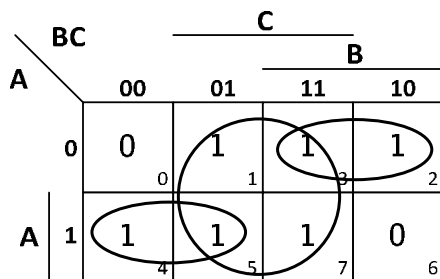
Ezek után a korábbi módszereket alkalmazva könnyen lehet kétszintű NEM-ÉS valamint NEM-VAGY hálózatokat tervezni. A tervezés lépései a következők:

1. Függvény felírása valamilyen formában.
2. Függvény egyszerűsítése, kétszintű hálózat tervezéséhez az ötödik fejezetben leírt valamelyik módszerrel.
3. Az egyszerűsített függvény alapján a kétszintű hálózat felrajzolása NÉV rendszerbeli elemekkel.
4. A hálózatban szereplő logikai kapuk átjelölése NEM-ÉS vagy NEM-VAGY kapukra, attól függően, hogy az egyszerűsített alak diszjunktív vagy konjunktív formában adott.

Ez a módszer kétszintű hálózatok tervezéséhez használható, ahol minden bemeneti jel két kapun megy át, mire a kimenethez ér. Ez a kijelentés magával vonja azt a feltételezést, hogy a bemenetek ponált és negált formában is adottak, azaz nincs külön inverter szint a bemenetek után.

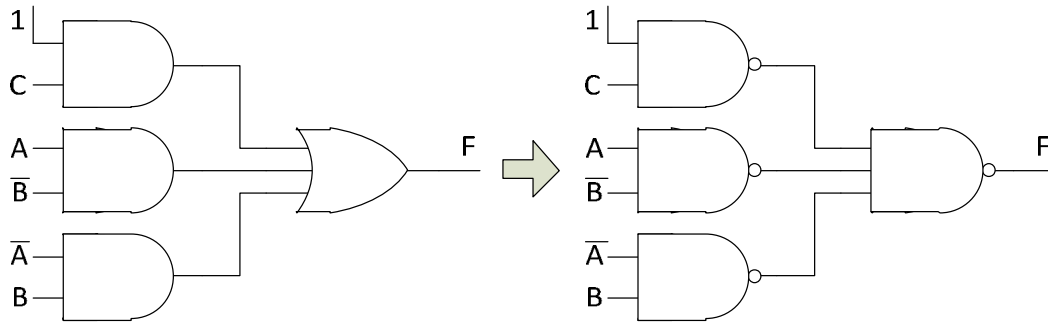
Legyen egy három bemenettel rendelkező hálózat az igazságtáblázatával adott. A Karnaugh táblából ki lehet olvasni a függvény egyszerűsített diszjunktív alakját, mely a következő $F = C + A\bar{B} + \bar{A}B$.

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



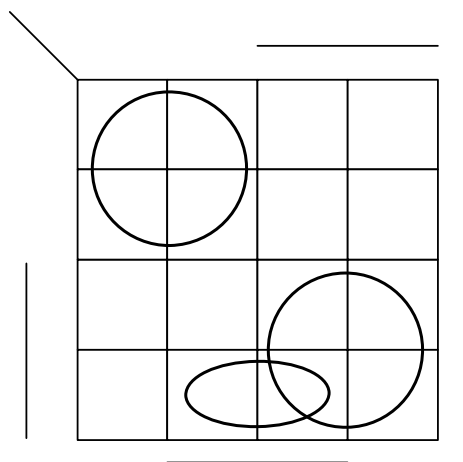
A függvény NEM-ÉS kapukkal megvalósított logikai áramköri realizációjához először szükség van a függvény NÉV rendszerbeli, kétszintű logikai áramköri realizációjára. Itt figyelni kell arra, hogy minden bemeneti jelnek kettő kapun kell áthaladnia a kimenetig. A C bemenetet közvetlenül is rá lehetne kötni a VAGY kapu bemenetére, de akkor a NEM-ÉS kapukra való átjelölés nem helyes megoldást eredményezne. A helyesen megvalósított NEM-ÉS hálózatban a C jel az első szinten lévő

NEM-ÉS kapura negált értékben érkezik meg szemben azzal, ha nem két kapun haladna át a kimenetig.

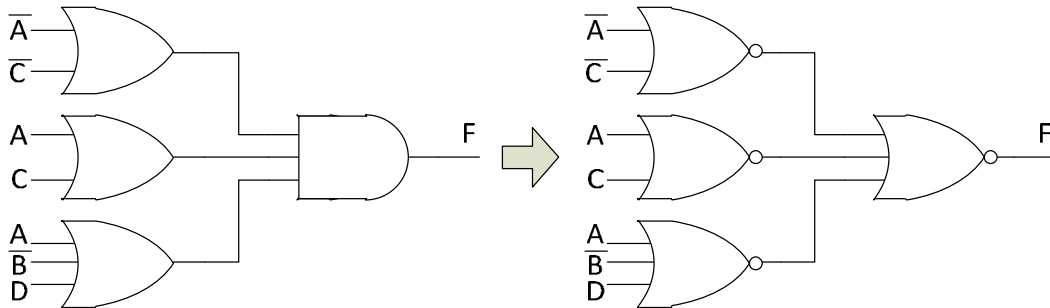


A NEM-VAGY hálózat tervezésnek bemutatásához legyen a négyváltozós logikai függvény az igazságtáblájával adott. A Karnaugh táblából ki lehet olvasni a függvény egyszerűsített diszjunktív alakját, mely a következő $F = (\bar{A} + \bar{C})(A + C)(A + \bar{B} + D)$.

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



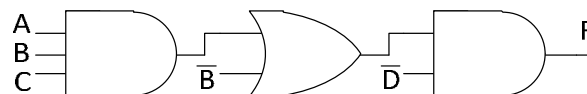
A függvény NEM-VAGY kapukkal megvalósított logikai áramköri realizációjához először szükség van a függvény NÉV rendszerbeli, kétszintű logikai áramköri realizációjára. A kapuk NEM-VAGY kapukra való átjelölésével lehet megkapni a megoldást.



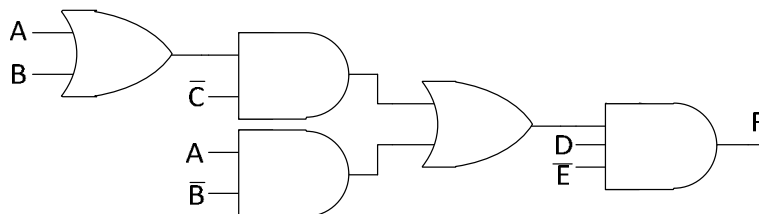
Többszintű hálózat tervezése NEM-ÉS és NEM-VAGY rendszerben

Az eddigi szisztematikus egyszerűsítési eljárások diszjunktív vagy konjunktív formája megoldást eredményeztek. Bármelyik típusú függvény összekapcsolása újabb logikai művelettel, egy vagy több másik függvénnyel, az új függvény már nem realizálható kétszintű hálózattal egyszerűsítés nélkül. Az ilyen típusú függvényeket megvalósító hálózatokat többszintű hálózatoknak nevezik.

Az $F = (ABC + \bar{B})\bar{D}$ függvényt háromszintű hálózattal lehet megvalósítani. A harmadik szinten az ABC ÉS kapcsolatot, a második szinten az $ABC + \bar{B}$ VAGY kapcsolatot, az első szinten pedig az $(ABC + \bar{B})\bar{D}$ ÉS kapcsolatot valósítja meg.



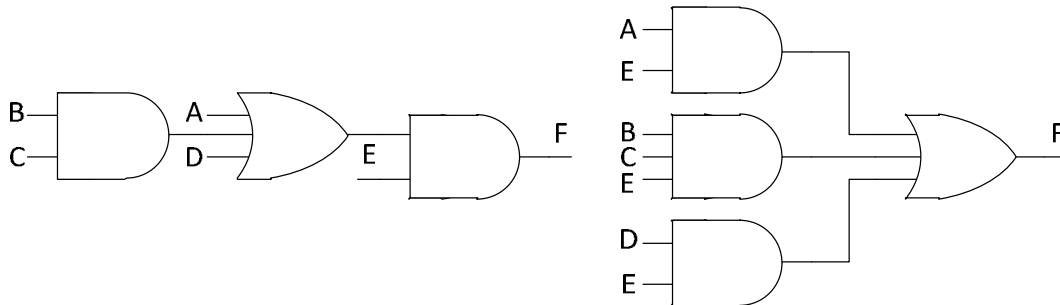
Az $F = ((A + B)\bar{C} + A\bar{B})D\bar{E}$ függvényt négy szintű hálózattal lehet megvalósítani. A negyedik szinten az $A + B$ VAGY kapcsolatot, a harmadik szinten az $(A + B)\bar{C}$ és $A\bar{B}$ ÉS kapcsolatokat, a második szinten az $(A + B)\bar{C} + A\bar{B}$ VAGY kapcsolatot, az első szinten pedig az $((A + B)\bar{C} + A\bar{B})D\bar{E}$ ÉS kapcsolatot valósítja meg.



A példából látszik, hogy a szintek száma a függvény algebrai alakjából meghatározható az utolsónak elvégzendő logikai művelettől kezdve a zárójeleken befelé haladva, megszámlolván az egymás után elvégzendő (azonos logikai szinteket jelentő) műveleteket.

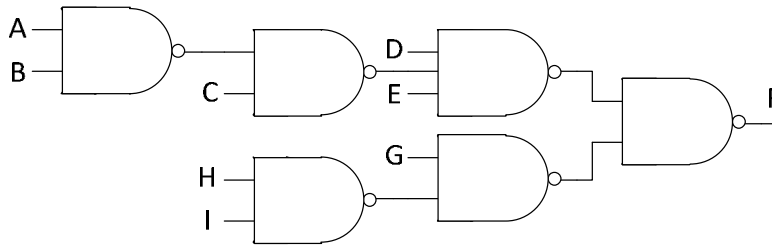
Az eddigiekben bemutatott módszerek segítségével kétszintű hálózatok tervezhetők bármilyen logikai függvényhez. Mivel minden logikai függvény felírható diszjunktív vagy konjunktív alakban, ezért két szinttel rendelkező hálózatként megvalósíthatók. A gyakorlatban viszont nem mindig célszerű kétszintű hálózatokat alkalmazni, mert a diszjunktív és a konjunktív alak bonyolultabb lehet, mint egy többszintű hálózatot leíró függvény.

Legyen $F = (A + BC + D)E$ egy ötváltozós logikai függvény, melynek egyszerűsített disztributív alakja $F = AE + BCE + DE$. Megvizsgálva a két logikai áramköri realizációt megállapítható, hogy a többszintű hálózat három kapuáramkörrel és hét kapubemenettel, míg a kétszintű hálózat négy kapuáramkörrel és tíz kapubemenettel valósítható meg, azaz a többszintű hálózat egyszerűbb megvalósítást eredményez.



Többszintű NEM-ÉS hálózatok

A többszintű NEM-ÉS hálózatok tervezéséhez meg kell vizsgálni annak működését. Legyen a kiinduló hálózat egy többszintű kizárólag NEM-ÉS kapukat tartalmazó logikai áramkör. Ennek az áramkörnek a kimeneti függvényét fel lehet írni a kapcsolási rajz alapján, azaz $F = \overline{\overline{\overline{DABCEGHI}}}$. A függvényt át lehet alakítani De-Morgan azonosság segítségével úgy, hogy legfeljebb változók legyenek negálva, kifejezések ne, azaz $F = \overline{\overline{DABCE}} + \overline{GHI} = D(\overline{A} + \overline{B} + \overline{C})E + G(\overline{H} + \overline{I})$.



A vizsgálat a NEM-ÉS hálózatok két jellegzetességére mutat rá.

- A hálózat páratlan szintjein VAGY, a páros szinteken ÉS logikai kapcsolat valósul meg.
- A páros szinteken bevezetett változók értékei ponáltan, a páratlan szinteken bevezetettek negáltan jelennek meg a kimeneten.

Ezen tulajdonságok alapján felrajzolható többszintű NEM-ÉS hálózat minden olyan logikai függvényhez, melynek legfelső szintjén VAGY kapcsolat van, és melyben kizárólag változók vannak negálva összetett kifejezések nem.

Legyen $F = (\overline{A} + C)(B + A\overline{D}) + B(C\overline{D} + A)\overline{G} + E(\overline{B} + A(\overline{C} + G))$ logikai függvény, melyet többszintű NEM-ÉS hálózattal kell megvalósítani. A függvényre teljesülnek a feltételek, azaz a legfelső szinten VAGY kapcsolat van és kizárólag logikai változók vannak negálva benne. Először a műveleti szinteket kell megállapítani.

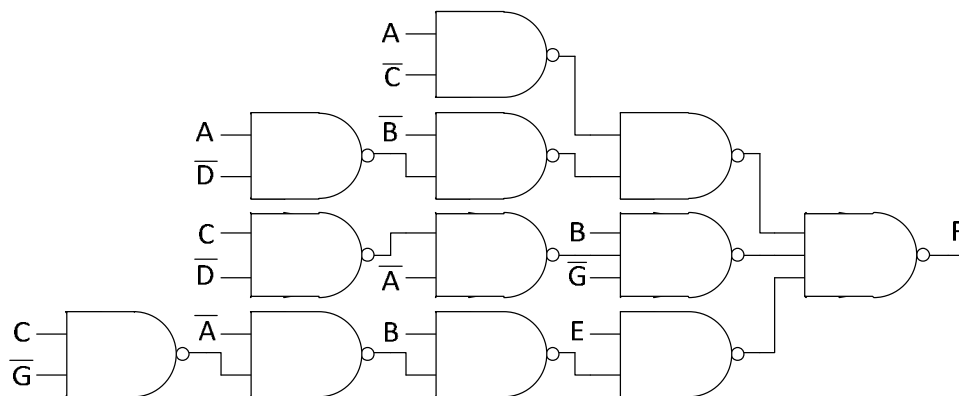
- Az első szinten VAGY kapcsolat van a következő kifejezések között.
 - $(\overline{A} + C)(B + A\overline{D})$, $B(C\overline{D} + A)\overline{G}$ és $E(\overline{B} + A(\overline{C} + G))$
- A második szinten ÉS kapcsolat van a következő kifejezések között.
 - $\overline{A} + C$ és $B + A\overline{D}$
 - B , $C\overline{D} + A$ és \overline{G}

- E és $\bar{B} + A(\bar{C} + G)$
- A harmadik szinten VAGY kapcsolata van a következő kifejezések között.
 - \bar{A} és C
 - B és $A\bar{D}$
 - $C\bar{D}$ és A
 - \bar{B} és $A(\bar{C} + G)$
- A negyedik szinten ÉS kapcsolat van a következő kifejezések között.
 - A és \bar{D}
 - C és \bar{D}
 - A és $\bar{C} + G$
- Az ötödik szinten VAGY kapcsolata van a következő kifejezések között.
 - \bar{C} és G

A NEM-ÉS hálózatoknak megfelelően páratlan szinteken VAGY kapcsolat, páros szinteken ÉS kapcsolat van. Ezek után meg kell vizsgálni, hogy melyik szinten milyen változó kerül be a függvénybe.

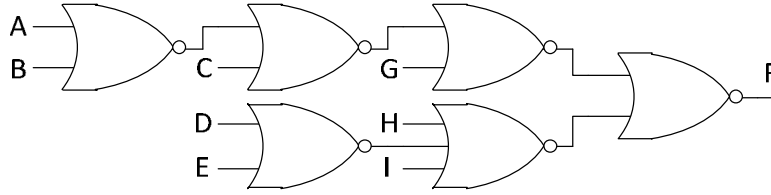
- Az első szinten nem kerül be változó
- A második szinten B , \bar{G} és E változók kerülnek be. Ez páros szint, ezért ezek ilyen formában kerülnek a kapuk kimenetére.
- A harmadik szinten \bar{A} , C , B , A és \bar{B} változók kerülnek be. Ez páratlan szint, ezért ezek ehhez képest negált formában kerülnek a kapuk kimenetére, azaz rendre A , \bar{C} , \bar{B} , \bar{A} és B .
- A negyedik szinten A , \bar{D} , C , \bar{D} és A változók kerülnek be. Ez páros szint, ezért ezek ilyen formában kerülnek a kapuk kimenetére.
- Az ötödik szinten \bar{C} és G változók kerülnek be. Ez páratlan szint, ezért ezek ehhez képest negált formában kerülnek a kapuk kimenetére, azaz rendre C és \bar{G} .

A vizsgálatok az alábbi logikai áramköri realizációt eredményezik.



Többszintű NEM-VAGY hálózatok

A többszintű NEM-VAGY hálózatok tervezéséhez meg kell vizsgálni annak működését. Legyen a kiinduló hálózat egy többszintű kizárólag NEM-VAGY kapukat tartalmazó logikai áramkör. Ennek az áramkörnek a kimeneti függvényét fel lehet írni a kapcsolási rajz alapján, azaz $F = \overline{\overline{\overline{A + B + C + G + H + \bar{D} + \bar{E} + I}}$. A függvényt át lehet alakítani De-Morgan azonosság segítségével úgy, hogy legfeljebb változók legyenek negálva, kifejezések ne, azaz $F = \overline{(\bar{A} + \bar{B} + \bar{C} + \bar{G})(H + \bar{D} + \bar{E} + I)} = ((A + B)\bar{C} + G)(H + \bar{D}\bar{E} + I)$.



A vizsgálat a NEM-VAGY hálózatok két jellegzetességére mutat rá.

- A hálózat páratlan szintjein ÉS, a páros szinteken VAGY logikai kapcsolat valósul meg, pont fordítva, mint NEM-ÉS hálózatoknál.
- A páros szinteken bevezetett változók értékei ponáltan, a páratlan szinteken bevezetettek negáltan jelennek meg a kimeneten, ugyanúgy, mint NEM-ÉS hálózatoknál.

Ezen tulajdonságok alapján felrajzolható többszintű NEM-VAGY hálózat minden olyan logikai függvényhez, melynek legfelső szintjén ÉS kapcsolat van, és melyben kizárólag változók vannak negálva összetett kifejezések nem.

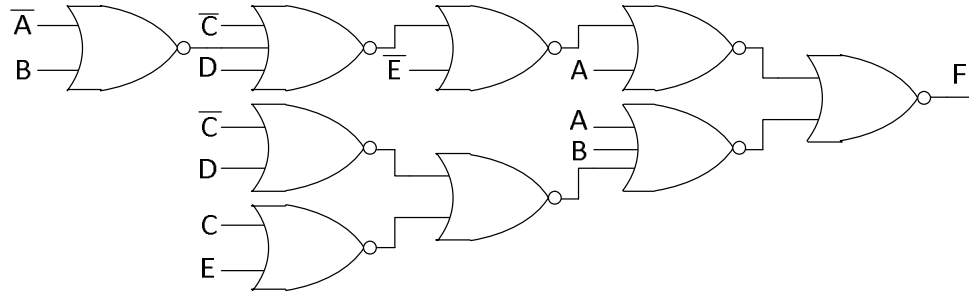
Legyen $F = ((\bar{C} + A\bar{B} + D)E + A)(A + B + (\bar{C} + D)(C + E))$ logikai függvény, melyet többszintű NEM-VAGY hálózattal kell megvalósítani. A függvényre teljesülnek a feltételek, azaz a legfelső szinten ÉS kapcsolat van és kizárólag logikai változók vannak negálva benne. Először a műveleti szinteket kell megállapítani.

- Az első szinten ÉS kapcsolat van a következő kifejezések között.
 - $(\bar{C} + A\bar{B} + D)E + A$ és $A + B + (\bar{C} + D)(C + E)$
- A második szinten VAGY kapcsolat van a következő kifejezések között.
 - $(\bar{C} + A\bar{B} + D)E$ és A
 - A, B és $(\bar{C} + D)(C + E)$
- A harmadik szinten ÉS kapcsolata van a következő kifejezések között.
 - $\bar{C} + A\bar{B} + D$ és E
 - $\bar{C} + D$ és $C + E$
- A negyedik szinten VAGY kapcsolat van a következő kifejezések között.
 - $\bar{C}, A\bar{B}$ és D
 - \bar{C} és D
 - C és E
- Az ötödik szinten ÉS kapcsolata van a következő kifejezések között.
 - A és \bar{B}

A NEM-VAGY hálózatoknak megfelelően páratlan szinteken VAGY kapcsolat, páros szinteken ÉS kapcsolat van. Ezek után meg kell vizsgálni, hogy melyik szinten milyen változó kerül be a függvénybe.

- Az első szinten nem kerül be változó
- A második szinten A, A és B változók kerülnek be. Ez páros szint, ezért ezek ilyen formában kerülnek a kapuk kimenetére.
- A harmadik szinten E változó kerül be. Ez páratlan szint, ezért ezek ehhez képest negált formában kerül a kapuk kimenetére, azaz \bar{E} .
- A negyedik szinten $\bar{C}, D, \bar{C}, D, C$ és E változók kerülnek be. Ez páros szint, ezért ezek ilyen formában kerülnek a kapuk kimenetére.
- Az ötödik szinten A és \bar{B} változók kerülnek be. Ez páratlan szint, ezért ezek ehhez képest negált formában kerülnek a kapuk kimenetére, azaz rendre \bar{A} és B .

A vizsgálatok az alábbi logikai áramköri realizációt eredményezik.



9. Leggyakrabban használt kombinációs hálózatok

A kombinációs hálózatokkal megoldható logikai feladatok közül jó néhány annyira elterjedt a gyakorlatban is, hogy ezeket az áramkörtípusokat integrált kivitelben, mint funkcionális áramkörök is gyártják. A leggyakrabban használt kombinációs hálózatok típusai a következők:

- kódolók
- dekódolók
- multiplexerek
- demultiplexerek
- komparátorok
- összeadók
- paritásvizsgáló áramkörök

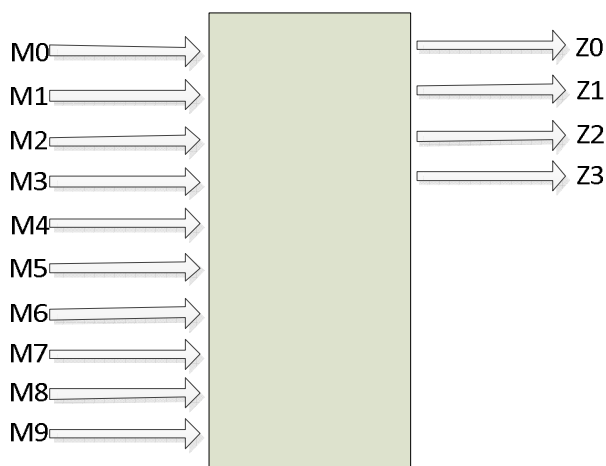
Most ezek részletesebb vizsgálatával foglalkozunk.

Kódolók

A kódolók olyan áramkörök, amelynek bármelyik 1 az m -ből bemenetének az aktiválása esetén egy k bites kódot szolgáltatnak.

Példa decimális-BCD kódoló áramkörre:

	Z3	Z2	Z1	Z0
M0	0	0	0	0
M1	0	0	0	1
M2	0	0	1	0
M3	0	0	1	1
M4	0	1	0	0
M5	0	1	0	1
M6	0	1	1	0
M7	0	1	1	1
M8	1	0	0	0
M9	1	0	0	1



Az igazságtáblázat alapján felírhatók az egyes kimeneteket működtető logikai függvények:

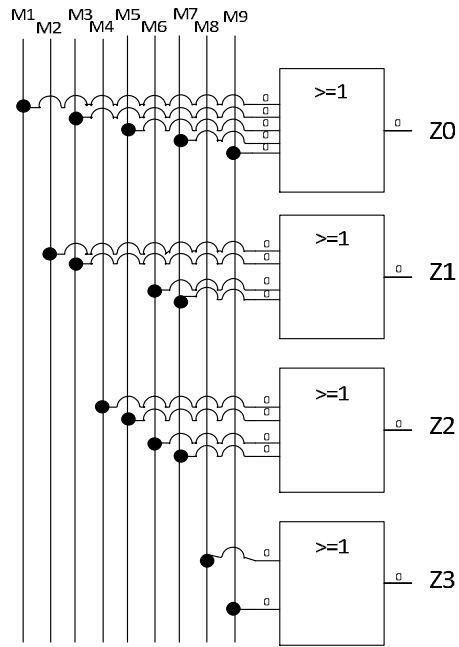
$$Z_0 = M_1 + M_3 + M_5 + M_7 + M_9$$

$$Z_2 = M_4 + M_5 + M_6 + M_7$$

$$Z_1 = M_2 + M_3 + M_6 + M_7$$

$$Z_3 = M_8 + M_9$$

A decimális-BCD kódoló megvalósító logikai függvények realizálása, a kódoló áramkör megvalósítása:



9.1 ábra: Decimális-BCD kódoló

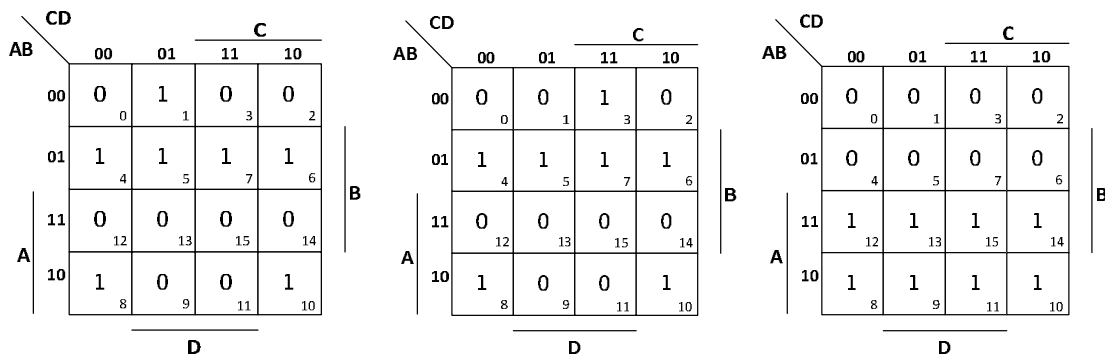
Az áramköri megoldás hátránya, hogy ha egyidejűleg több bemenet válik aktívvá, a kimenetek állapota nem egyértelműen meghatározható. Erre adnak megoldást a prioritáskódoló áramkörök.

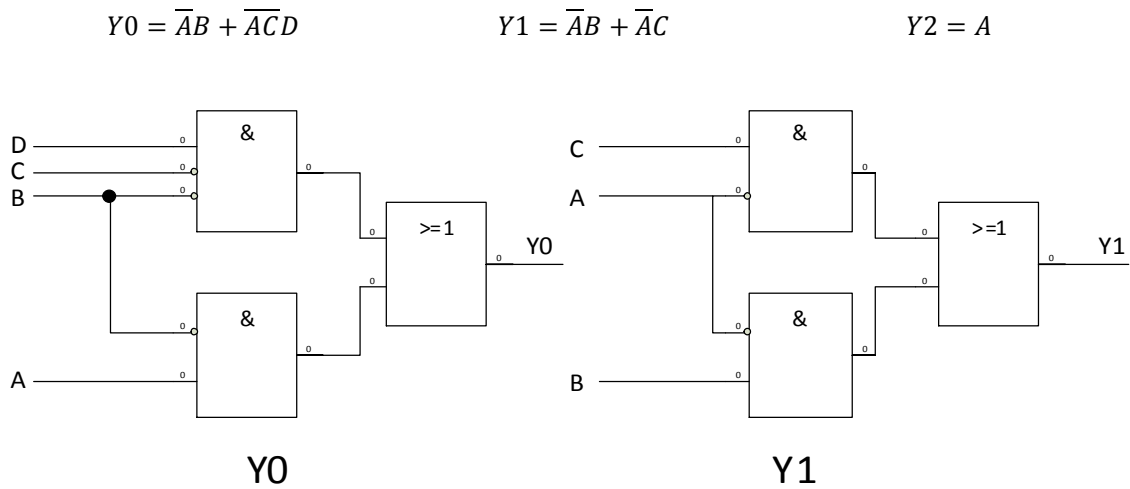
A kódoló áramkörök egy fontos és speciális területe az ún. prioritáskódoló áramkörök alkalmazása. Például billentyűzet kódolók alkalmazásaként. Az előbb ismertetett kódoló áramkörtől annyiban térnek el, hogy egyidejűleg több bemenet aktiválása esetén is csak a nagyobb prioritású bemenet kódját adja a kimenetre az áramkör.

Példaként az alábbi táblázatban 4-ből 3-ba prioritáskódoló igazságtáblázata látható:

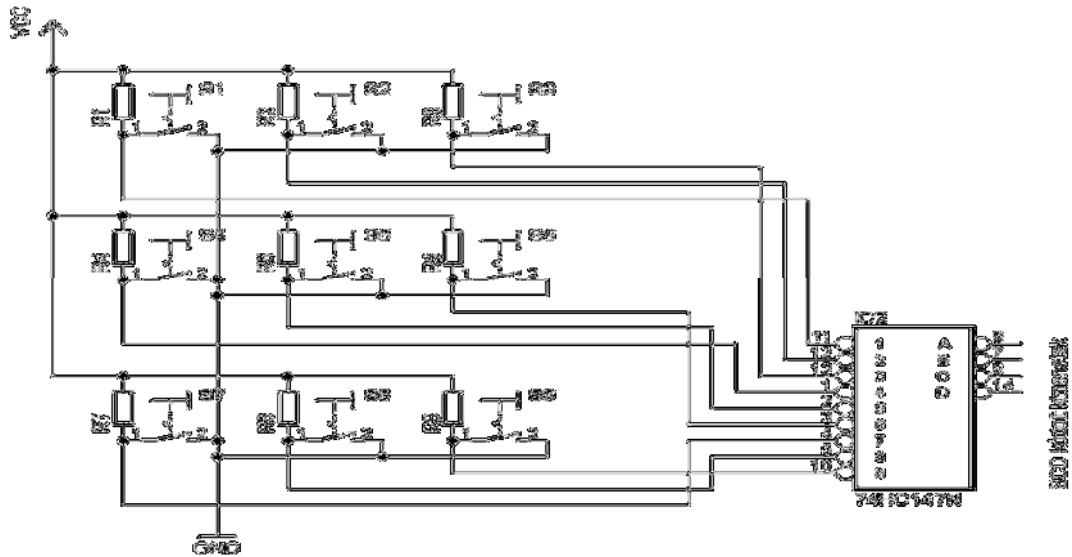
A	B	C	D	Y2	Y1	Y0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	0
0	1	X	X	0	1	1
1	X	X	X	1	0	0

Alább az igazságtáblázat alapján kitöltött KV (Karnaugh-Veitch) táblázatok és az egyszerűsítés után kapott logikai függvények:





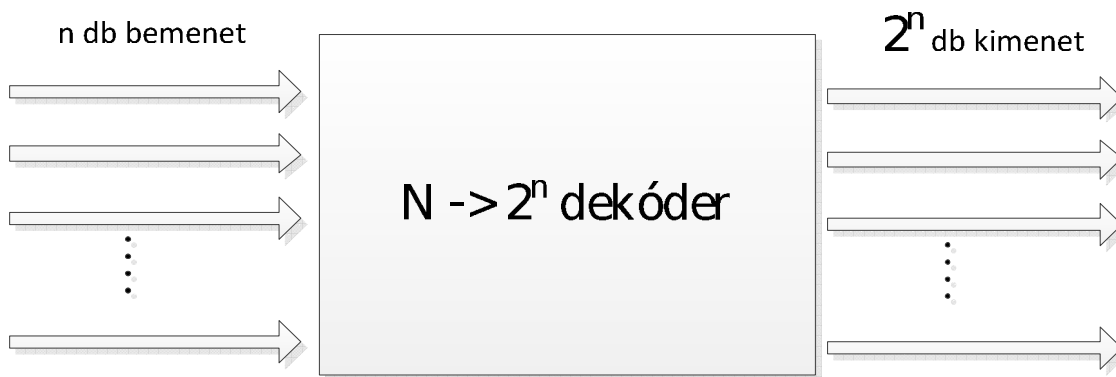
A prioritáskódoló áramkör megvalósított áramköri rajzai. Természetesen Y2-t nem kell logikai áramkörökkel realizálni.



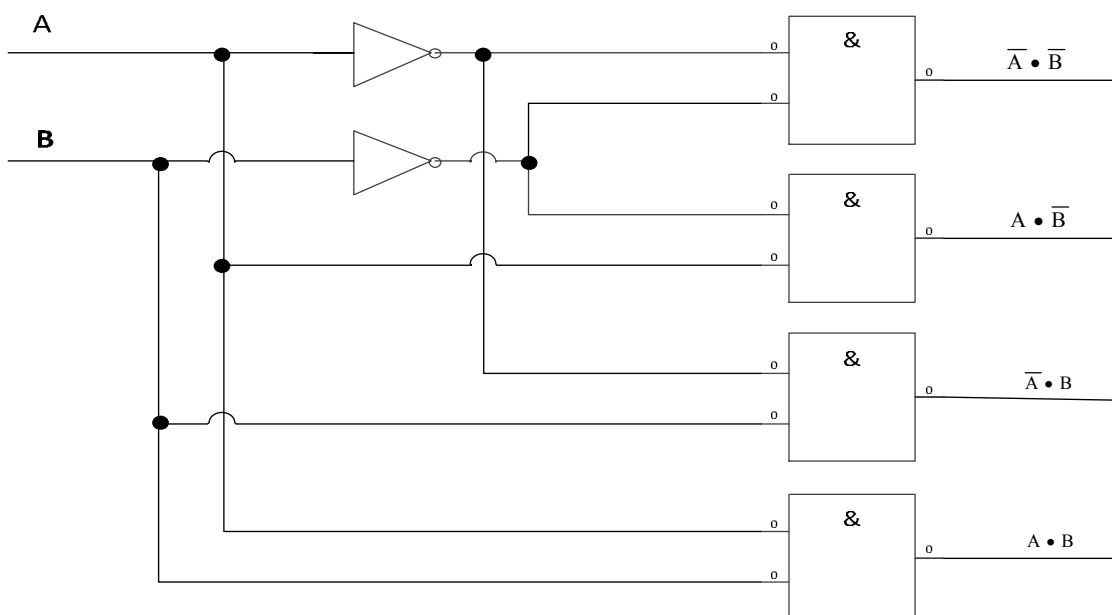
9.2 ábra: Billentyűzetkódoló áramkör SN74147 típusú prioritáskódoló áramkör alkalmazásával.

Dekódolók

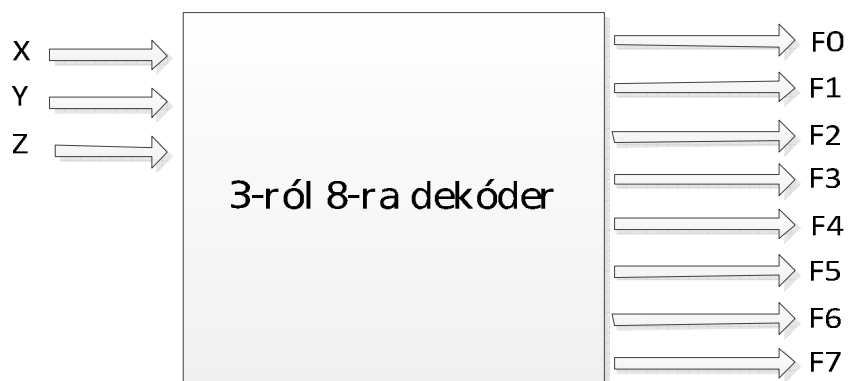
A dekódoló áramkörök a kódolók fordítottjának is tekinthetők. Az áramkörök bemenetére adott kód alapján egyetlen kimenet válik aktívvá.



Az alábbi ábrán egy bináris 2-ről 4-re dekódoló kapcsolási rajza látható. Az áramkör egyszerűsége miatt, a működést könnyen megérthetjük igazságtáblázat és függvények nélkül is. A kimeneti értékeket előállító kombinációs hálózat a bináris számképzés szabályai szerint működik.

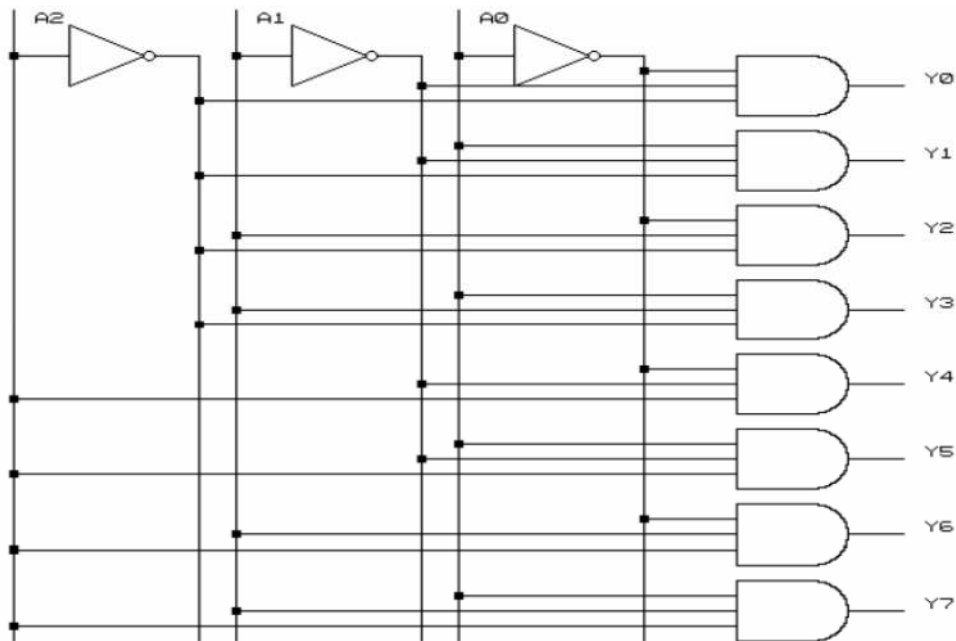


Az alábbi ábrán egy kicsivel bonyolultabb felépítésű 3-ról 8-ra dekódoló áramkör elvi rajza látható. Az áramkör három bemenettel és nyolc kimenettel rendelkezik.



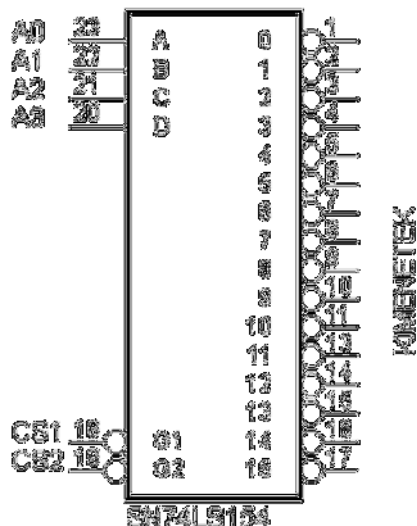
A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

A kitöltött igazságtáblázat és az előző dekóder működési vázlatára alapján elkészíthetjük az áramköri rajzot.

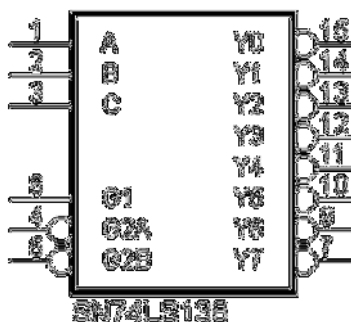


9.3 ábra: 3-ról 8-ra dekódoló kapcsolási rajza

4-ről 16-ra dekódoló áramkör 74HC154. Az áramkör 4 db úgynevezett kiválasztó vagy címző bemenettel rendelkezik és kettő darab engedélyező bemenete van. Ezek alacsony aktív bemenetek és logikai ÉS kapcsolatban vannak egymással. A kimenetek is alacsony aktív szintűek, ami azt jelenti, hogy a kiválasztott kimenet lesz alacsony logikai szintű, míg a többi magas.



Példa: 74 LS138 3-ról 8-ra dekódoló, engedélyező bemenetekkel.



G1, G2A és G2B engedélyező bemenetek logikai ÉS kapcsolatban állnak egymással, valamint G2A és G2B negáltak. Legyen $EN = G1 \cdot \overline{G2A} \cdot \overline{G2B}$

Igazságtáblázat:

A	B	C	EN	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	1	0	1	1	1	1	1	1
0	1	0	1	1	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0	1
X	X	X	0	1	1	1	1	1	1	1	0

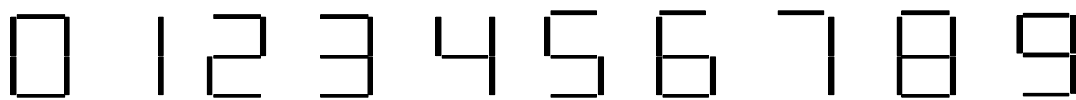
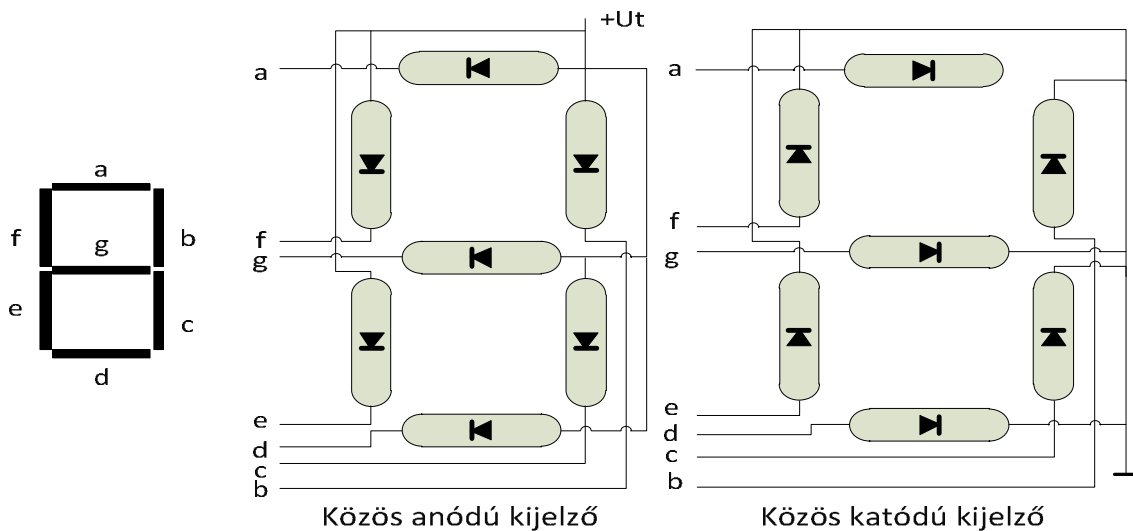
A kimenetek alacsony aktívak. Ezt az áramkört gyakran alkalmazzák úgynevezett címdekóder áramkörnek is, mivel a bemeneteire adott bináris értéknek megfelelő kimenete lesz alacsony aktív szintű, ami illeszkedik a legtöbb memória és periféria áramkör kiválasztó bemenetéhez.

7 szegmenses kijelző dekóder-meghajtó áramkörök

A dekóderek egy speciális felhasználási területe a 7 szegmenses számkijelzők meghajtásához szükséges jelek előállítására. A dekódernek 4 bemenete van, mivel a kijelző vezérlő jelei BCD (Binary

Coded Decimal) kódban érkeznek az áramkör bementére. Így ezeket a dekódereket 4-ből 7-re dekódernek is lehet nevezni.

A hétszegmentes kijelző ábráját, a szegmensek jelölését és kivezetéseit a következő ábra mutatja:



A következő példa egy úgynevezett közös anódos kijelző, ahol a szegmenseket megvilágító LED diódák anódjai vannak közösítve. Ebben az esetben, ahogy az ábrán is látszik, a közös anód kapcsolódik a tápfeszültségre és a megfelelő LED bekapcsolásához a LED katódját földre kell kapcsolni, természetesen egy megfelelő nagyságú áramkorlátozó ellenálláson keresztül. Az ilyen típusú kijelzők meghajtásához azok az áramkörök alkalmasak, melyek alacsony aktív kimenetűek. Ezt a kimenetekre rajzolt karika szimbólumok jelzik.

A 7 szegmenses kijelzőknek létezik olyan típusa is, ahol a LED-ek fordított irányban vannak bekötve. Ezeket a kijelzőket közös katódos kijelzőknek nevezzük. Meghajtásukhoz természetesen magas aktív kimenettel rendelkező dekóder áramkörre van szükség. A kijelző szegmenseit az ábrán látható elrendezésben az ABC kis betűivel jelölik a-tól -g-ig.

A kijelző működtetéséhez mind a hét szegmensset megfelelően vezérlő logikai hálózatot kell tervezni. Tehát 7 db logikai függvényt kell megvalósítani, egyenként mind a hét szegmensnek.

Példaként nézzük meg az „a” szegmensset meghajtó logikai hálózat tervezési lépéseit, egykörös katódos kijelző esetén.

Első lépésként a működésnek megfelelő igazságtáblázatot kell kitölteni. A táblázat kitöltésekor a 9-nél nagyobb bemeneti értékek esetén a kimeneteket közömbös (don't care term) állapotúnak vettük figyelembe. A táblázat mind a hét szegmens állapotait tartalmazza.

Decimális érték	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1

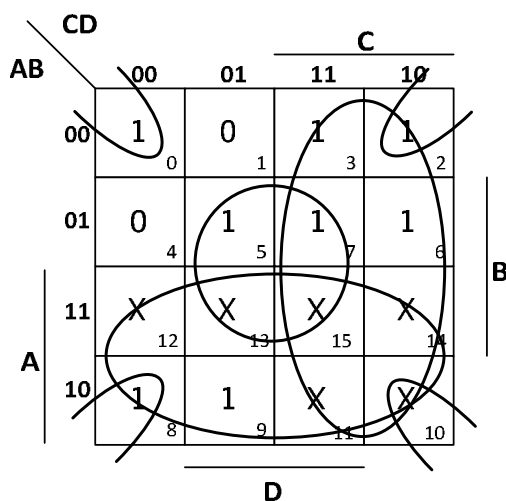
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	-	-	-	-	-	-	-
11	1	0	1	1	-	-	-	-	-	-	-
12	1	1	0	0	-	-	-	-	-	-	-
13	1	1	0	1	-	-	-	-	-	-	-
14	1	1	1	0	-	-	-	-	-	-	-
15	1	1	1	1	-	-	-	-	-	-	-

Amennyiben olyan kijelző meghajtó áramkört szeretnénk tervezni, amelyik a 16-os, hexadecimális karakterek megjelenítésére is képes, akkor ezt az igazságtáblázat megfelelő sorainak kitöltésével kell figyelembe venni.

A kitöltött „a” oszlop alapján felírhatjuk a szegmenset működtető logikai függvény kanonikus alakját:

$$a = \overline{A}BCD + \overline{A}BC\overline{D} + AB\overline{C}D + \overline{A}BCD + \overline{A}BC\overline{D} + ABC\overline{D} + ABCD + \overline{A}BCD + \overline{A}BC\overline{D}$$

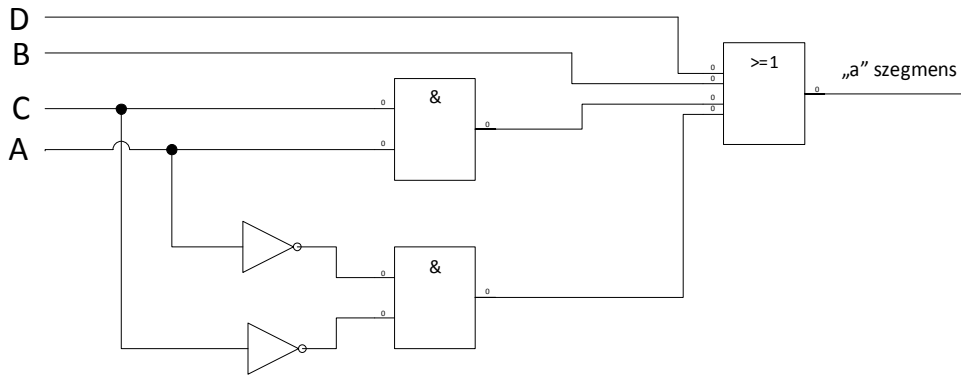
A legegyszerűbb alak előállításához a Karnaugh táblázatos grafikus egyszerűsítéssel kaphatjuk meg az egyszerűsített logikai függvényt.



Az egyszerűsítés után kapott logikai függvény:

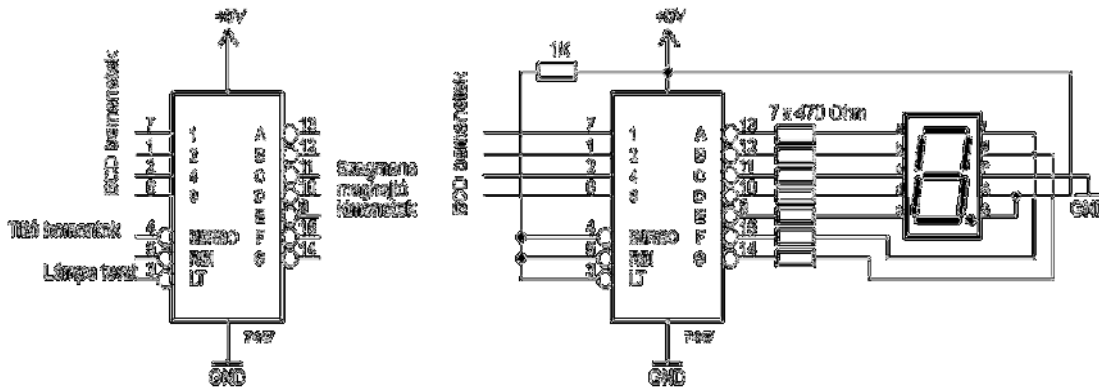
$$f_a = A + C + BD + \overline{B}D$$

Az „a” szegmens működtetését végző egyszerűsített logikai függvény realizációja az alábbi ábrán látható.



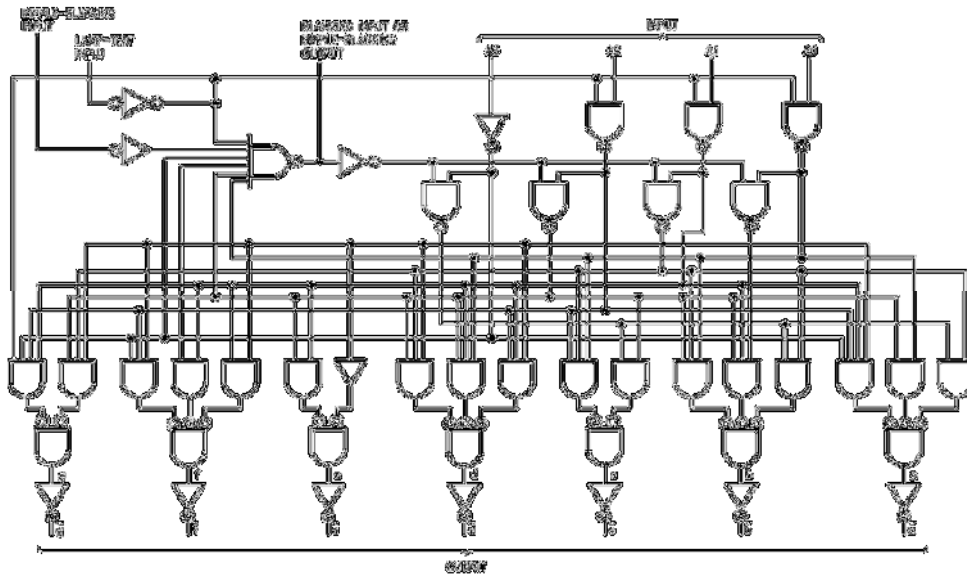
Ugyanezeket a lépéseket elvégezve kaphatjuk meg a többi szegmens vezérlő jelét előállító logikai hálózat függvényeit, amelyek alapján felrajzolhatjuk a végleges áramköri megvalósítás rajzait. Ha a kijelző dekóder logika hálózatát a példától eltérő módon szeretnénk megvalósítani, - például közös anódos kijelzővel, vagy hexadecimális kijelzéssel – akkor az igazságtáblázat megfelelő módosítása után, a már megismert lépéseket kell újra végrehajtani.

Az ábrán a működést megvalósító, kereskedelmi forgalomban készen kapható 74LS74 típusú integrált áramkör rajzjele és a kijelző bekötése látható.



Az áramkör a 4 db vezérlőjel és a kimeneteken kívül rendelkezik még a következő bemeneti jelekkel. LAMP TEST (LT), az a bemenet alacsony aktív és az összes szegmens bekapcsolására szolgál tesztelési célból. BLANKING INPUT (RBI) bemenet szolgál több kijelző kaszkádosításakor a megfelelő digitek fényének kioltására, ha a kijelző a sor elején vagy végén helyezkedik el és „0” értéket mutat. A (BI/RBO) láb bemenet és kimenet is lehet BLANKING INPUT/RIPPLE BLANKING OUTPUT. Ez a láb szolgáltatja a kimenetet a következő szegmens számára.

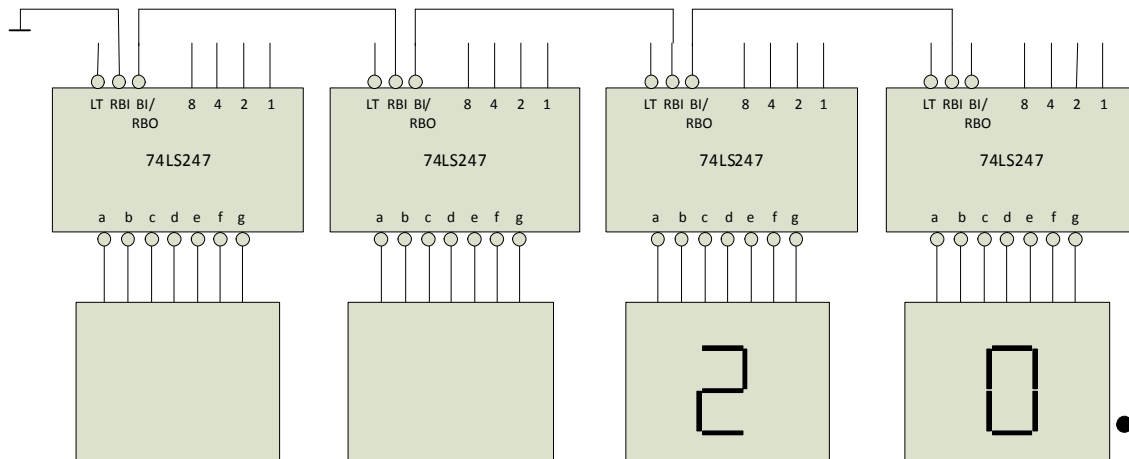
A teljes áramkör kapcsolási rajza a gyártó katalóguslapja alapján a következő ábrán látható [SN7447].



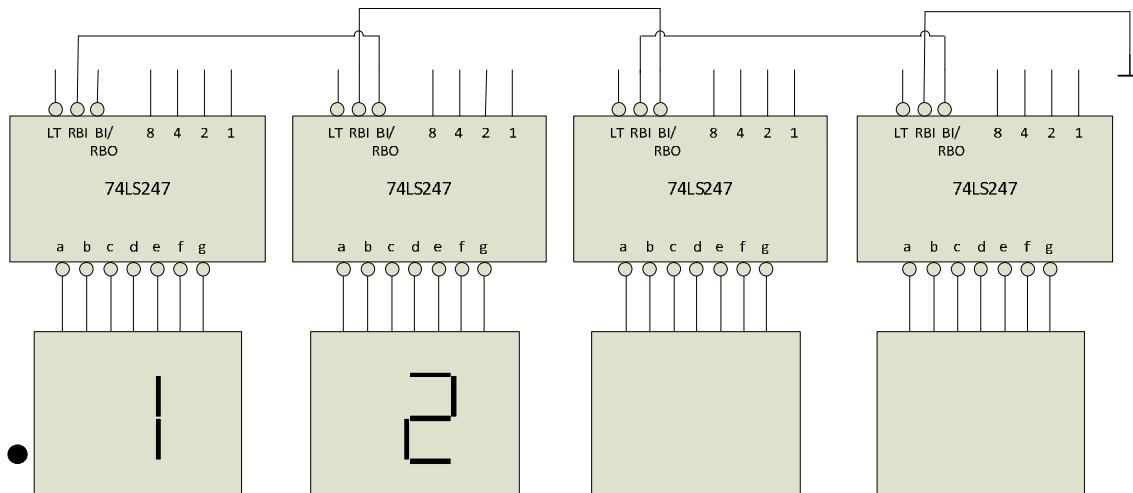
A logikai hálózat által megvalósított jelzéseképek a következők:



A következő ábra a kijelző meghajtó áramkör egy konkrét alkalmazását mutatja. A megfelelő vezérlő jelek összekötésével az áramkör a számlánc elején megjelenő „0” értéket nem jeleníti meg.



A következő kapcsolási elrendezés esetén a decimális pont utáni digitek közül nem világítanak a szám végén előforduló „0” értékek.



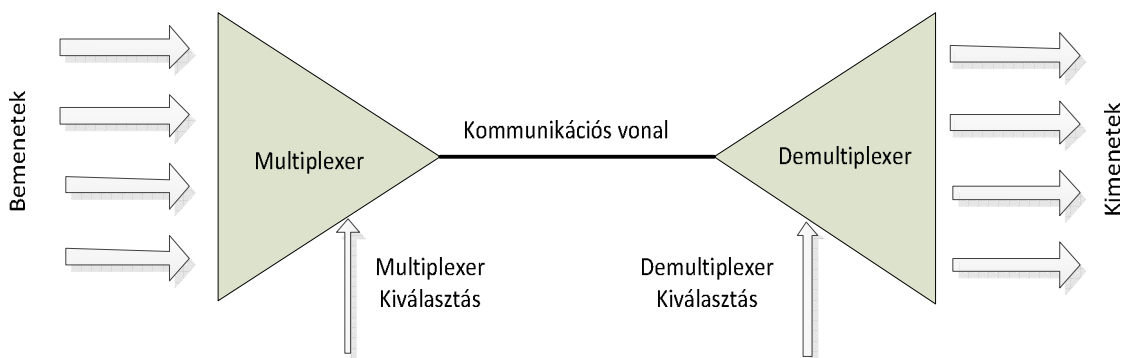
Multiplexerek – demultiplexerek

A logikai függvények realizálásának egyik másik lehetősége a digitális multiplexerek és demultiplexerek használata. Az áramköri egység vázlatát a következő ábrán láthatjuk.

A multiplexer nem képes a jeleket egyesíteni, mert a több bemenetből egyet kiválaszt és a kimenetre adja. Működési elve korai telefonközpontokhoz hasonlítható, maga a multiplexer a telefonos kisasszony volt.

A multiplexerek (MUX) feladata, hogy n számú adatból egyet kiválasszon és azt az egyetlen adatkimenetére továbbítsa. A MUX-nak mindig csak egy adatbemenete lehet aktív a kiválasztásnak megfelelően. A kiválasztás (címezés) a szelekt bemenettel történik. A demultiplexerek olyan kombinációs hálózatok, amelyek a „D” adatbemenet tartalmát az kiválasztó bemenet által kijelölt „Y” kimenetre juttatják.

A demultiplexerek feladata az adatok szétoosztása. A bemeneten lévő adat a címbemenetek által meghatározott sorszámú kimenetre kerül. A demultiplexernek egyszerre csak egy kimeneti vonalán lehet jel. A demultiplexernél a megcímezett kimenet nem aktív lesz, hanem felveszi a bemenet értékét.



A bemeneti oldalon látható multiplexer egység bemenetválasztóként működik. A kiválasztó (select) bemenetekkel meghatározott bemenet jelét a multiplexer kimenetére kapcsolja. A vételi oldalon szükség van egy olyan egységre, amely elvégzi a visszaalakítás műveletét, ez a **demultiplexer** vagy *demux*. Az ábra jobb oldalán látható demultiplexer áramkör a multiplexerrel szinkronban működtetve a bemenő jelét a megfelelő kimenetére kapcsolja. Így továbbítható például nagyszámú jel időben eltolva egyetlen vezetéken.

Ilyen funkciók lehetnek még a zajszűrés, jelhelyreállítás stb. A demultiplexáláshoz szükség van ugyanarra a vezérlő jelre, amit a multiplexer is használt. Ez a vezérlő jel mondja meg, hogy melyik bemenetet használja.

Egy példa a multiplexerre: a telefonközpont. A telefonközpont a hívót és hívottat összekötve egy időbeli multiplexálást végez. Szintén multiplexer egy műsorszóró műhold.

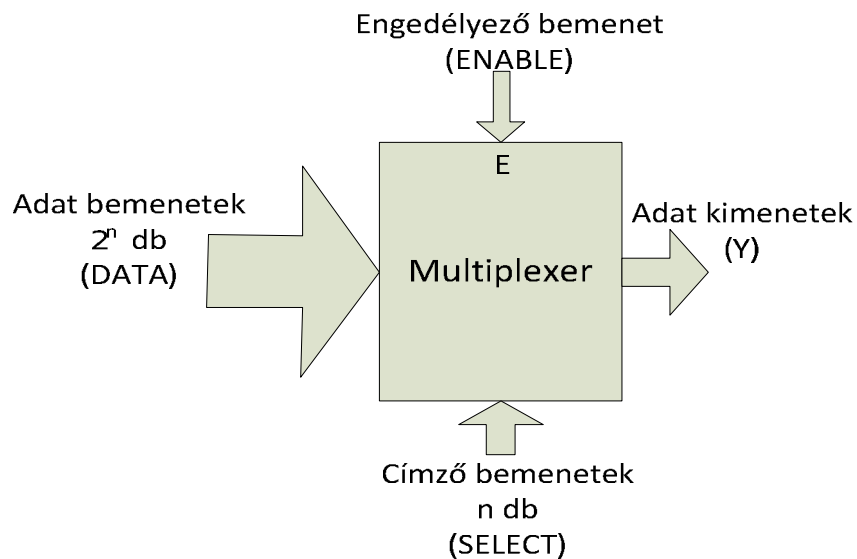
Multiplexerek

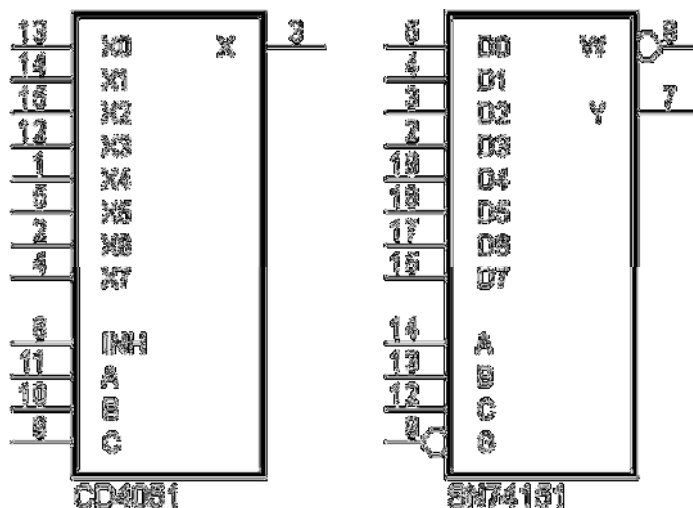
A multiplexereket, működési módjuk alapján a következő típusokba sorolják:

- időosztásos multiplexerek
- frekvenciaosztásos multiplexerek
- hullámhossz-osztásos multiplexerek
- kód-osztásos multiplexerek

Időosztásos multiplexálás esetén a bemeneti jelek felváltva egyetlen közös kimenetre kapcsolódnak. Ilyen például az ábrán látható multiplexer forgó kapcsolóval történő megvalósítása, vagy egy elektromechanikus telefonközpont. Frekvenciaosztásos multiplexer a bemeneti jeleket egy időben továbbítja ugyanazon az átviteli úton, azok az elfoglalt frekvencia tartomány alapján különíthetők el egymástól, mint például az analóg kábeltv rendszer. Hullámhossz-osztásos multiplexer például az optikai átvitel esetében használatos (lényegét tekintve ez is frekvenciaosztás). A frekvenciaosztásos és időosztásos multiplexerek használata a "klasszikus" telefon technikában fejlődött ki, a kód-osztásos multiplexer technikák pedig a digitális átvitel elterjedésével nyertek egyre nagyobb teret (szigorúan véve a kód-osztásos multiplexer is időbeli multiplexálást végez).

Egyértelműen analóg technológia a frekvencia- és hullámhossz-osztásos multiplexálás, míg digitális technológia a kód-osztásos multiplexálás. Az időosztásos technológia mindkét csoportba besorolható. Nagyobb rendszerek felépítése érdekében ezeket az elveket egymás mellett egyidejűleg is alkalmazhatják.





9.4 ábra: CD4051 analóg multiplexer és SN74151 digitális multiplexer

A címző bemenetek logikai változóival egyértelműen kijelölhetjük azt az egyetlen bemenetet, amelynek a jele a kimenetre el fog jutni. Csak egyetlen bemenet választható ki egyidejűleg és csak ennek a bemenetnek a jele jut a kimenetre.

Léteznek analóg multiplexerek is, ezeknél a bemenetek kijelölése szintén bináris jelekkel történik, a kiválasztott jel azonban nemcsak logikai feszültségszinteket, hanem tetszőleges analóg értéket is felvehet. Az analóg multiplexerek belső felépítése is különbözik a digitálisakétól. Ezeket nem kizárólag logikai hálózatokból építik fel, hanem a bemenetválasztást és az engedélyezést végző logikai hálózat analóg kapcsolóelemeket vezérel, amelyek általában tervezérlésű tranzistorokból épülnek fel. Az alábbi ábrán a CD4051 típusú analóg multiplexer áramköri rajza látható.

A kiválasztás működése az alábbi igazságtáblázat alapján érthető meg.

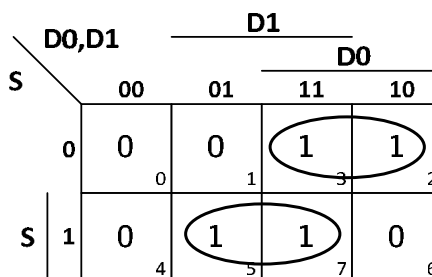
S1	S0	D3	D2	D1	D0	Y
0	0	X	X	X	D0	D0
0	1	X	X	D1	X	D1
1	0	X	D2	X	X	D2
1	1	D3	X	X	X	D3

Az S0, S1 kiválasztó bemenetek által kiválasztott bemenet jele jelenik meg a kimeneti lábon, eközben a többi bemenet állapota a kimenet szempontjából közömbös.

Példa: Multiplexer tervezése

A multiplexer tervezési menetét egy egyszerű két bemenetű multiplexeren mutatjuk be. Az áramkörnek két adatbemenete, egy kiválasztó bemenete és egy kimenete van. Az igazságtáblázatból látható, hogy S=0 esetén Y=D0, S=1 esetén Y=D1.

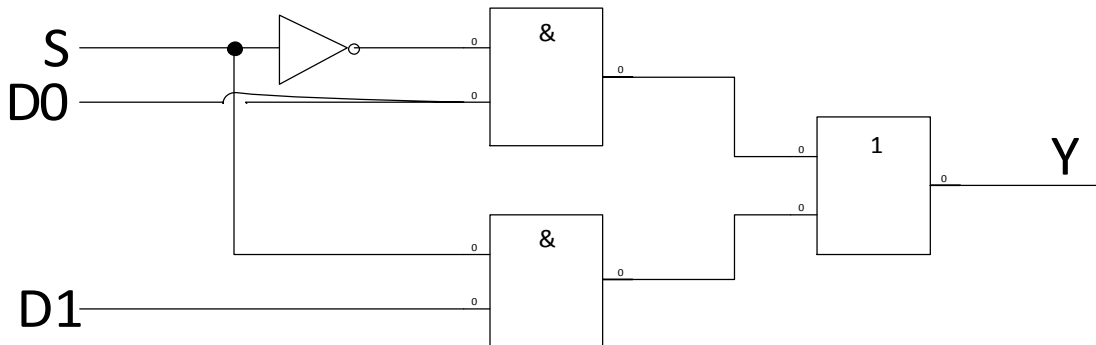
S0	D1	D0	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1



1	1	0	0
1	1	1	1

$$Y = \bar{S}D_0 + SD_1$$

A multiplexer működését megvalósító logikai hálózat:

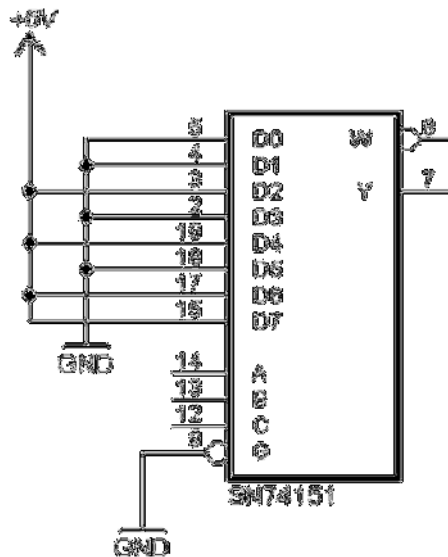


Természetesen hasonló elvek alapján bonyolultabb felépítésű vagy esetleg működés tekintetében eltérő multiplexereket is lehet tervezni.

Logikai függvények megvalósítása multiplexerekkel

Egy n számú kiválasztó bemenettel és 2^n számú adatbemenettel rendelkező multiplexer segítségével bármely n bemenetű logikai függvény realizálható. A megoldás alapja, hogy a kiválasztó bemenetekre kapcsolt adott bemeneti kombinációhoz tartozó kimeneti logikai szintet kötjük a multiplexer megfelelő adatbemenetére.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



A fenti példa egy háromváltozós logikai függvény realizációját mutatja, egy három kiválasztó bemenettel rendelkező multiplexer felhasználásával.

Vegyük észre!!! A fenti példában szereplő 8 bemenetű multiplexert felhasználhatjuk az összes lehetséges négyváltozós Boole függvény előállítására. Négy bemeneti és egy kimeneti változó esetén a lehetséges függvénykombinációk száma:

$$N_f = (2^m)^{2^n} = (2^1)^{2^4} = 2^{16} = 65536$$

ahol $n = 4$ a bemeneti változók száma

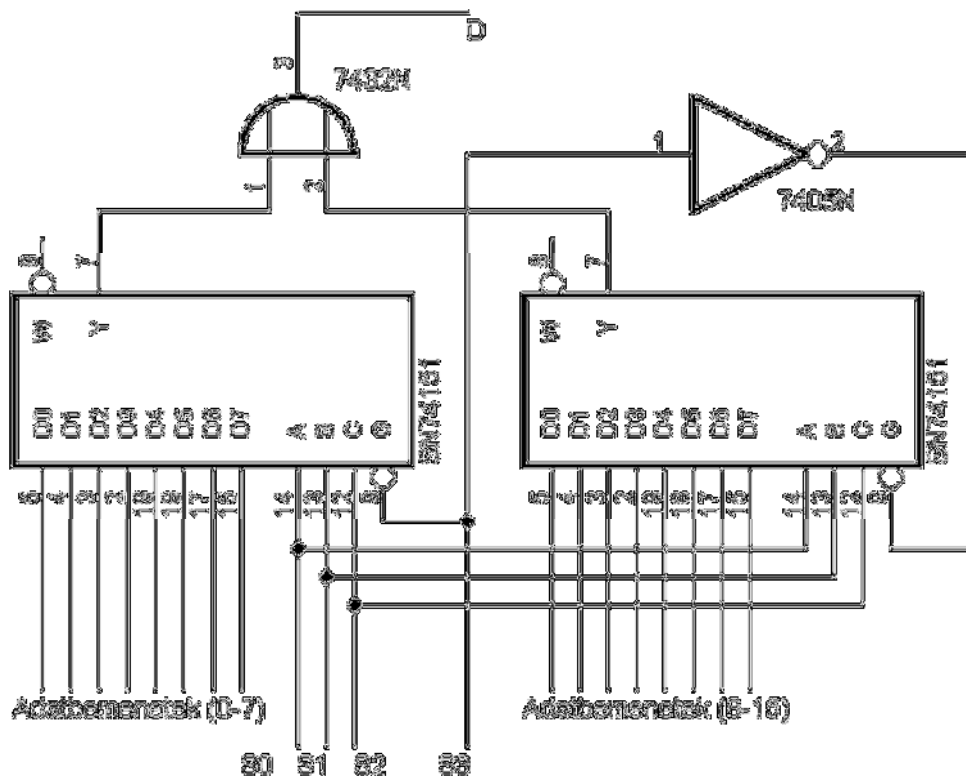
$m = 1$ a kimeneti változók száma

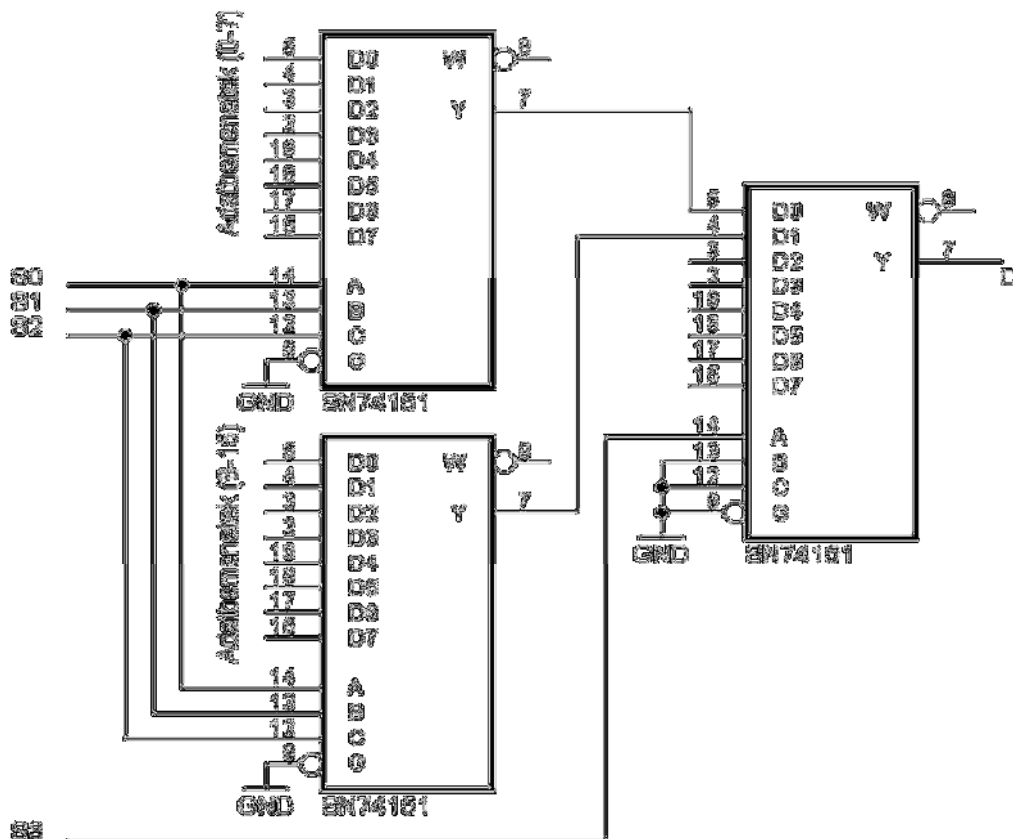
A megoldáshoz azt kell észrevennünk, hogy a bemenetekre nem csak 0 és 1 logikai értéket kapcsolhatunk, hanem a negyedik, D változó ponált, vagy negált értékeit is köthetjük. Mivel egy

bemenetre így négyféle logikai érték kerülhet (0, 1, D, \overline{D}), a nyolc bemenetre összesen $4 * 4 * 4 * \dots = 4^8 = 655$ féle kombináció. Ezek közül választ az A,B,C bemenet, tehát ez nem jelent újabb szabadsági fokot. Ez tulajdonképpen nem más, mint a négyváltozós függvény D-re vonatkozó diszjunkt dekompozíciója.

Multiplexerek bővítése

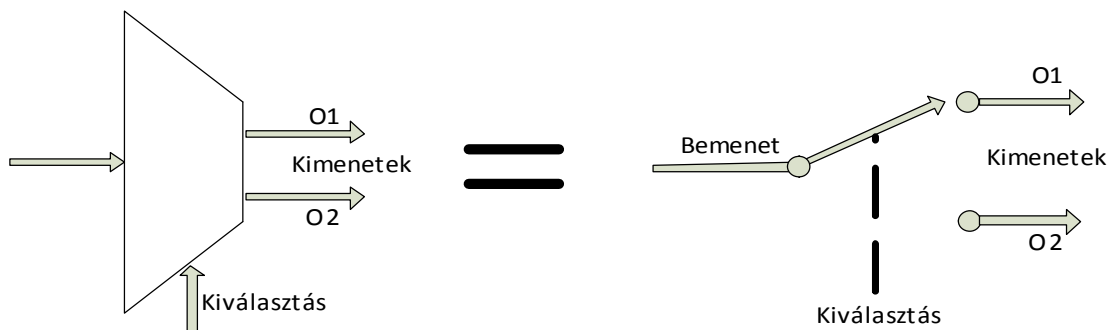
Amennyiben a bemenetek számát meg kell növelni, lehetőség van a multiplexer egységek bővítésére. A két multiplexer egység gyakorlatilag párhuzamosan kapcsolódik egymással. Közös a három kiválasztó bemenetük (S0, S1, S2) és közös az adatkimenet (D). Mivel a kimenetek nem háromállapotúak, az összekapcsolásuk csak logikai áramkör közbeiktatásával lehetséges, vagy egy újabb multiplexer szükséges a kiválasztásukhoz. A következő ábrán a bemenetek száma a két alkalmazott multiplexer kimenetei számának az összege, vagyis ugyanolyan típusokat használva megduplázódik. A két áramkör között a harmadik kiválasztó bemenet választ (S3). Az inverter biztosítja, hogy a két áramkör engedélyező bemenete ellentétes értékű vezérlőjelet kapjon.



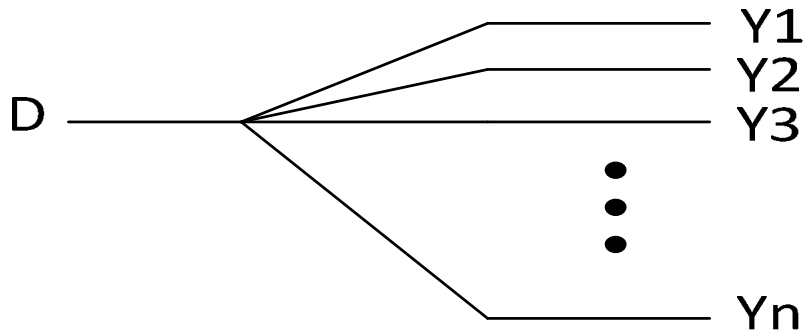


Demultiplexerek

A demultiplexer egy olyan digitális kapcsoló, amely egy bemenő forrást n kimenet valamelyikére kapcsolja. Általában n -ből 2^n -re dekódolót használunk. A demultiplexerek ezen kívül rendelkezhetnek még engedélyező bemenettel is.



A demultiplexerek feladata az adatok szétszétvása. A bemeneten lévő adat a címbemenetek által meghatározott sorszámú kimenetre kerül. A demultiplexernek egyszerre csak egy kimeneti vonalán lehet jel. A demultiplexernél a megcímezett kimenet nem aktív lesz, hanem felveszi a kiválasztott bemenet értékét.

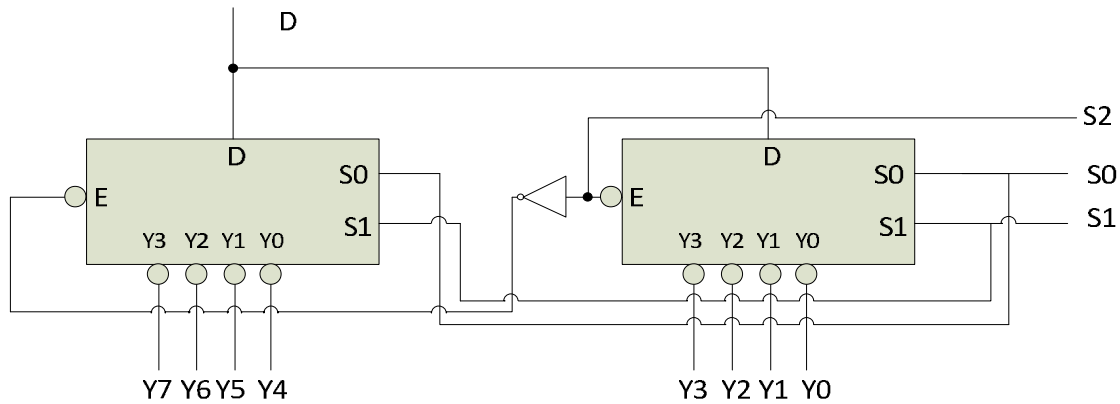


Példa: Három bemenetű, engedélyező bemenettel rendelkező demultiplexer működésére.

Igazságtáblázat:

A0	A1	A2	E	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
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	0	-	-	-	-	-	-	-	-
0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0
1	0	1	1	0	0	0	0	0	1	0	0
1	1	0	1	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Demultiplexerek bővítése. Az alábbi példa alapján, a multiplexerekhez hasonlóan, lehetőség nyílik a demultiplexer áramkörök bővítésére is, amennyiben a kimenetek száma nem elegendő. A megoldás során a két demultiplexer egység párhuzamosan kapcsolódik egymással, ez azt jelenti, hogy közös a két kiválasztó bemenetük (S0, S1) és közös az adatbemenet (D). A kimenetek száma a két alkalmazott demultiplexer kimenetei számának az összege, vagyis ugyanolyan típusokat használva megduplázódik. A két áramkör között a harmadik kiválasztó bemenet választ (S2). Az inverter biztosítja, hogy a két áramkör engedélyező bemenete ellentétes értékű vezérlőjelet kapjon.



Összeadó áramkörök

Az összeadó áramkörök bináris értékek összegét képzik, logikai áramkörös megvalósítással.

Felépítésük szerint lehetnek:

- Félösszeadók
- Teljes összeadók

Működési mód tekintetében:

- Soros összeadó
- Párhuzamos összeadó

Az operandusokat tekintve:

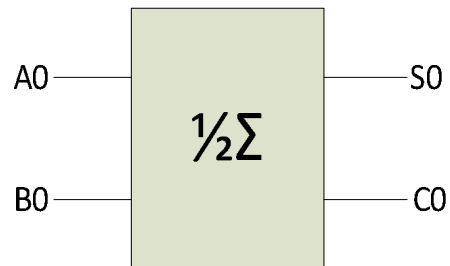
- Bináris összeadó
- BCD összeadó

Félösszeadó (half-adder: HA)

A0 és B0 bemenetre kapcsolt két bináris érték összegét képezi. A kimenet értéke S0-on jelenik meg. A képződött átvitel C0. Nem veszi figyelembe az előző helyiértéken képződött átvitelt, ezért hívják félösszeadónak. Emiatt csak a legkisebb helyiértéken alkalmazható.

A félösszeadó igazságtáblája:

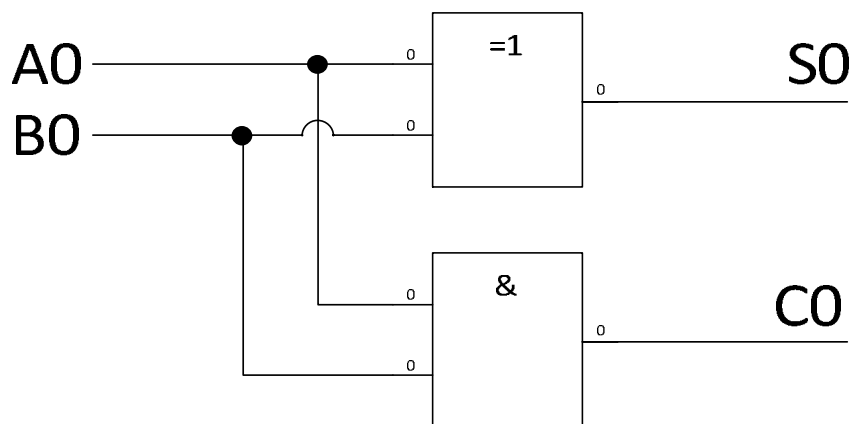
A0	B0	S0	C0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S0 = \overline{A0} \cdot B0 + A0 \cdot \overline{B0} = A0 \oplus B0$$

$$C0 = A0 \cdot B0$$

A működést megvalósító kapcsolási vázlat:



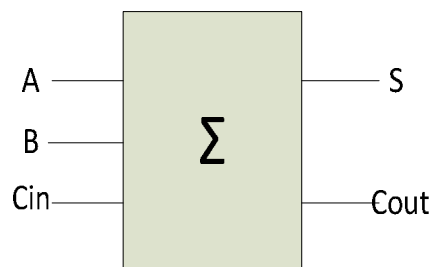
Teljes összeadó (full-adder: FA)

Ez az áramkör előző helyiértéken képződött átvitelt is képes figyelembe venni.

Az A és B bemenetekre kapcsolt két logikai érték összegét képezi. Az előző helyiértéken képződő átvitelt a Carry In bemeneten veszi figyelembe. S kimenet az eredmény, Carry Out az összeadás során képződött átvitel értéke.

A teljes összeadó áramkör igazságtáblázata:

A	B	Cin	S	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



$$S = A + B + C_{in}$$

aritmetikai kifejezéssel

$$S = A \cdot \overline{B} \cdot \overline{C_{in}} + \overline{A} \cdot \overline{B} \cdot C_{in} + A \cdot B \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C_{in}}$$

logikai kifejezéssel

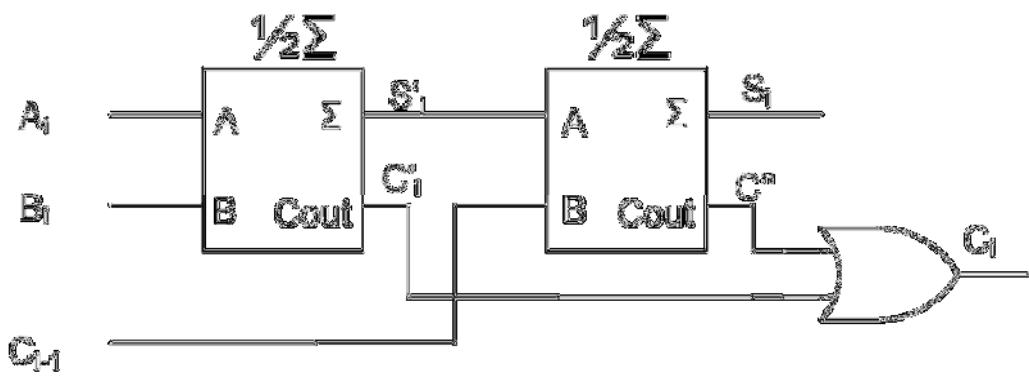
$$C_i = A \cdot B + A \cdot C_{i-1} + B \cdot C_{i-1}$$

logikai kifejezéssel felírva.

Tehát a képződő átvitel függ az adott helyen az összegtől és előző helyen képződő átviteltől is.

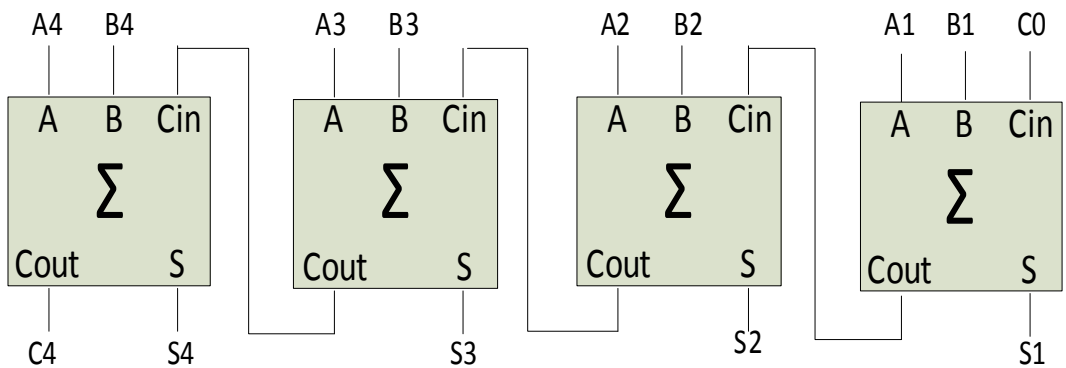
Teljes összeadó áramkör készítése két félösszeadó alkalmazásával.

Az első félösszeadó A és B összegét képezi. Az eredmény S'_i kimeneten jelenik meg. A második összeadó fokozat a kapott eredményhez hozzáadja az átvitel értékét ($S'_i + C_{i-1}$). Az eredmény C'' kimeneten kapjuk. Mivel az A+B összeadása során is képződhet átvitel, tehát vagy az első, vagy a második fokozat ad átvitel értéket, a két átvitel eredménye vagy kapcsolattal képezhető.



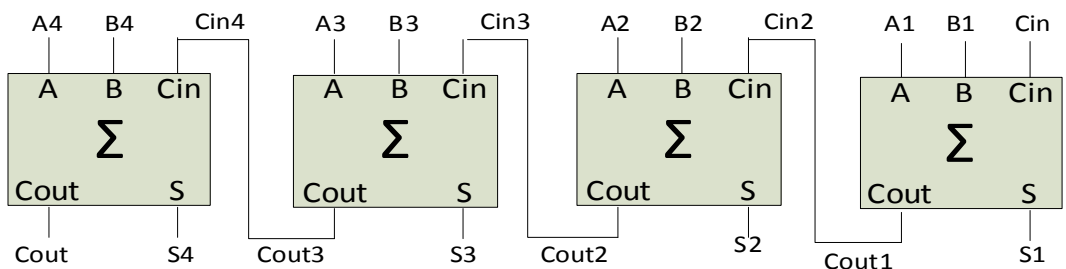
Több bit összeadásához több összeadó áramkört kell összekapcsolni. Az összeadás módja lehet soros vagy párhuzamos rendszerű.

Az alábbi ábrán soros összeadó áramkört (Ripple Carry Adder - RCA) láthatunk. Az 1-es indexű első fokozat adja össze a legkisebb helyiértékű bitet, majd ennek az átvitel kimenete kapcsolódik a következő fokozat átvitel bemenetére, és így tovább. Az eredmény értékeit az aktuális összeadó kimenetén olvashatjuk le. A legelső fokozat átvitel bemenetét vagy az előző fokozat kimenetét, vagy ha ez a legkisebb helyiérték, akkor logikai '0' értékre kell kötni (gyakorlati megvalósításban föld potenciálra szokás kötni). Az összeadólánc utolsó fokozatának átvitel kimenetén (C4) lesz a teljes lánc átvitel értéke leolvasható.



Az ily módon felépített összeadó hátránya, hogy lassú működésű, mivel az előző fokozat kimenetének már fel kell vennie az értéket ahhoz, hogy a következő fokozat ezzel számolni tudjon.

Az összeadás sebességét lehet gyorsítani, ha az összeadó áramkörök párhuzamos működése mellett az átvitel értékét egy külön logikai hálózattal állítjuk elő. Az ilyen módon működő összeadó áramköröket párhuzamos összeadóknak hívjuk. (Look-ahead Carry Adder)



$$C_{g4} = A_4 \cdot B_4$$

$$C_{p4} = A_4 + B_4$$

$$C_{g3} = A_3 \cdot B_3$$

$$C_{p3} = A_3 + B_3$$

$$C_{g2} = A_2 \cdot B_2$$

$$C_{p2} = A_2 + B_2$$

$$C_{g1} = A_1 \cdot B_1$$

$$C_{p1} = A_1 + B_1$$

Az eddig megismert elvek és összefüggések alapján a párhuzamos működésű összeadó elvét is könnyen megérthetjük.

A félösszeadó működését leíró összefüggések:

$$S0 = A0 \oplus B0$$

$$C0 = A0 \cdot B0$$

A teljes összeadó átvitel képzését leíró összefüggést átrendezve kapjuk:

$$C_i = A \cdot B + A \cdot C_{i-1} + B \cdot C_{i-1} = A \cdot B + C_{i-1}(A + B)$$

ahol az AB szorzat az aktuális összeadónál képződött átvitel (Generate Carry) C_g , A+B összeg pedig az un. terjedő átvitel (Propagate Carry) C_p . Tehát az aktuális helyiértéken képződő átvitel egyenlő az aktuálisan képződött átvitel és a terjedő átvitel VAGY kapcsolatával. Ezek alapján felírhatók az alábbi összefüggések:

$$Co^u t_i = C_{gi} + Cin_i \cdot C_{pi}$$

$$Cin_i = Co^u t_{i-1}$$

Ezt az összefüggést alkalmazva a számláncban szereplő összes összeadó áramkörre, figyelembe véve a fenti ábra felépítését, eljuthatunk a párhuzamos teljes összeadó működési vázlatáig.

Paritás-generáló és figyelő áramkörök

A paritás vagy párosság ellenőrzés a legegyszerűbb hibafelügyelő eljárás. Az aktuális bináris kódban az egyesek számát párosra vagy páratlanra egészíti ki egy plusz bittel. A bináris érték és a paritás vizsgálatával eldönthető, hogy sérült-e az információ. Megkülönböztetünk páros és páratlan paritást aszerint, hogy párosra vagy páratlanra egészítjük ki az egyesek számát.

Az eljárás hátránya, hogy csak egyszerűségénél fogva csak a hiba detektálására alkalmas, javításra nem. Több bit esetleges sérülése esetén nem egyértelmű, hogy meghibásodott-e az adat.

A legegyszerűbb két bites paritásképző áramkör egy kizáró vagy kapcsolat:

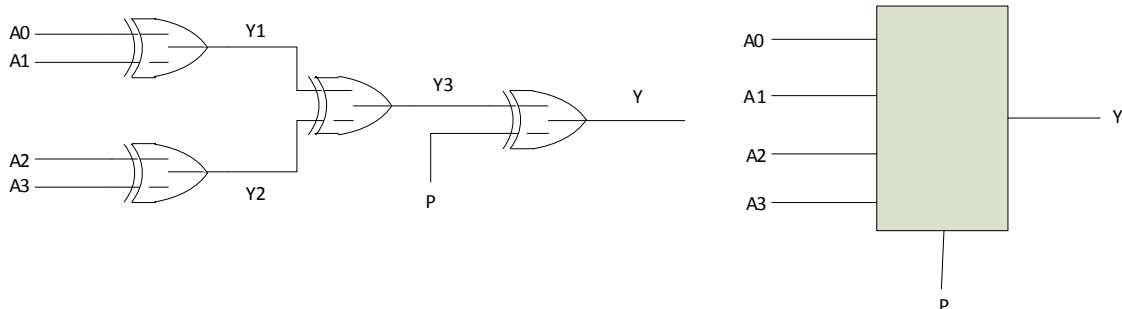
Páros paritás		Páratlan paritás	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Ez az értékek páros paritását képezi. A páratlan paritás egyszerű invertálással előállítható.

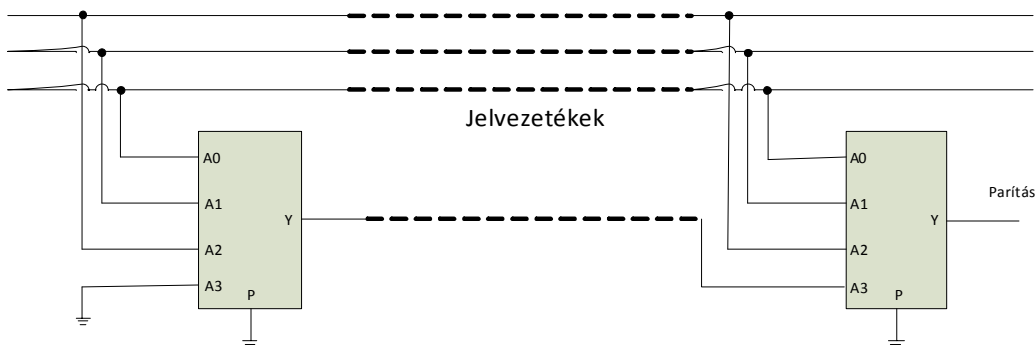


Több bites értékek paritásának ellenőrzéséhez az egyszerű paritásvizsgáló kapcsolást kaszkádosíthatjuk.

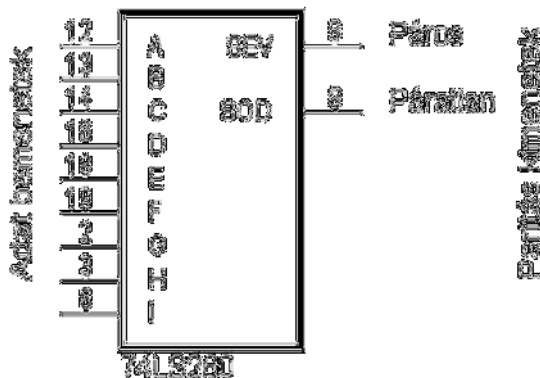


Ahol a P bemenet a paritás típusának (páros, páratlan) beállítására szolgáló bemenet.

Adatátviteli rendszerek úgy végzik a paritás vizsgálatát, hogy az adó oldalon generált paritás értéket elküldik az adattal együtt. A vevő oldalon is előállítják a paritás értékét, majd összehasonlítják a kapott paritás értékkel. Egyezés esetén a fentebb már említett feltételeknek megfelelően nem történt hiba. Eltérés esetén az adatokat újra el kell küldeni. Egy egyszerű paritásgeneráló és ellenőrző adatátviteli hálózat vázlatja látható a következő ábrán.



A paritásgeneráló és ellenőrző áramkörök integrált kivitelben is elérhetőek. Általánosan használt típusaik SN74LS180, SN74LS280.

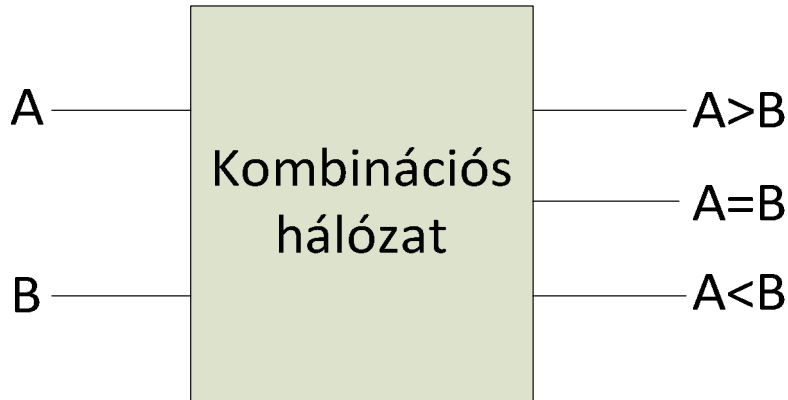


Digitális komparátorok

A digitális komparátorok két bináris érték közötti relációt jelzik, azaz kisebb, nagyobb vagy egyenlő. A három eset közül mindig csak egy lehet igaz.

Egy n bites számokat összehasonlító áramkörnek mindig $2n$ db adatbemenete van és a kimeneti állapotoknak megfelelő három darab kimenete. A bemenetekhez jön még az előző helyiértékekről kapott kisebb, nagyobb, egyenlő bővítő bemenetei jel.

Egy egybites komparátor felépítését mutatja a következő ábra.



Az egybites komparátor igazságtáblázata:

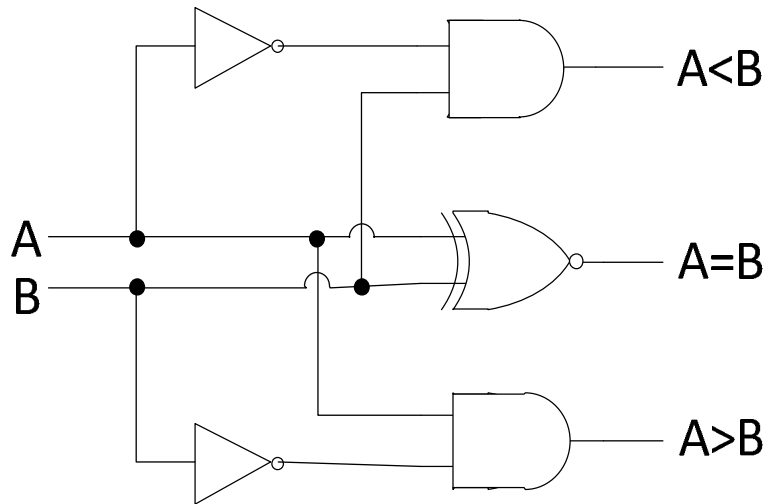
A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

$$f_{A<B} = \bar{A} \cdot B$$

$$f_{A>B} = A \cdot \bar{B}$$

$$f_{A=B} = \bar{A} \cdot \bar{B} + A \cdot B = \overline{A \oplus B}$$

A működést megvalósító logikai hálózat:



Példa: Kétbites komparátor

Két bináris szám akkor egyenlő, ha a megfelelő biteik megegyeznek egymással. Ez alapján a szabály alapján, figyelembe véve az egybites komparátornál kapott logikai függvényt, igazságtáblázat nélkül is felírható az egyelőséget jelző kimenet logikai függvénye:

$$f_{A=B} = \overline{A_0 \oplus B_0} \cdot \overline{A_1 \oplus B_1} = (A_0 \cdot B_0 + \bar{A}_0 \cdot \bar{B}_0)(A_1 \cdot B_1 + \bar{A}_1 \cdot \bar{B}_1)$$

Az egyenlőtlenségi relációkhoz a következő összefüggések alapján juthatunk:

$$A < B, \text{ ha } A_1 < B_1 \text{ vagy } A_1 = B_1 \text{ és } A_0 < B_0$$

Az $A < B$ relációt megvalósító logikai függvény:

$$f_{A < B} = \overline{A_1} \cdot B_1 + (A_1 \cdot B_1 + \overline{A_1} \cdot \overline{B_1}) \overline{A_0} \cdot B_0$$

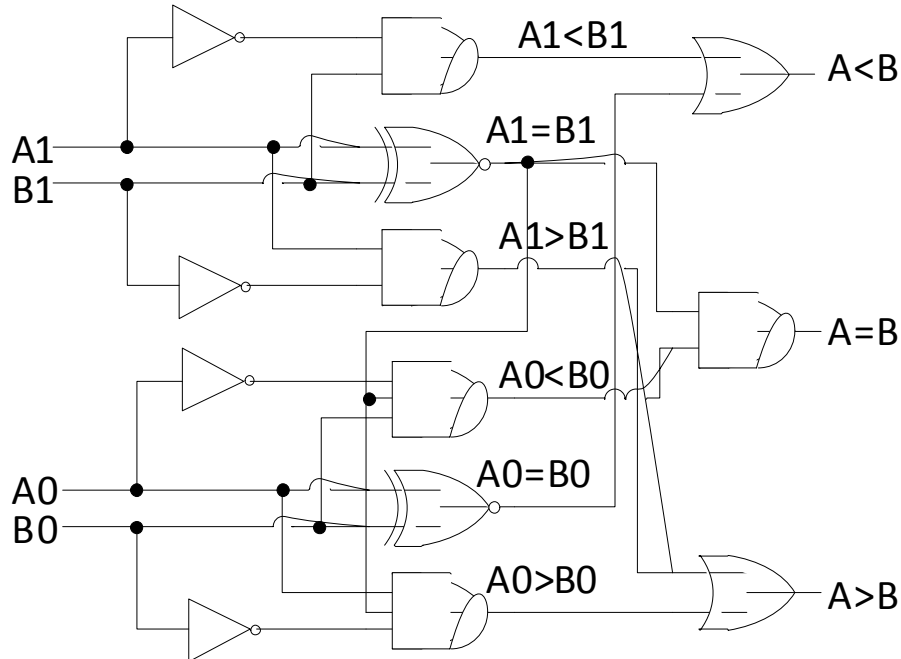
Valamint

$$A > B \text{ ha } A_1 > B_1 \text{ vagy } A_1 = B_1 \text{ és } A_0 > B_0$$

Az $A > B$ relációt megvalósító logikai függvény:

$$f_{A > B} = A_1 \cdot \overline{B_1} + (A_1 \cdot B_1 + \overline{A_1} \cdot \overline{B_1}) A_0 \cdot \overline{B_0}$$

A realizáció után A és B két bites számot összehasonlító logikai hálózat:



Példa: Több bites komparátorok:

Integrált kivitelben 4 bites komparátor áramköröket forgalmaznak, mind TTL, mind CMOS alapú áramkörökből. A TTL típus típusjele SN7485, a CMOS típus CD4063. Példaként most a TTL áramkört mutatjuk be, de működést tekintetében teljesen megegyezik a CMOS típussal.

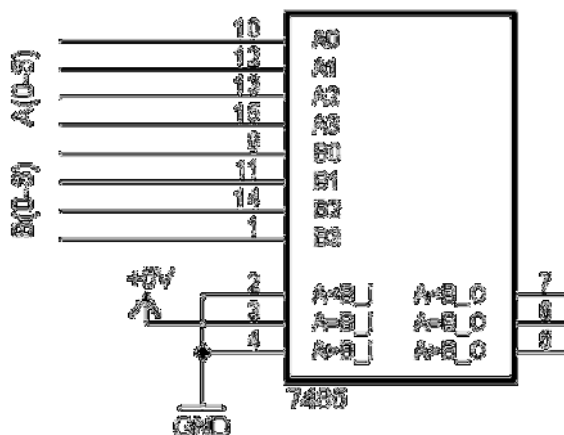
Az SN7485 négybites nagyság komparátor 2x4 db számbemenettel rendelkezik (A0-A3, B0-B3), amelyek a két összehasonlítandó szám bitjei. Ezeken felül három darab ún. bővítő bemenettel:

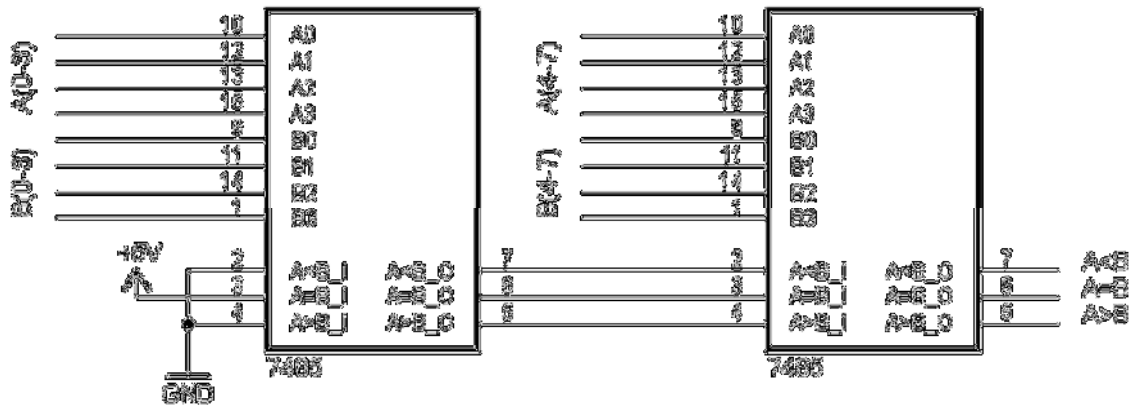
$$A_{i-1} < B_{i-1}, A_{i-1} = B_{i-1}, A_{i-1} > B_{i-1}$$

Kimenetei az összehasonlítás eredményeként kapott relációkat jelzik:

$$A_i < B_i, A_i = B_i, A_i > B_i$$

Amennyiben több bites számok összehasonlítása szükséges, a komparátor egységeket lehet kaszkádosítani, ahogy az alábbi ábra mutatja:





Ezzel a megoldással $n \times 4$ bites komparátor egység készíthető. A legkisebb helyiértéket összehasonlító egység bővítő bemeneteit az ábrának megfelelő módon kell bekötni. Az $A < B$ és $A > B$ bemenetekre logikai hamis értéket tehát alacsony szintet, az $A = B$ bemenetre logikai igaz, tehát magas szintet kell kapcsolni.

10. Kombinációs hálózatok megvalósítása Memóriával, illetve Programozható Logikai Eszközökkel

Memóriák: bevezetés

Memóriák legfőbb fizikai paraméterei – a teljesség igénye nélkül – a következők, amelyek az egyes memória modulok gyártóinak adatlapjain pontosan fel vannak tüntetve:

- Címvonál (Address): címvonalak száma határozza meg a memória mélységét.
- Adatvonál (Data): Az adatvonalak száma egy rekesz/cella szélességét („data width”, de hívják még szóhosszúságnak „word-length” is) határozza meg. Multiplexált cím/adat vonalként is össze szokták vonni, a memória lábak mérséklése miatt
- A memória kapacitása: a tárolható szavak számát jelöli. Ha ‘n’ darab címvezeték feltételezünk, akkor memória 2^n számú rekeszt/cellát tartalmaz (tehát egyértelműen kifejezhető a címek, illetve adatvonalak szélességéből)
- Írasi tranzakciót (Write) jelző vonal,
- Olvasási tranzakciót jelző vonal (Read): ez utóbbi kettőt gyakran egy jelként vonják össze, azért hogy spóroljanak a memória lábakkal (pl: RnW = Read Not Write: azaz ha alacsony logikai szint, írás történik, ha magas aktív, akkor pedig olvasás)
- Memória modul órajele (Clock)
- Memória kiválasztása, engedélyezése (Chip Select, vagy Enable)

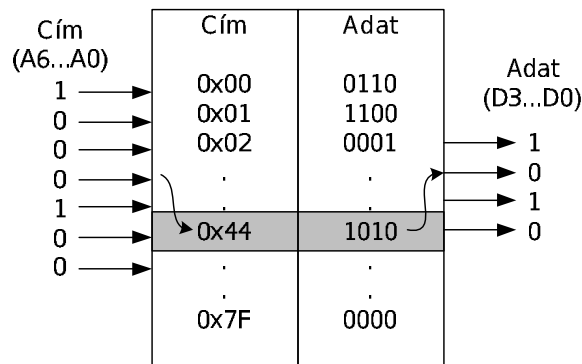
Memóriák elérésének két fő módja a következő:

- RAM (Random Acces Memory): véletlen hozzáférésű írható / olvasható memória
- ROM (Read-Only Memory): csak olvasható memória

Ezekben belül statikus (frissítés nélküli), illetve dinamikus (működés közbeni frissítést igénylő) memóriákat is meg kell különböztetni.

Memóriák működéséről röviden:

Egy adott memória rekesz (cella) tartalmát úgy lehet kiolvasni, hogy a címbemenetekre adjuk a kiolvasandó cella sorszámát (indexét), majd engedélyezzük a Chip Select, valamint a Read jeleket. A címek dekódolása után az adatkimeneteken egy véges úgynevezett hozzáférési idő elteltével az azonosított rekeszben tárolt adat értéke (lehet újabb cím is) jelenik meg. Az írasi tranzakció hasonló módon történik, mint olvasásnál. A címeket általában hexadecimális formátumban definiálják, de a logikai hálózatokkal történő összehasonlítás végett később bináris formában adjuk meg. A memória-gyártók adatlapjain feltüntetett ciklusidő a két egymás közötti írasi-, vagy olvasási tranzakció közötti időkülönbséget jelenti. A felsorolt fizikai illetve időbeli paramétereken kívül rengeteg további paramétert kell figyelembe venni egy memória használata során.

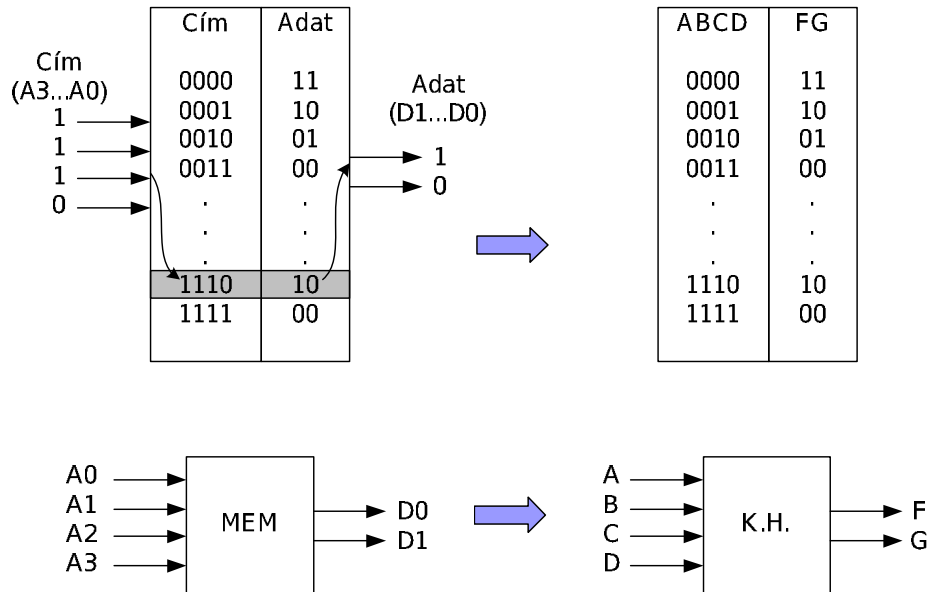


10.1 ábra: Memória általános felépítése: címvonalak (7-bites), és adatvonalak (4-bites) megadásával

Kombinációs hálózatok és a memóriák kapcsolata

A kombinációs hálózatokban (a továbbiakban K.H.) a független változók egy adott bemeneti kombinációjára az igazságtábla ugyanazon sorában feltüntetett kimeneti kombináció a válasz. Tehát ha a kombinációs hálózat bemenetén megadunk egy bináris számot, válaszul egy másik bináris számot várunk a kimeneteken.

Pontosan ez az analógia mutatkozik meg a memóriák működése esetében is: azaz kiolvasáskor mindenegyes cím megadásakor egy előzőleg betöltött adat jelenik meg a kimeneten. Vagyis, ha egy memóriát egy vele azonos számú be-, és kimenettel rendelkező K.H. igazságtáblája szerint töltünk fel, akkor ez a **memória helyettesítheti** magát a **kombinációs hálózatot** (10.2 ábra). A következő részekben megvizsgáljuk, hogyan építhető fel egy K.H. egy memória elem segítségével.



10.2 ábra: Kombinációs hálózat és memória ekvivalenciája (n=4 bemenet, m=2 kimenet esetén)

A kombinációs logikai hálózat memória elemekkel történő megvalósításának tulajdonságai a következők. Előnyök:

- Könnyen átprogramozhatók, így a fejlesztési szakaszban nem kell újraépíteni logikai kapuk felhasználásával az egész áramkört (még apró változtatásnál sem).
- Nem igényel függvény-egyszerűsítést (minimalizálást).
- Nem fordulhat elő benne statikus, és dinamikus hazárd (lásd későbbi 11. Fejezetben), mivel nem logikai kapukból és huzalozással épül fel, hanem az igazságtábla alapján, direkt módon kell meghatározni a beírni kívánt inicializáló adatokat.
- Viszont itt is van/kialakulhat funkcionális hazárd, amit pl. szinkronizációval szüntethetünk meg (nem szomszédos bemeneti címváltozásokra → kimeneti adatváltozások). Erre megoldás az EN engedélyező/tiltó bemenet bevezetése.

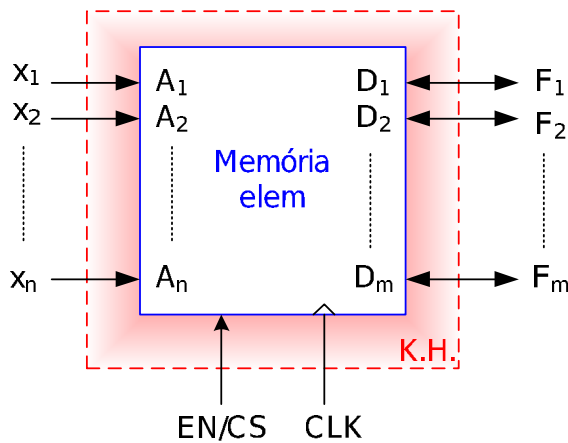
Hátrányok:

- Egy memória-áramkör lassabb, mint a logikai kapukból összeállított huzalozott kombinációs hálózat (főleg, ha több hierarchia szinten összekötött memória áramkörök késleltetését tekintjük).
- Speciális időzítési feltételekkel fogadhat csak jeleket (pl. a címnek bizonyos ideig stabilnak kell maradnia ahhoz, hogy ki lehessen olvasni az érvényes kimeneti értéket).

- Míg a függvény-egyszerűsítéssel kapott megoldás esetleg csak néhány kapuból állna, addig a memóriába a teljes igazságtáblázatot be kell programozni (tárolni): azaz n darab bemenethez mindenképp egy 2^n kapacitású memóriát kell választani (katalógus).
- A memóriaelem a jelenlegi félvezető gyártástechnológia esetén is legtöbb esetben drágább, mint a K.H. alkalmazása.

Memória áramkör alkalmazása több-kimenetű Kombinációs Hálózatok megvalósítására

K.H. \leftrightarrow Memória



- Bemeneti kombináció = címek
- Kimeneti kombináció = adat kimenetek
- EN/CS (Enable, más néven Chip Select) – memória elem engedélyezése, vagy tiltása
- **Huzalozott VAGY kapcsolat:** azt jelenti, hogy egyszerre akár több memória elem kimenete is összekapcsolható „huzalozott” módon, de ekkor mindig csak egy memória áramkör működése engedélyezett (EN='1'), míg a többi tiltott (EN='0').

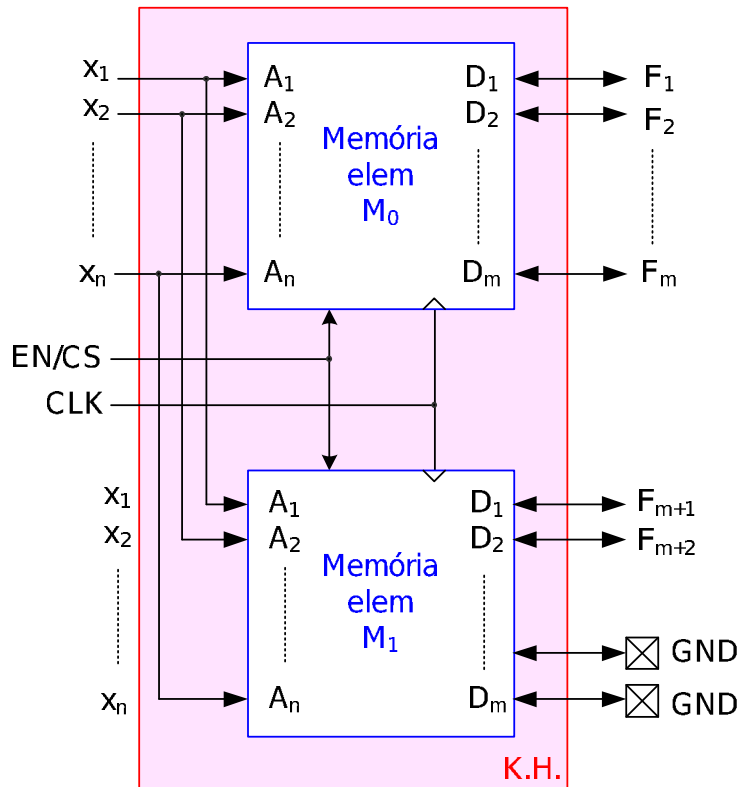
10.3 ábra: Kombinációs hálózat megvalósítása memória áramkör segítségével

A K.H. megvalósításának lehetséges esetei a következők:

- 1.) A megvalósítandó K.H. bemenete kevesebb, mint a memória címbemenete (a memória nem használt plusz cím-bemeneteit le kell földelni, GND).
- 2.) A megvalósítandó K.H. kimenete kevesebb, mint a memória adatkimenete (a memória nem használt plusz adat-kimeneteit le kell földelni, GND).
- 3.) A megvalósítandó K.H. kimenete több, mint a memória adatkimenete (több memória elem kell).
- 4.) A megvalósítandó K.H. bemenete több, mint a memória címbemenete (több memória elem, dekóder kell).
- 5.) A megvalósítandó K.H. bemenete és kimenete is több, mint a memória címbemenete, ill. adatkimenete (több memória elem, dekóder kell).

A lehetséges esetek közül az első kettő megvalósítása (1-2) nem igényel különösebb vizsgálatot, míg az utóbbi hármat (3.-5.) részletesebben is meg kell vizsgálni.

3.) K.H. megvalósítása memóriával – ha a megvalósítandó K.H. **kimeneteinek** száma több ($m+k$), mint a memória adatkimeneteinek száma (m): ekkor nem elegendő egyetlen memória elem, hanem kiegészítő memória elemekkel kell bővíteni a hálózatot. A következő 10.4 ábrán, a megvalósítandó K.H kimeneteinek száma $k:=2$ -vel több kell, hogy legyen, mint amennyit egy memória elem biztosítani tud (m), ezért az F_{m+1} , illetve F_{m+2} kimeneteket egy újabb (alsó) memória elem D_0 , illetve D_1 -el jelölt adatkimeneteire kell bekötni. Az alsó M_1 memória elem nem használt $D_2 \dots D_m$ adat kimeneteit földre (GND) kell kötni.

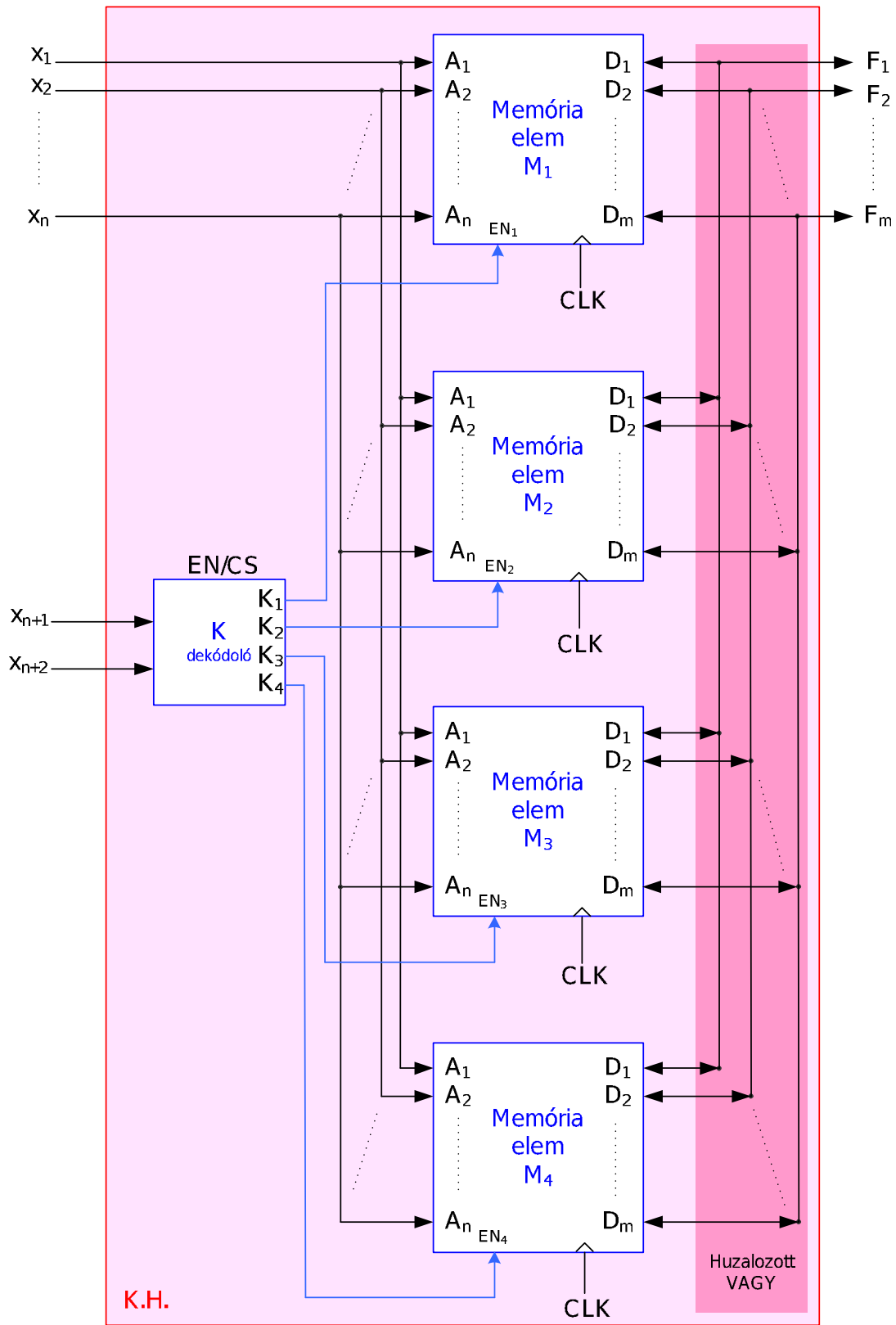


10.4 ábra: Kombinációs hálózat megvalósítsa memória áramkör segítségével

4.) K.H. megvalósítása memóriával – ha a megvalósítandó K.H. **bemeneteinek** száma több ($n+p$), mint a memória cím-bemeneteinek száma (n): ekkor nem elegendő egyetlen memória elem, hanem kiegészítő memóriákkal kell bővíteni a hálózatot. A következő példában legyen a $p = 2$, tehát a megvalósítandó K.H. kettővel több bemenetet igényel, mint amennyi címbemenete egy memória elemnek van. Továbbá, mivel 2^n számú bináris kombináció helyett 2^{n+p} -t kell tárolni ($= 2^p \times 2^n$, azaz 4×2^n , ha $p=2$), összesen 4 memória elem alkalmazását fogja jelenteni. Ezzel együtt a memória elemek engedélyezését is meg kell oldani, amelyhez egy dekódoló logikát használunk. A bővítendő 'K' átkódoló logikai hálózat beépítése fogja vezérelni a memóriák EN/CS jelének engedélyezését, vagy éppen tiltását, úgy, hogy egy időben egyszerre csak egy memória legyen aktív. A 'K' átkódoló a hagyományos „n-ből 1 kód”, azaz a „one-hot” egyes-súlyú dekódolást alkalmazza, (itt természetesen más dekódoló mechanizmus használatára is lehetőség van). Az egyes-súlyú dekódoló logika a két többlet bemenet $x(n+1)$, illetve $x(n+2)$ bemenetek esetén a következőképpen adható meg:

$x(n+1)$	$x(n+2)$	K1	K2	K3	K4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

10.5 ábra A kapcsolódó dekódoló logika működésének leírása



10.6 ábra: Kombinációs hálózat megvalósítása memória áramkör segítségével

5.) K.H. megvalósítása memóriával – ha egyrészt a megvalósítandó K.H. **bemeneteinek** száma több ($n+p$), mint a memória cím-bemeneteinek száma (n), másrészt a megvalósítandó K.H. **kimeneteinek** száma is több ($m+k$), mint a memória adat-kimeneteinek száma (m). Ebben az esetben sem elegendő egyetlen memória elem alkalmazása, hanem kiegészítő memória elemekkel kell bővíteni a hálózatot.

Előző 3.) és 4.) módszereket együttesen kell alkalmazni a bővítéshez.

Megjegyzés: nincs akadálya annak, hogy a 'K' átkódoló, vagy elő-feldolgozó hálózatot is memóriával (tipikusan ROM-al) valósítsanak meg. Azonban ekkor a több-szintű, sorba kapcsolt memória hálózat miatt csökkenhet a megvalósított hálózat működési sebessége.

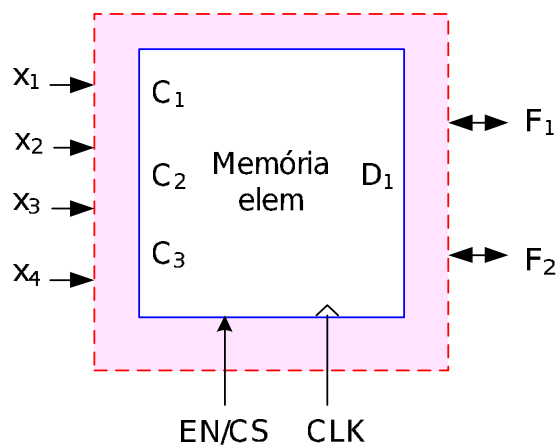
Példa:

Az megadott Memória elem segítségével realizáljon egy K.H.-ot, amelynek kimeneti logikai függvénye a következő:

$$F^{n=4}_1(x_1, x_2, x_3, x_4) = \sum_{i=0}^{2^n-1} (0,1,4,15)$$

$$F^{n=4}_2(x_1, x_2, x_3, x_4) = \sum_{i=0}^{2^n-1} (0,1,8,9,11)$$

A memória elem felépítése a következő:



10.7 ábra: Memória elem felépítése

Kérdések.

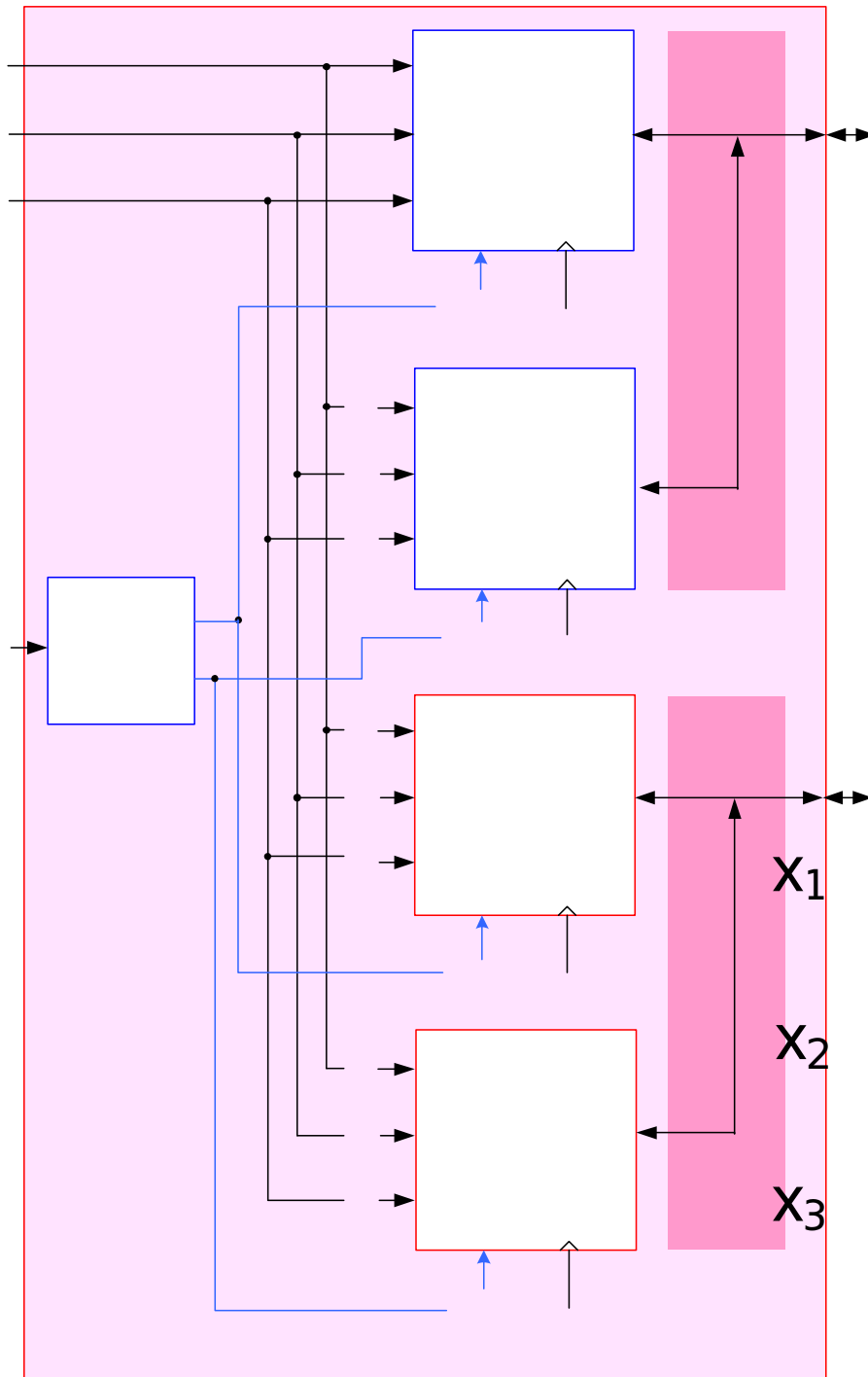
- Adja meg a megvalósítandó K.H. pontos felépítését a memória elem felhasználásával (rajz)!
- Ha szükséges, adja meg a dekódoló logika ('K') igazságtáblázatát is!
- Adja meg pontosan a memória inicializálását, feltöltését is!

Megoldás:

- Mivel egyrészt a Memória elem cím-bemeneteinek száma ($n=3$) 1-el *kevesebb*, mint a realizálandó K.H. bemeneteinek ($n+p=4$, ahol így $p=1$) száma, másrészt a memória adat-kimeneteinek száma ($m=1$) is 1-el *kevesebb*, mint a megvalósítandó K.H. kimeneteinek ($m+k=2$, ahol így $k=1$) száma, ezért egyetlen memória elemmel nem lehet megvalósítani a K.H.-ot. Az áramkört bővíteni kell, azaz:
 - egyrészt több (2 memória elem) kell F_m kimenetenként,
 - másrészt, Dekódoló logika ('K') integrálása szükséges, amely a $p=1$ db többlet bemenet ($x(4)$) hatására, a dekódolt jelek segítségével fogja vezérelni az egyes memória modulok

engedélyezését / tiltását. A helyes működéshez: $p \rightarrow 2^p$, azaz $1 \rightarrow 2$ dekóder szükséges, így K1, K2 dekódolt jelek vezérlik a CS/EN jeleket.

Az áramkör felépítése:



10.8 ábra: K.H. megvalósítása memória elemek segítségével

b) Dekódoló (1→2) logika igazságtáblázata:

x(4)	K1	K2
0	1	0
1	0	1

c) Memória elemek inicializálása a következő:

$$F^{n=4}_2(x_1, x_2, x_3, x_4) = \sum_{i=0}^{2^n-1} (0,1,8,9,11)$$

$$F^{n=4}_1(x_1, x_2, x_3, x_4) = \sum_{i=0}^{2^n-1} (0,1,4,15)$$

	Cím	F1
MEM#1	0	1
	1	1
	2	0
	3	0
	4	1
	5	0
	6	0
	7	0

	Cím	F2
MEM#3	0	1
	1	1
	2	0
	3	0
	4	0
	5	0
	6	0
	7	0

	Cím	F1
MEM#2	8	0
	9	0
	10	0
	11	0
	12	0
	13	0
	14	0
	15	1

	Cím	F2
MEM#4	8	1
	9	1
	10	0
	11	1
	12	0
	13	0
	14	0
	15	0

10.9 ábra: Memória elemek feltöltése

Kombinációs hálózatok és a Programozható logikai áramkörök kapcsolata

Miért lehet fontos a programozható logikai eszközök alkalmazása?

A programozható logikai eszközök kifejlesztése előtt a digitális logikai áramkörök megvalósítására ROM memória elemeket használtak. A memóriáknak fejezet első részében ismertetett előnyös tulajdonságaik mellett vannak azonban hátrányaik is:

- Általában sokkal lassabbak, mint a dedikált logikai áramkörök.
- Nagyobb fogyasztásúak.
- Általában drágábbak is, főként a nagy sebességű félvezető eszközök.
- A kimeneteik tartalmazhatnak hazárdokat (nem kívánt kimeneti értékeket), amikor a bemeneteik változnak.

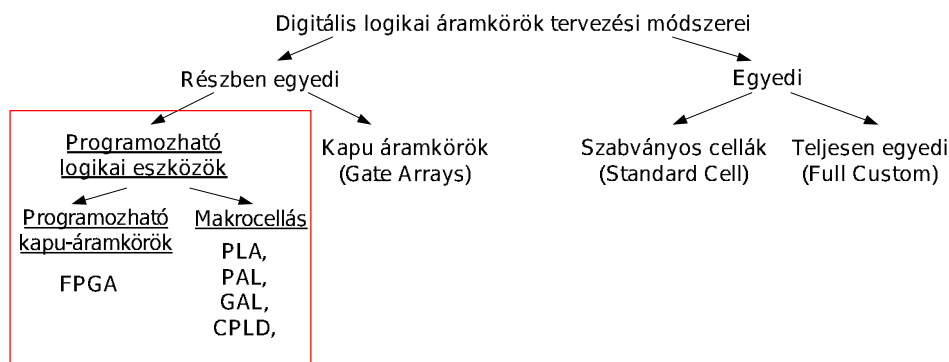
Az 1980-as évek előtti időszakban, a digitális áramkörök logikai hálózatainak tervezése során még nem álltak rendelkezésre olyan modern fejlesztő eszközök, mint napjainkban. Nagy komplexitású (sok bemenetű, sok kimenetű) logikai kombinációs és sorrendi hálózatok tervezése éppen ezért lassú és körülményes volt, sokszor papír alapú tervezéssel, többszöri manuális ellenőrzéssel, számításokkal párosult. Így egy prototípus-tervezés során nagy volt a hibavalószínűség, és fejlett szimulációs eszközökről (CAD) sem beszélhettünk akkoriban.

Ma mindezek együttese automatizált módon áll rendelkezésre (EDA – elektronikai tervezés-automatizálás), amely a programozható logikai architektúrák használata mellett (PLD), mind a

nyomtatott áramkörök (PCB), mind pedig az alkalmazás specifikus integrált áramkörök (ASIC) relatív gyors prototípus fejlesztését, megvalósítását (implementáció) és tesztelését (verifikáció) támogatja, valamint minimalizálja az esetlegesen előforduló hibákat. A hardver/firmware/szoftver részeket, együttesen és konzisztens módon lehet tervezni és tesztelni.

Az automatizált elektronikai tervezés mellett a programozható logikai eszközök használata tovább csökkenti a fejlesztésre fordítandó időt, és így minimalizálja a költségeket is. Éppen ezért sok alkalmazási területen érdekesebb először az adott funkció kifejtését egy programozható logikai eszközön megvalósítani és letesztelni, majd pedig – ha teljesülnek a követelmény specifikációban megfogalmazott feltételek – következhet csak a kitesztelt funkciónak megfelelő alkalmazás specifikus ASIC áramkör tervezése és tesztelése. Ez nagyban lerövidíti az integrált áramkörök fejlesztési idejét és nem megtérülő költségeit (NRI).

A felhasználó által definiálható logikai eszközök lehetséges megvalósítási formáit a következő 10.9 ábra foglalja össze:



10.10 ábra: Felhasználó által definiálható logikai megvalósítások

Két fő megvalósítási forma létezik áramkörök tervezése során: az Egyedi, illetve Részben egyedi tervezés. Az „Egyedi” ágban két további módszer létezik. Egyrészt a „Teljesen egyedi” esetben a tervező a legkisebb absztrakciós szinttől haladva, tranzisztorokból építkezik: annak teljes áramköri rajzolatát (layout), és paraméterezését is egyedileg kell, hogy beállítsa, míg „Szabványos cella” könyvtárak használata esetén kismértékben lehetőség van egy technológiai alkatrész adatbázisból a tranzisztorok behelyezésére, majd összekötésére (amelyek a teljesen egyedi módszerrel készültek). E két utóbbi módszer jellemzi leginkább a mai nagyon nagy integráltsági fokkal (tranzisztor számmal) rendelkező ASIC VLSI alkatrészeket. Ezek a módszerek biztosítják a legnagyobb teljesítményt és legkisebb fogyasztást, amely a leghosszabb prototípus-fejlesztési idővel, és a legmagasabb nem megtérülő költségekkel párosul.

A másik, „Részben egyedi” ágban egyrészt „Kapu Áramköröket” használhatunk, melyeket a gyártó rendelkezésre bocsát, viszont általában nem módosítható belső struktúrával rendelkeznek. Míg a konfigurálhatóság mértékét tekintve a legfejlettebb módszer, hogy programozható logikai eszközöket alkalmazunk. A teljesség igénye nélkül jelen jegyzetben a Programozható logikai áramkörök rövid bemutatására koncentrálunk.

Programozható logikai áramkörök

A Programozható logikai áramköröket (PLD: Programmable Logic Devices) általánosan a kombinációs logikai hálózatok és sorrendi hálózatok tervezésére használhatjuk. Azonban míg a hagyományos kombinációs logikai hálózatok dedikált összeköttetésekkel, illetve kötött funkcióval (kimeneti függvény) rendelkeznek, addig a programozható logikai eszközökben pontosan ezek változtathatók, az alábbi lehetséges módokon:

- A felhasználó által egyszer programozható / konfigurálható logikai eszközök (OTP: One Time Programmable), amelynél a gyártás során nem definiált funkció egyszer még megváltoztatható (ilyenek pl. a korai PAL, PLA eszközök).
- Többször, akár tetszőleges módon programozható logikai eszközök = rekonfigurálható (ilyenek pl. a korábbi GAL, vagy a mai modern CPLD, FPGA eszközök).

Def. Konfigurálás – mielőtt az eszközt használni szeretnénk egy speciális (manapság általában JTAG szabványú) programozó segítségével fel kell programozni: le kell tölteni a konfigurációs állományt. A programozás a legtöbb PLD esetében a programozható összeköttetések típusától függően azok beállításával történik. A programozható összeköttetésekben a következő lehetséges alkatrészek találhatóak:

- Biztosíték (Fuse): átégetésük után nem visszafordítható a programozási folyamat (OTP). Korábban a PAL eszközök népszerű kapcsoló elemeként használták.
- Antifuse technológia: (OTP), az antifuse-os kristályszerkezetű kapcsoló elem „átolvasztása” után egy nagyon stabilan működő összeköttetést kapunk, amely sajnos szintén nem visszafordítható folyamatot jelent. A technológia drága, az előállításához szükséges maszk-retegek nagy száma miatt, nagyon jó zavarvédetség elérése érdekében használják (pl. úrkutatás).
- SRAM cella + tranzisztor: tetszőlegesen programozható (FPGA-k esetén legelterjedtebb kapcsolás technológia), az SRAM-ban tárolt inicializáló értéktől függően vezérelhető a tranzisztor gate-elektrodája.
- SRAM cella + multiplexer: tetszőlegesen programozható az SRAM cellában tárolt értéktől függően (kiválasztó jel) vezérelhető a multiplexer.
- Lebegő kapus tranzisztor (Floating Gate) technológia: elektromosan tetszőlegesen programozható, a mai EEPROM/Flash technológia alapja.

A mai modern, nagy integráltsági fokú eszközök estén (CPLD, FPGA) azonban már nem csak egyszerű programozható összeköttetés-hálózatról beszélhetünk, hanem konfigurálható logikai elemekből alkotott blokkokat, illetve bizonyos FPGA-k esetén a rendelkezésre álló dedikált erőforrásokat (memória blokkok, illetve szorzó áramkörök) is programozhatunk.

A programozható logikai eszközök (10.9 ábra) közül a következő fontosabb **típusokat** vizsgáljuk meg röviden:

1. Makrocellás típusok

(kis/közepes integráltsági fokkal):

- a) PLA: Programozható logikai tömb, OTP – általában biztosítékot használt
- b) PAL: Programozható ÉS/tömb logika (sajnos nem a legáltalános elnevezéssel definiálták), OTP – általában biztosítékot használt
- c) GAL: Generikus tömb logika: többször törölhető és programozható

(nagy integráltsági fokkal):

- d) CPLD: Komplex programozható logikai eszköz

2. Programozható kapu-áramkörök

(nagyon-nagy integráltsági fokkal):

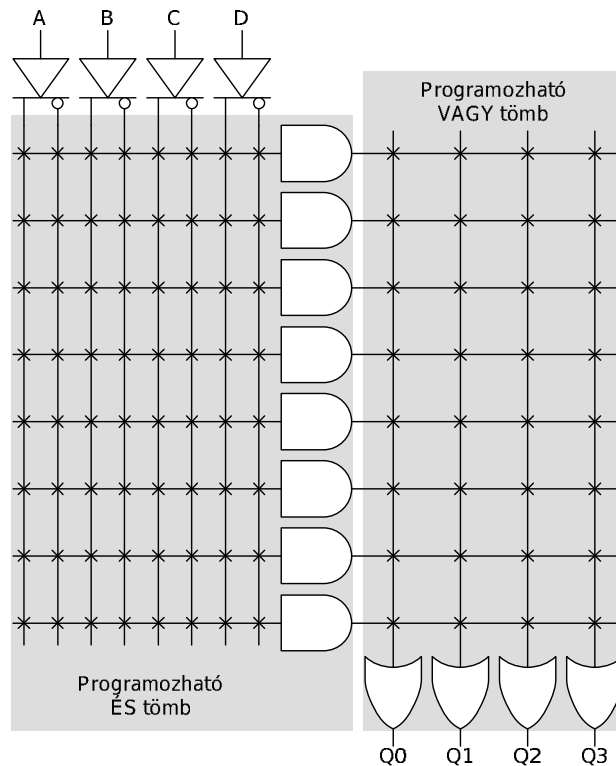
- FPGA: Felhasználó által tetszőlegesen programozható (újraconfigurálható) kapu áramkör

Makrocellás típusok

A *makrocellás* elnevezés abból származik, hogy a programozható logikai tömbök hálózatát egyrészt programozható logikai részekre (ÉS / VAGY tömbök), illetve kimeneti logikai cellákra, úgynevezett „makrocellákra” osztották. A makrocellák általában néhány logikai áramkört (inverter, multiplexert) tartalmaznak, valamint egy elemi D-tárolót. Ezáltal ha szükséges, regiszterelni lehet a kimeneti függvényt, majd pedig vissza lehet csatolni annak értékét a bemenetre.

a.) PLA (Programmable Logic Array): Programozható Logikai Tömb

1970-ben, a TI (Texas Instruments) által kifejlesztett eszköz mindkét részhálózata (ÉS, illetve VAGY tömb) programozható összeköttetéseket tartalmazott, amelyek segítségével tetszőleges mintermek tetszőleges VAGY kapcsolata előállítható (DNF alakot), ezáltal bármilyen kombinációs logikai hálózat realizálható volt (természetesen adott bemenet, ill. kimenet szám mellett). A programozható ÉS / VAGY tömbökben úgynevezett „programozható kapcsolók” vannak elhelyezve a horizontális/vertikális vonalak metszéspontjában. Amennyiben a Q_n kimenet(ek)re tárolókat kötünk (pl. egyszerű D tárolót), majd pedig visszacsatoljuk a programozható logikai hálózat bemenete(i)re akár egy sorrendi hálózati viselkedést is meghatározhatunk.



10.11 ábra: PLA (Programozható Logikai Tömb) általános felépítése

Programozása biztosítókkal:

Az összeköttetés-mátrix metszéspontjaiban kis biztosítékok (fuse) helyezkednek el. Gyárilag általában logikai '1'-est definiál, tehát vezetőképes. Ha valamilyen áramköri programozó eszközzel, feszültséget kapcsolunk rá → átégethető: tehát szigetelővé (nem-vezető) válik, és logikai '0'-át fog reprezentálni.

A biztosíték átégetése, csak egyszer lehetséges, utána már csak a programozott állapotot fogja tárolni (OTP).

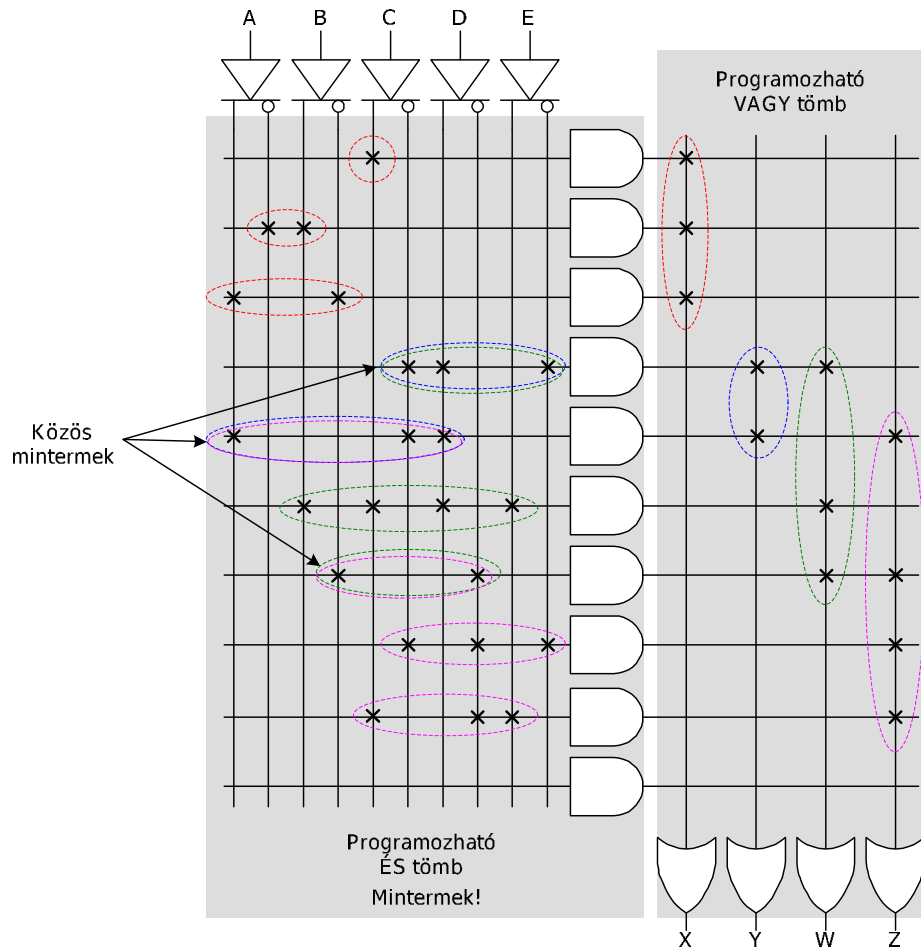
Példa:

PLA áramkört alkalmazva realizálja a következő 5 bemenetű (A,B,C,D,E) és 4 kimenetű (X,Y,W,Z) logikai hálózatot, ha adottak az alábbi kimeneti függvények:

$$\begin{aligned}
 X &= C + \bar{A} \cdot B + A \cdot \bar{B} \\
 Y &= \bar{C} \cdot D \cdot \bar{E} + A \cdot \bar{C} \cdot D \\
 W &= \bar{C} \cdot D \cdot \bar{E} + B \cdot C \cdot D \cdot E + \bar{B} \cdot \bar{D} \\
 Z &= A \cdot \bar{C} \cdot D + \bar{B} \cdot \bar{D} + \bar{C} \cdot \bar{D} \cdot \bar{E} + C \cdot \bar{D} \cdot E
 \end{aligned}$$

Ahol lehetséges, először érdemes kiemelni a közös mintermeket és csak egyetlen egyszer kell megvalósítani az ÉS-tömb szintjén, amelyeket a különböző kimenetekhez a VAGY tömb szintjén rendelhetünk. A függvényminimalizáláshoz természetesen használhatóak a korábbi fejezetekben

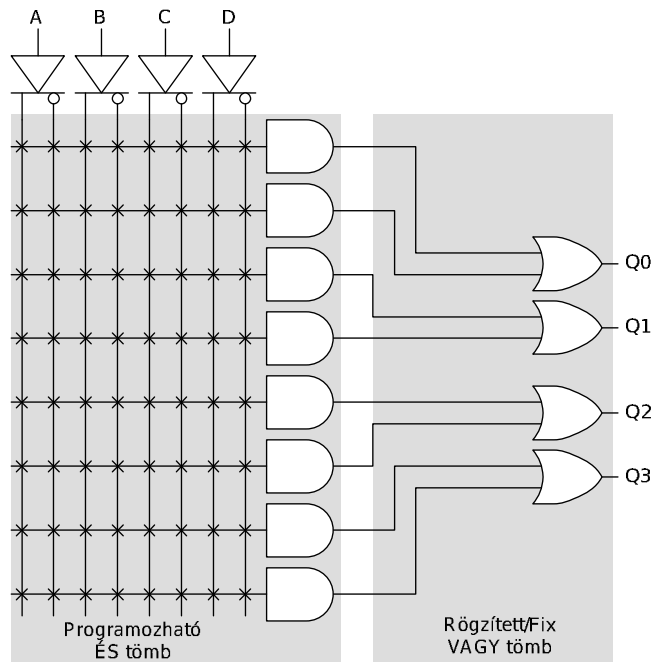
ismertetett függvény-optimalizáló eljárások: pl. grafikus minimalizálás, vagy Quinne-McCluskey módszer. A megvalósítás PLA áramkör felhasználásával az alábbi 10.11 ábrán látható:



10.12 ábra: Feladat megvalósítása PLA áramkörön

b.) PAL (Programmable And/Array Logic): Programozható ÉS/ Tömb logika

Elsőként, 1978-ban az MMI (Monolithic Memories Inc.) jelent meg ilyen programozható eszközökkel, majd pedig későbbi jogutódja a Lattice Semiconductor, illetve az AMD a 80'-as évek végén. A PAL hálózatban a programozható részt az ÉS tömb jelenti, míg az VAGY tömb fix/rögzített. Így a tetszőleges mintermeknek csak egy véges kombinációja (VAGY) állítható elő: a lehetséges kimeneti függvények variálhatóságából veszítünk, cserébe viszont a VAGY részek dedikált útvonalainak jelterjedési sebessége nagyobb, míg az eszköz mérete kisebb és ezáltal olcsóbb is lesz. Ezáltal a metszéspontokban kevesebb kapcsoló szükséges („gyorsabb”, mint a PLA). Hasonlóan a PLA-khoz, amennyiben a Qn kimenet(ek)re tárolókat kötünk (pl. egyszerű D tárolót), majd pedig visszacsatoljuk a programozható PAL logikai hálózat bemenete(i)re akár sorrendi hálózati viselkedést is könnyen valósíthatunk.



10.13 ábra: PAL (Programozható ÉS Logika) áramköri felépítése

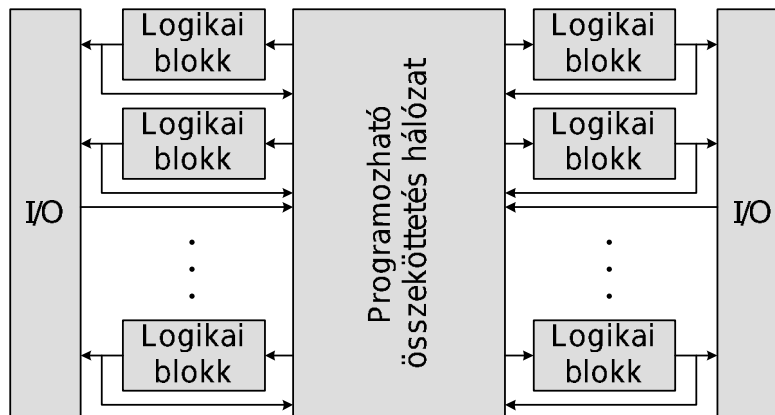
c.) GAL (Generic Array Logic): Általános tömb logika

1985-ben a Lattice Semiconductor [LATTICE] fejlesztette ki elsőként, amely a PAL-nak egy továbbfejlesztett változatát képviseli. Ugyanolyan belső struktúrával rendelkezik, mint egy PAL áramkör, azonban többször programozható: tehát törölhető és újraprogramozható eszköz. EEPROM technológiát (lásd. lebegő-gate) alkalmaz. Később a National Semiconductor, és AMD is megjelent saját GAL sorozataival a piacon.

d.) CPLD (Complex Programmable Logic Devices): Komplex-programozható Logikai eszközök

Valójában átmenetet képeznek a kis/közepes integráltsági fokú makrocellás PLD-k GAL/PAL áramkörei, illetve a nagy integráltsági fokú FPGA kapu-áramkörök között. A GAL/PAL áramköröktől architektúráisan annyiban különbözik, hogy ki lett bővítve: nem egy-, hanem több logikai cellamátrixot tartalmaz, amelyek konfigurálható blokkok reguláris struktúrájában vannak elrendezve. A mai modern FPGA áramköröktől viszont az különbözteti meg felépítésben, hogy nem tartalmaz dedikált erőforrásokat (pl. szorzók, memória blokkok).

A teljesség igénye nélkül a legnagyobb gyártók, amelyek jelenleg is aktív szereplői a CPLD-k piacának a következők: Xilinx [XILINX], Altera [ALTERA], Lattice Semiconductor [LATTICE], Atmel [ATMEL] stb.



10.14 ábra: CPLD általános felépítése

A CPLD-kben található Logikai Blokk-ok (makrocellák)

- egyrészt logikai kapuk tömbjeit (hasonlóan a PAL/GAL áramkörök felépítéséhez – DNF alak),
- másrészt regisztereket (D-tárolókból) tartalmaznak a logikai tömbök által előállított kimenetek átmeneti tárolásához, valamint
- multiplexereket, mellyel a programozható összeköttetés hálózatra, vagy I/O blokkok celláihoz lehet továbbítani a Logikai Blokkok által előállított kimeneti értékeket. Ezáltal nemcsak logikai kombinációs hálózatokat, hanem sorrendi hálózatokat is egyszerűen megvalósíthatunk CPLD-k segítségével

A CPLD-kben található Programozható összeköttetés hálózat vagy teljes összeköttetést (mindenki-mindekivel), vagy részleges összeköttetést (valamilyen struktúra szerint, pl. bemenettel – kimenettel, főként régi CPLD típusok esetén) biztosít az egyes blokkok között. Kikapcsoláskor a CPLD konfigurációs memóriája megtartja értékét (non-volatile típus), ezért nem kell egy külső pl. ROM memóriát használni az inicializációs minták tárolásához, bekapcsoláskor ezek automatikusan betöltésre kerülnek. A CPLD-eket közkedvelten alkalmazzák különböző interfészek jeleinek összekapcsolásához (glue-logic), amennyiben a jeleken átalakításra is szükség van, továbbá árak az FPGA-k árainál jóval kedvezőbbek.

FPGA kapu-áramkörök

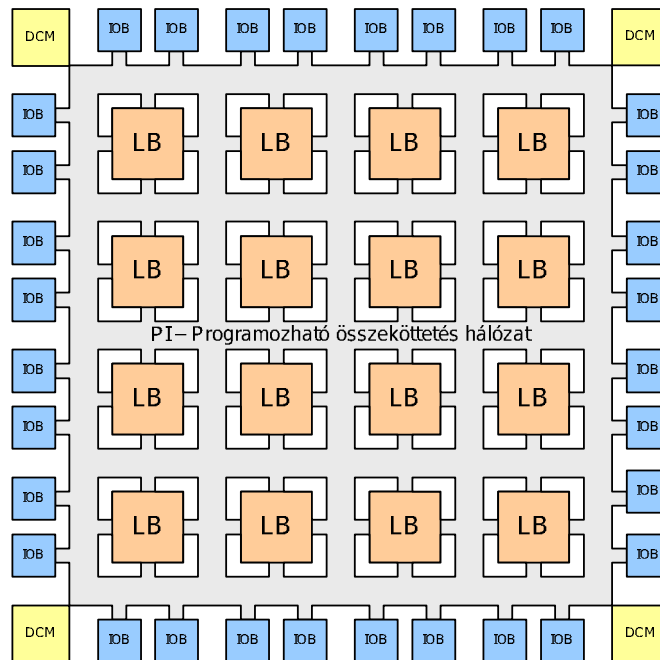
Az FPGA áramkörök (Field-Programmable Gate Array), magyarul „újraprogramozható kapu áramkörök” architektúráisan tükrözik mind a PAL, ill. CPLD felépítését, komplexitásban pedig a CPLD-eket is felülmúlják. Nagy, illetve nagyon-nagy integráltsági fokkal rendelkezik: ~10.000 - ~100.000.000 ekvivalens logikai kaput is tartalmazhat gyártótól, és sorozattól függően.

FPGA-k általános felépítése a 10.16-os ábrán látható. A következő főbb generikus komponensekből épül fel:

- LB: Konfigurálható Logikai Blokkok, amelyekben LUT-ok (Look-up-table) segítségével realizálhatók például tetszőleges, több bemenetű (ált. 4 vagy 6), egy-kimenetű logikai függvények. Ezek a kimeneti értékek szükség esetén egy-egy D flip-flopban tárolhatók el; továbbá multiplexereket, egyszerű logikai kapukat, és összeköttetéseket is tartalmaznak.
- IOB: I/O Blokkok, amelyek a belső programozható logika és a külvilág között teremtenek kapcsolatot. Programozható I/O blokkok kb. 30 ipari szabványt támogatnak (pl. LVDS, LVCMOS, LVTTTL, SSTL stb.).
- DCM/PLL: Digitális órajel menedzselő áramkör, amely képes a külső bejövő órajelből tetszőleges fázisú és frekvenciájú belső órajel(ek) előállítására
- PI: az FPGA belső komponensei között a programozható összeköttetés hálózat teremt kapcsolatot (lokális, globális és regionális útvonalak segítségével, melyeket konfigurálható kapcsolók állítanak be)

Dedikált erőforrások a következők (FPGA típusoktól és komplexitástól függően):

- BRAM: Blokk-RAM memóriák, melyek nagy mennyiségű (~×100Kbyte – akár ~×10Mbyte) adat/utasítás tárolását teszik lehetővé – FPGA típusától függően
- MULT / vagy DSP Blokkok: beágyazott szorzó blokkokat jelentenek, amelyek segítségével egyszerűbb szorzási műveletet, vagy a DSP blokk esetén akár bonyolultabb DSP MAC (szorzás-akkumulálás), valamint aritmetikai (kivonás) és logikai műveleteket is végrehajthatunk nagy sebességgel – szintén az FPGA kiépítettségétől függően
- Beágyazott processzorok: a mai modern FPGA-kon található még begyázott dedikált, ún. hard processzor mago(ka)t, vagy tetszés szerint konfigurálható ún. szoft-processzor mag(ok) is.



10.15 ábra: FPGA kapu áramkörök általános felépítése

A mai modern FPGA-k a nagyfokú flexibilitásukkal, nagy számítási teljesítményükkel, és gyors prototípus-fejlesztési – ezáltal olcsó kihazatali (piacra kerülési) költségükkel – igen jó alternatívát teremtenek a mikrovezérlős, illetve DSP alapú implementációk helyett (pl. jelfeldolgozás, hálózati titkosítás, beágyazott rendszerek, stb.). Fejlődésüket jól tükrözi a mikroprocesszorok és az FPGA technológia fejlődési üteme között fennálló nagyfokú hasonlóság a méretcsökkenés Moore-törvényének megfelelően.

11. Hazárd jelenségek

Hazárd jelenségek háttére: bevezetés

A korábbiakban tárgyalt fejezetekben a kombinációs logikai hálózatokban lévő kapuk késleltetését, illetve az összeköttetések/vezetékek jelterjedési késleltetését az egyszerűség végett nem vettük figyelembe. Azt feltételeztük, hogy a bemeneti jelek egyszerre érkeznek meg, és a kimeneti érték ezzel egyidejűleg jelenik meg (végtelenül rövid idő alatt – 0 ns /ps nagyságrendben történik).

A valóságban azonban ezeknek a késleltetési viszonyoknak nagyon fontos befolyásoló, ezért nem elhanyagolható szerepük van a digitális áramkörök (akár logikai, akár szekvenciális) működésére, amelyeket még a tervezés során ki kell küszöbölni, vagy meg kell szüntetni.

Kombinációs logikai hálózatok (K.H.) esetén a hazárdoknak alapvetően három fajtáját különböztetjük meg:

- Statikus,
- Dinamikus,
- Funkcionális.

A szekvenciális hálózatok (S.H.) esetén két további hazárd jelenséget is meg kell vizsgálni, de ezek ismertetésére most nem térünk ki:

- Rendszer hazárd (kritikus versenyhelyzet néven is ismert),
- Lényeges hazárd.

Def.: A bemeneti kombináció változásakor az egyes jelek terjedésében mutatkozó különböző késleltető hatások átmenetileg olyan kimeneti kombináció(ka)t hozhatnak létre, amelyek zavart okozhatnak a hálózat működésében. E hatások veszélyességét fokozza, hogy a jelterjedési késleltetéseket előre pontosan megadni nem lehet, és nagyban függ a belső/külső környezeti feltételektől (pl. hőmérséklet, öregedés stb. paraméterektől). Az ilyen hibajelenségeket a rendszertelen és véletlenszerű jellegük miatt **hazárdjelenségeknek** nevezzük. A hazárdok lehetnek késleltetés okozta nem-kívánt kimenetek, állapotok. Hazárd alakulhat ki, ha egy kapu kimenete a bemenetek változásához képest csak véges időn belül változik (szilícium lapkán lévő elektron-, és lyukvezetés következtében).

Cél: tervezéskor törekedni kell a kiküszöbölésükre

Hazárdok kialakulása:

a.) Jelterjedési (propagation delay) vagy „megszólalási” késleltetés:

A logikai kapu bemeneteinek és a kimeneteinek változása közötti időkülönbség miatt (bár rövid, de véges tranziens idő alatt változik meg). Példa: egy TTL 74LS eszközöknél, 1-gates kapu esetén a tipikus jelterjedési késleltetés kb. 5ns.

Függhet:

- Jelalak a bemeneten (waveform)
- Hőmérséklet
- Kimenet terhelése (output loading – Fan-out)
- Disszipált teljesítmény (operating power)
- Logikai eszköz típusa (type / device family)

b.) Összeköttetési (interconnection delay) késleltetés:

A logikai kapukat összekötő vezetéken lévő véges jelterjedés miatt alakulhat ki. Példa: egy ~20 cm/ns sebességű jelátvitel az elektromos vezetéken (azaz a vezetéken megtett út és időegység viszonya)

- bizonyos vezetékosszúság felett léphet fel akkor, ha gyors a jelünk (rövid felfutási idővel rendelkezik)
- Szórt kapacitás, ill. induktivitás (olyan mintha a jel a vezetéken egy késleltető áramkörön haladna keresztül). Ekkor tápvonal hatás jelentős, és ún. tápvonal modellt kell használni (egyébként koncentrált paraméterű a modellt).

Technológia fejlődésének hatása a késleltetésekre:

- Az építőelem készlet technológiai fejlődésével (integrációs fok növekedésével SSI → VLSI módszerek) a kapuk jelterjedési késleltetése egyre inkább összemérhető a vezetékek jelterjedési késleltetésével.
- KATALÓGUS: kapu építőelem leírásokban általában a min. / tipikus (nominális) / maximális jelterjedési értékek is adottak.

Switching Characteristics

at $V_{CC} = 5V$ and $T_A = 25^\circ C$

Symbol	Parameter	$R_L = 2\text{ k}\Omega$				Units
		$C_L = 15\text{ pF}$		$C_L = 50\text{ pF}$		
		Min	Max	Min	Max	
t_{PLH}	Propagation Delay Time LOW-to-HIGH Level Output	3	10	4	15	ns
t_{PHL}	Propagation Delay Time HIGH-to-LOW Level Output	3	10	4	15	ns

11.1 ábra TTL SN74LS00 típusú 4-elemű 2-bemenetű NAND kapukat tartalmazó IC kapcsolási karakterisztikája (adatlap részlet [SN74LS00])

a.) Statikus hazárd

Def.: Kétszintű ($l = 2$) digitális logikai kombinációs hálózatokban jöhet létre, ahol adott két szomszédos bemeneti kombináció (az előző és az új bemeneti kombinációk Hamming-távolsága 1). Jelölje őket rendre X és X' , amelyre a kimeneti függvényérték azonos $F(X) \equiv F(X')$.

Ha a bemeneti kombináció egyik szomszédról a másikra változik, mialatt a kimenetén átmenetileg F^* érték jelenik meg, amely $F^* \neq (F(X) \equiv F(X'))$ -al, vagyis a kimenetén $F(X) \rightarrow F^* \rightarrow F(X')$ változás következik be, a késleltetési viszonyoktól függően a hálózatban **statikus hazárd** alakul ki.

Def.: Statikus hazárdmentesítés

A kétszintű (ÉS-VAGY) diszjunktív hálózat pontosan akkor tekinthető statikus hazárdtól mentesnek, ha az logikai '1'-es kimeneteket előállító bemeneti kombinációk (mintermek) közül bármely két szomszédos mintermhez található egy olyan ÉS kapu, amelynek kimenete mindkét szomszédos bemeneti kombináció (minterm) esetén '1'. Azaz bármely két szomszédos mintermhez található legalább egy olyan implikáns, amely mindkét mintermet lefedi.

Statikus hazárd **megszüntetése:** Elsőként az 1.szinten az ÉS kapu(k) segítségével a szomszédos mintermek összevonását kell megvalósítani, majd pedig a 2. szinten ez(eke)t kell VAGY kapcsolatba hozni, így módosítva a hálózatot.

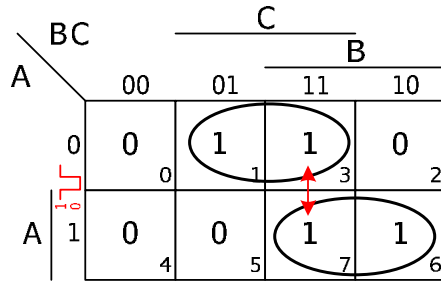
Példa: Statikus hazárd

Adott a következő F logikai függvény legegyszerűbb DNF alakja. Vizsgáljuk meg, és ha szükséges hazárdmentesítsük a hálózatot!

$$F^{n=3} = \sum_{i=0}^{2^3-1} (1,3,6,7) = A \cdot B + \bar{A} \cdot C$$

Megoldás:

Rajzoljuk fel az F függvénynek megfelelő Karnaugh táblát (legegyszerűbb DNF alak szerint):



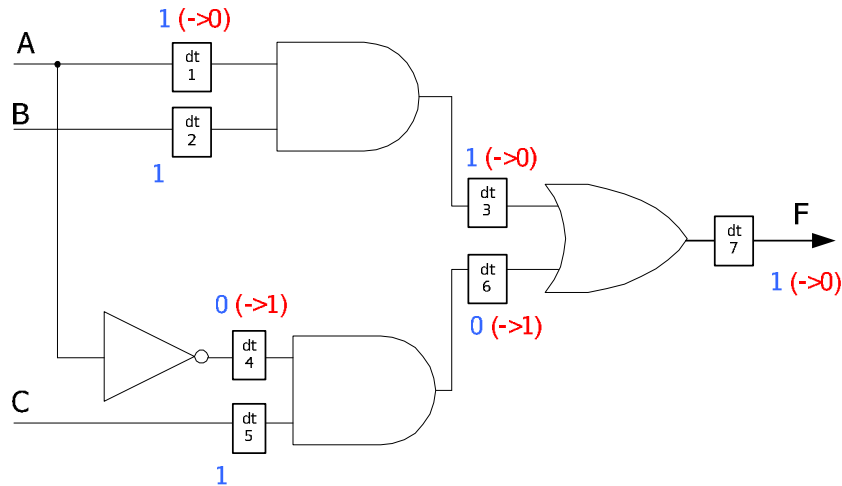
11.2 ábra F függvény legegyszerűbb DNF alakját megvalósító Karnaugh tábla

Látható, hogy két 2-es összevonást kapunk, amelyek között nincs lefedés a (3,7) mintermeken. Amennyiben az A független logikai változó értéket vált (akár A: '0' → '1', akár A: '1' → '0'), tehát A, B, C változókon szomszédos változás következik be, akkor statikus hazard alakulhat ki a (3,7) mintermek között.

Rajzoljuk fel ekkor a 11.2 ábrán lévő Karnaugh táblának megfelelő kapcsolási rajzot, és vizsgáljuk meg a hálózat működését elsőként a késleltetések nélkül, majd pedig a késleltetések figyelembe vételével a hazard szempontjából. Az elvi logikai rajzot kiegészítjük koncentrált késleltetésekkel (jelölve Δt_i -kel). Azt feltételezzük, hogy a hálózatot szomszédos bemeneti változás éri (egyetlen bemeneti jel változik meg). Mi játszódik le ekkor a hálózatban?

I. eset – Késleltetési viszonyok figyelembe vétele nélkül:

Az elvi logikai rajzon a koncentrált késleltetések most $\Delta t_i = 0$, minden i esetén, tehát azt feltételezzük, hogy végtelenül gyorsan terjed a jel az összeköttetéseken.



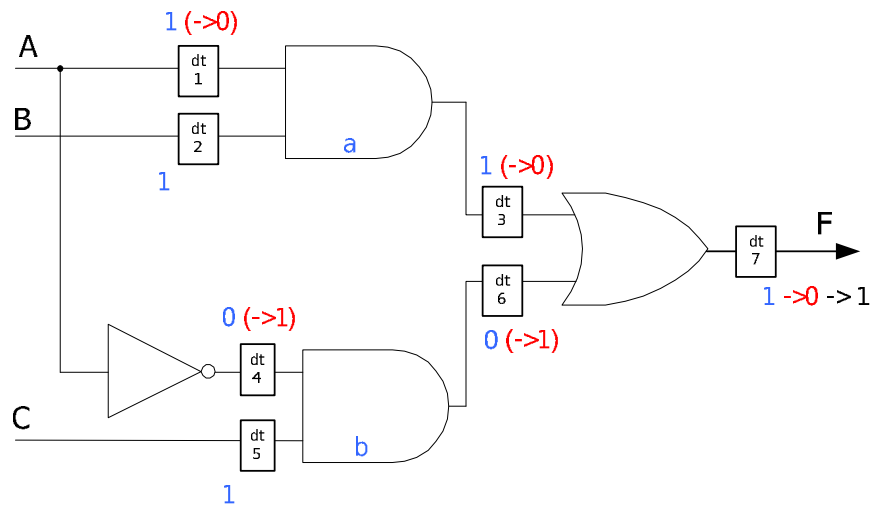
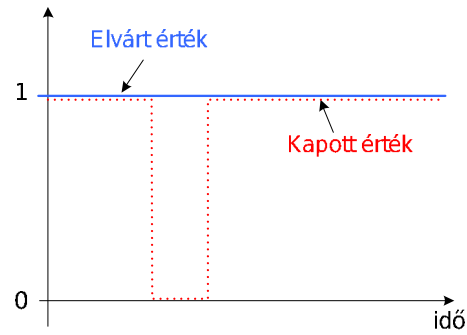
11.3 ábra F függvény kapcsolási rajza a késleltetési viszonyok figyelembe vétele nélkül

Amikor a bemeneten ABC=111, majd a vele szomszédos változás ABC=011 következik be (tehát A: '0' → '1' lesz). A kimeneten mindkét esetben F='1' -et várnánk.

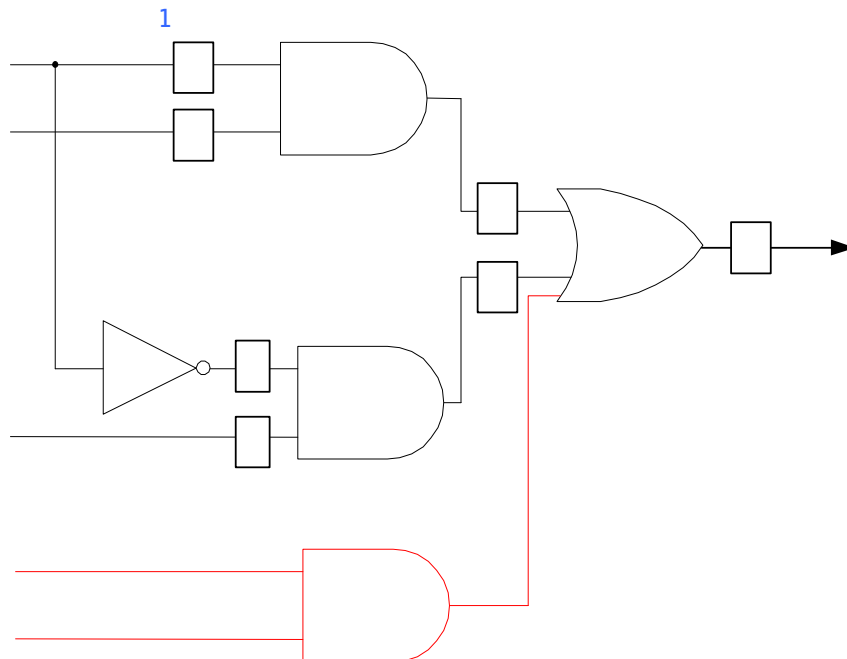
II. eset – Késleltetési viszonyok figyelembe vételével:

Tegyük fel, hogy $(dt1+dt3) < (dt4+dt6)$. Ekkor a VAGY kapunak a felső 'a'-val jelölt ÉS ága előbb hajtódik végre (értékelődik ki), mint az alsó 'b'-vel jelölt ÉS ág. Ez azt jelenti, hogy előbb megy végbe a felső ágon az $1 \rightarrow 0$ (A) átmenet, mint az alsó ágon a $0 \rightarrow 1$ (inverter) változás. Így a felső kapu bemenetén előbb vált '0'-ra, így előáll a '00' párosítás, amelyre kis ideig a kimenet is F='0' lesz. Csak

$(dt_4+dt_6) - (dt_1+dt_3) > 0$ idővel később lesz megint a várt $F=1$. Tehát a kimeneten egy $1 \rightarrow 0 \rightarrow 1$ jelváltás megy végbe, ami hibát, amely a definíció szerint **statikus hazárdot** jelent!



11.4 ábra F függvény kapcsolási rajza a késleltetési viszonyok figyelembe vételével



		BC		C		B	
		00	01	11	10		
A	0	0	1	1	0		
	1	0	0	1	1		
		0	1	2			
		4	5	6	7		

11.5 ábra Statikus hazárd megszüntetése: F függvény kapcsolási rajza a késleltetési viszonyok figyelembe vételével és a hazárdmentesítéssel

Megjegyzés: a hazárdmentesített hálózat nem a legegyszerűbb DNF alakot fogja ábrázolni a redundancia miatt. A hazárdmentesített hurok a fenti ábrán pirossal van ábrázolva.

Megjegyzés: Több kimenetű TSH hálózatokban a statikus hazárd minden kimeneten felléphet a szomszédos jelváltozásra. Kiküszöböléséhez a módszer hasonlóan történik, mint egy-kimenetű esetben, de csak kimeneti függvényenként külön-külön vizsgálva kell az összes prímisszorzót megvalósítani. (Azaz az összesített prímisszorzó tábla képzése és lefedése elhagyható – nincs lényeges prímisszorzó). A közös prímisszorzókat elegendő egyszer megvalósítani.

NTSH eset intuitív módszer: próbálgatással előállítani a kétszintű statikus hazárdmentes hálózatot (optimális legnagyobb lefedéseket keresni, miközben vizsgálni kell a lefedések közötti átmeneteken ne alakuljon ki statikus hazárd.)

Megjegyzés: Statikus hazárd több kimenetű hálózatokban

- Teljesen specifikált több-kimenetű hálózatokban a statikus hazárd minden kimeneten felléphet a szomszédos jelváltozások hatására. Kiküszöböléséhez a megismert egy-kimenetű módszert kell alkalmazni, de kimeneti függvényenként külön-külön kell az összes prímisszorzót megvalósítani. A közös prímisszorzókat elegendő egyszer megvalósítani.
- NTSH többkimenetű hálózatok esetében a módszer intuitív, próbálgatással kell előállítani a kétszintű statikus hazárdmentes hálózatot. Optimális lefedéseket keresni, miközben vizsgálni kell a lefedések közötti átmeneteken a statikus hazárdot.

b.) Dinamikus hazárd

Def.: Adott két szomszédos bemeneti kombináció (Hamming távolságuk = 1), jelölje őket X és X' amelyre az elvárt kimeneti függvényérték eltérő $F(X) \neq F(X')$!

Ha a bemeneti kombináció egyik szomszédról a másikra változik ($X \rightarrow X'$), mialatt a kimenetén átmenetileg $F(X) \rightarrow F(X')$ helyett $F(X) \rightarrow F' \rightarrow F(X) \rightarrow F(X')$ jelváltozások játszódnak le, akkor a késleltetési viszonyoktól függően a hálózatban **dinamikus hazárd** van.

Def.: Kialakulásának feltétele. Olyan többszintű ($l > 2$) digitális logikai kombinációs hálózatokban jöhet létre, ahol a statikus hazárd az alacsonyabb hierarchia szinteken nem lett megszüntetve, kiküszöbölve.

Megszüntethető, az alacsonyabb hierarchia szinteken történő statikus hazárd kiküszöbölésével.

Példa: Dinamikus hazárd

Adott a következő F logikai függvény. Vizsgáljuk meg dinamikus hazárd szempontjából, és ha szükséges hazárd-mentesítsük a hálózatot!

$$F^{n=5} = (\bar{A}D + E) \cdot (\bar{A}B + AC)$$

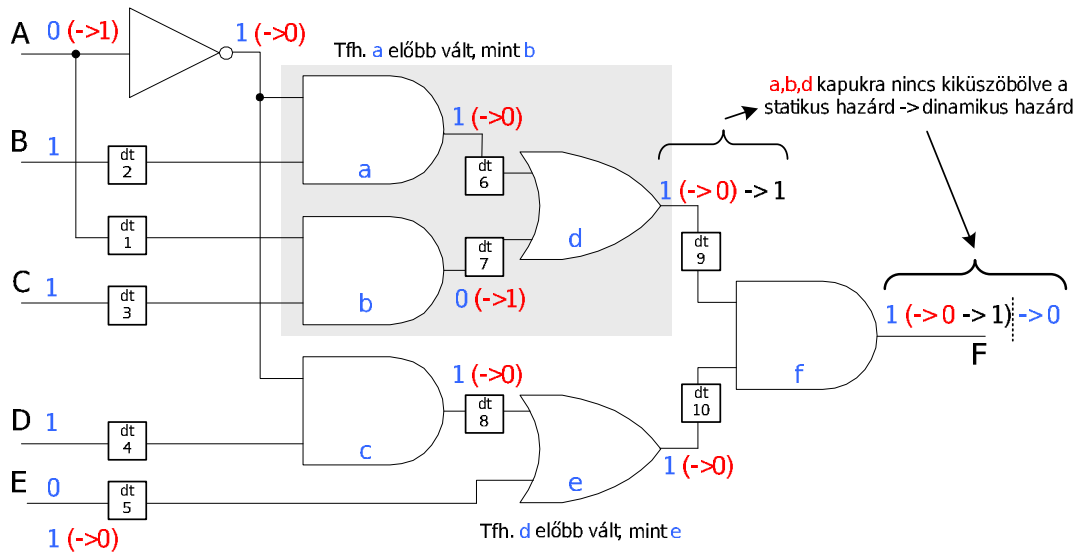
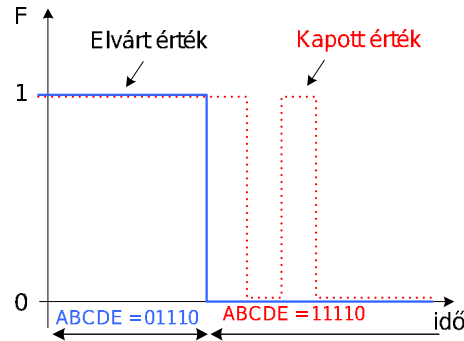
f_1 :szomszédosság!

Megoldás:

A fenti kifejezés egy háromszintű, ÉS-VAGY-ÉS hierarchiában ($I > 2$) valósítható csak meg.

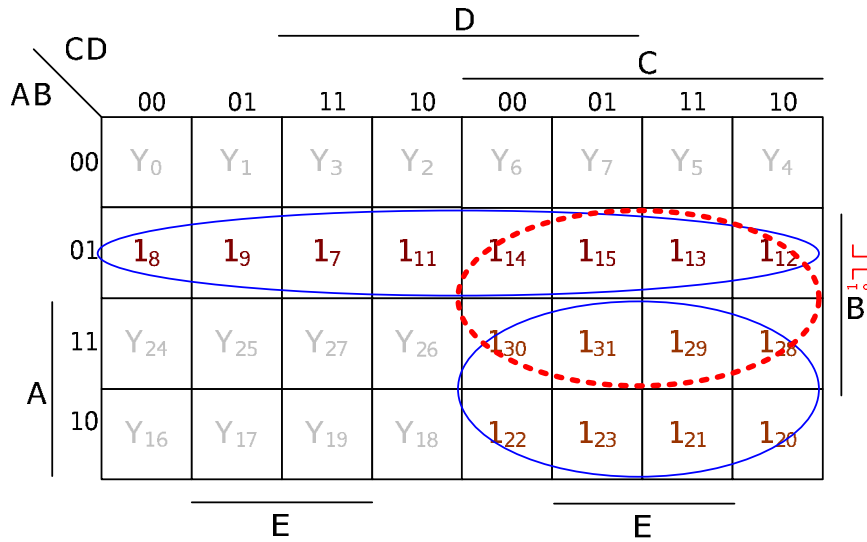
Amikor a bemeneten $ABCDE = \underline{0}1110$, majd a szomszédos változás $ABCDE = \underline{1}1110$ következik be (tehát A: '0' \rightarrow '1' lesz) a kimeneten esetben $F = '1' \rightarrow '0'$ lenne az elvárt érték. Ehelyett, a késleltetési viszonyoktól függően többszörös értékváltozás történik az F függvény kimenetén. Időbeliség:

- Tfh. az ÉS kapuk szintjén a felső ágba 'a' előbb vált (gyorsabb megszólalási idejű), mint 'b' a középső ágba
- Tfh. a VAGY kapuk szintjén szintén a felső ágba 'd' előbb vált, mint 'e' az alsó ágba



11.6 ábra F függvény vizsgálata dinamikusan (statikus) hazárd esetén

Kimeneti logikai függvény Karnaugh táblája a következő:

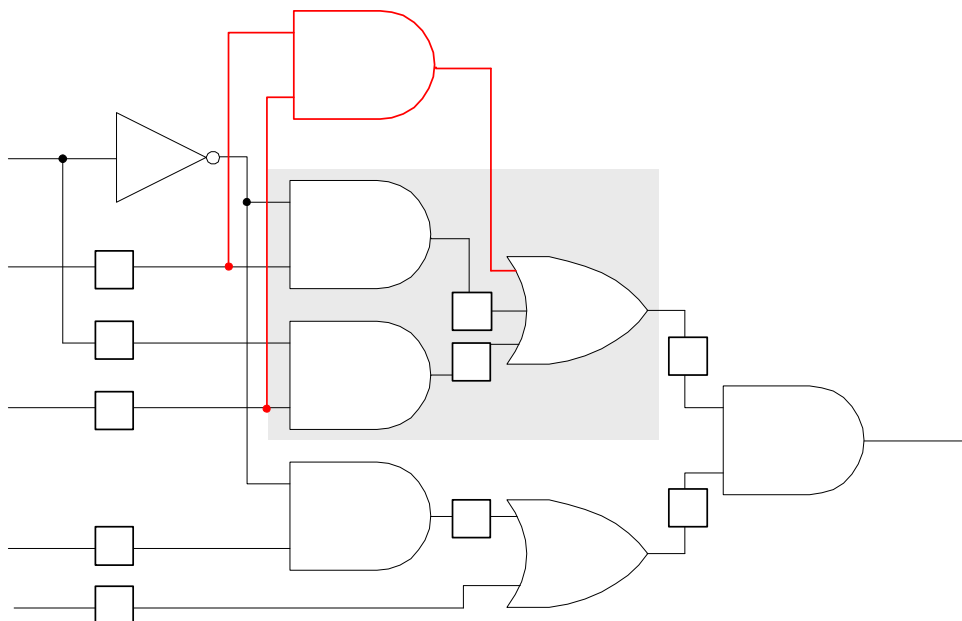


11.7 ábra F függvény szomszédos f_1 alakját megvalósító Karnaugh tábla ($n=5$ változó esetén egy lehetséges felírás)

Dinamikus hazárd megszüntetése: statikus hazárd megszüntetésével hierarchia szintenként történik.

$$F^{n=5} = (\bar{A}D + E)(\bar{A}B + AC + \underbrace{BC}_{\substack{\text{hazárdmentesítő} \\ \text{hurok}}})$$

A megszüntetésnél először szintén az alacsonyabb hierarchia szinten lévő (itt: 'BC' implikáns, amelyet 11.7 / 11.8 ábrán piros szaggatott vonal jelöl) mintermet állítjuk ÉS kapcsolattal, majd pedig az így kapott mintermet az egyel felette lévő VAGY kapuhoz adjuk hozzá.



11.8 ábra Dinamikus hazárd megszüntetése: F függvény kapcsolási rajza a késleltetési viszonyok figyelembe vételével és a hazárdmentesítéssel

c.) Funkcionális hazard

Az eddigi példákban csak szomszédos bemeneti változások esetén vizsgáltuk a hazard jelenségeket (késleltetések időbeli hatását). Mostantól viszont tetszőleges bemeneti (akár nem szomszédos) kombinációváltozásokra is meg kell vizsgálni, milyen változások játszódhatnak le a hálózat kimenetén (Hamming távolság két bemeneti kombináció között > 1), azaz ha egy változó, bármelyik másik (vagy akár több) változóval egyszerre vált értéket. Továbbá például a vizsgált statikus hazard esetén elegendő időt várakozva a kimenet az elvárt (becsült) logikai- és feszültség- értékre áll be. Azonban lehetnek olyan hazard-jelenségek is, amelyek az idő múlásával sem szűnnének meg, ekkor a tervezés szintjén kell beavatkozni (ilyen lesz a funkcionális hazard is, lásd megszüntetés).

Def. A nem szomszédos bemeneti kombinációk változásai, amikor akár több bemeneti változó is egyszerre változhat, **funkcionális hazard** kialakulásához vezethet. A nem szomszédos bemeneti változásokat hálózat egyes részei szomszédos változások sorozataként érzékelik. (Megjegyzés: a későbbiekben tárgyalt aszinkron sorrendi hálózatokban ez nem kívánt állapotátmeneteket okozhat)

Megszüntetés:

I. mód: a hálózatba szándékosan beépített késleltetésekkel úgy kell beállítani a jelterjedési késleltetés értékeit, hogy azok minden lehetséges megváltozásakor csak olyan „közbenső értékek alakuljanak ki” (Példa 1-ben $ABC=000 = 1$), amelyek nem hoznak létre átmeneti hibát.

Puffer (buffer): *páros számú inverter fokozat*. Ez egyszerű és hatékony megoldás, de lassítja a működést.

II. mód: *szinkronizáló órajelekkel ún. „elnyeletni” a hazard jelenséget.* (Példa 2.) Ez a megvalósítási mód túlmutat a kombinációs logikai hálózatok megvalósításán (lásd. Sorrendi logikai hálózatok).

Példa: Funkcionális hazard (megszüntetés I. módszere)

Vegyünk egy 3-változós F függvényt (DNF szerint összevont alak):

$$F^{n=3}(A, B, C) = \sum_{i=0}^7 (0,1,4) = \bar{A} \cdot \bar{B} + \bar{B} \cdot \bar{C}$$

Vizsgáljuk meg a bemeneti változás során az 'A' és 'C' változók értékváltozásait. (Ugyanis előfordulhat, hogy A megváltozása, más esetben C megváltozása jut el előbb bizonyos kapuk bemenetére.)

		BC		C		B	
		00	01	11	10	11	10
A	0	1	1	0	0	1	2
	1	1	0	0	0	4	6

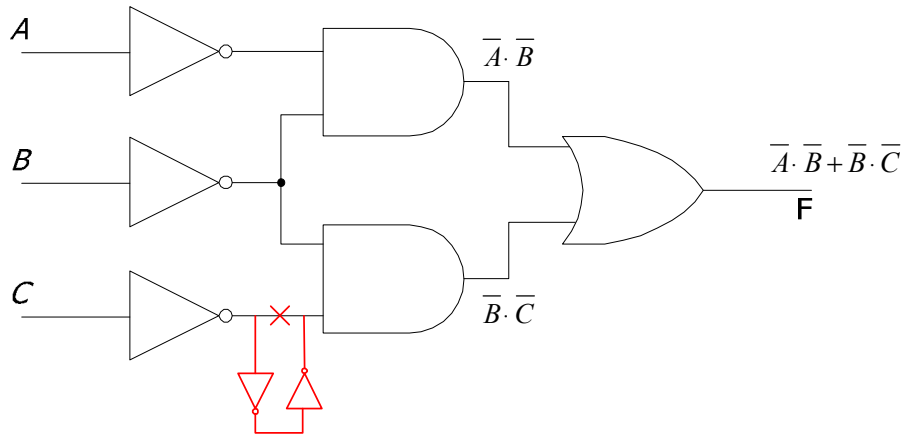
Nem szomszédos változás (változás sorozat): $m4 \rightarrow m1$ ($4 \rightarrow 1$)

- 4: 100
- 1: 001 (4 és 1 nem szomszédok: 100 -> 001)

Az átmenet két lehetséges úton realizálható (változás sorozattal):

- $4 \rightarrow 0 \rightarrow 1 = 100 \rightarrow 000 \rightarrow 001 = '1' \rightarrow '1' \rightarrow '1'$
- $4 \rightarrow 5 \rightarrow 1 = 100 \rightarrow 101 \rightarrow 001 = '1' \rightarrow '0' \rightarrow '1'$ (funkcionális hazard!!/ átmeneti hiba)

Megszüntetése páros számú inverter fokozattal. Abban az ágban kell elhelyezni a késleltető puffert, amelyekben a kimeneti érték megváltozik, miközben szomszédos változások sorozatán keresztül adjuk meg a nem-szomszédos bemeneti változás teljesülését. Így ez most a m4 -> m1 átmenethez kerül, azaz a C változását késleltetjük, azért hogy az A megváltozása realizálódjon előbb.



11.9 ábra Funkcionális hazárd megszüntetése késleltető puffer használatával (páros számú invertert helyezve a késleltetni kívánt ágba). A változása előbb realizálódik, B megváltozása később

Példa: Funkcionális hazárd (megszüntetés II. módszere)

Vegyünk egy 3-változós F függvényt:

$$F^{n=3}(A, B, C) = \sum_{i=0}^7 (1,4) = \bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot \bar{C}$$

Vizsgáljuk meg a bemeneti változás során az 'A' és 'C' változók értékváltozásait. (Ugyanis előfordulhat, hogy A megváltozása, más esetben C megváltozása jut el előbb bizonyos kapuk bemenetére.)

		BC		C		B	
		00	01	11	10	3	2
A	0	0	1	0	0	1	2
	1	1	0	0	0	4	6

Nem szomszédos változás (változás sorozat): m4 → m1 (4 → 1)

- 4: 100
- 1: 001 (4 és 1 nem szomszédok: 100 -> 001)

Az átmenet két lehetséges úton realizálható (változás sorozattal):

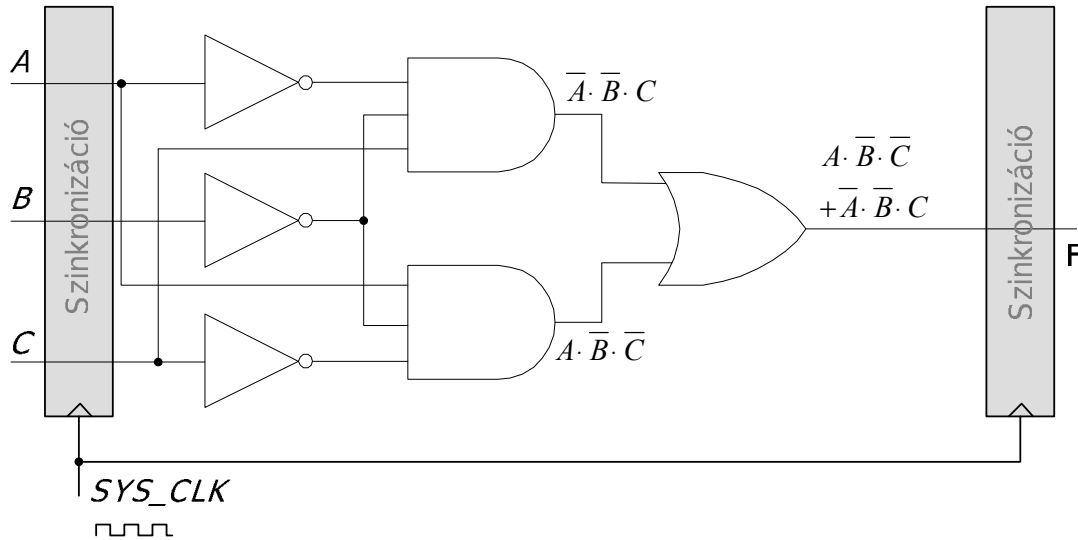
- 4 -> 0 -> 1 = 100 -> 000 -> 001 = '1' -> '1' -> '1' (funkcionális hazárd!!/ átmeneti hiba)
- 4 -> 5 -> 1 = 100 -> 101 -> 001 = '1' -> '0' -> '1' (funkcionális hazárd!!/ átmeneti hiba)

Mint látható, mindkét átmenet funkcionális hazárdot rejt, tehát mialatt az A vagy a C értéket vált az időbeliségtől függetlenül, a kimenet is értéket vált. Ebben az esetben mindkét úton funkcionális hazárdot kapunk, tehát egyszer késleltetéssel (pufferrel) nem küszöbölhető ki.

Megszüntetése:

Szinkronizációval (órajel fel-, vagy lefutó élére működtetjük a beépített tárolókat, amelyek a hálózat be- és kimeneteit tárolják). Bemeneti jelekből SYNC_CLK-val mintát veszünk (felfutó, vagy lefutó élre). Mintavétel a bemeneten → Szinkronizált bemenet → K.H. → Mintavétel a kimeneten → Szinkronizált kimenet (lásd Sorrendi Hálózatok)

Elvi kapcsolási rajz a következő:



11.10 ábra Funkcionális hazard megszüntetése szinkronizáló elemek beépítésével a K.H bemeneteire, illetve kimeneteire

További példák hazard jelenségekre:

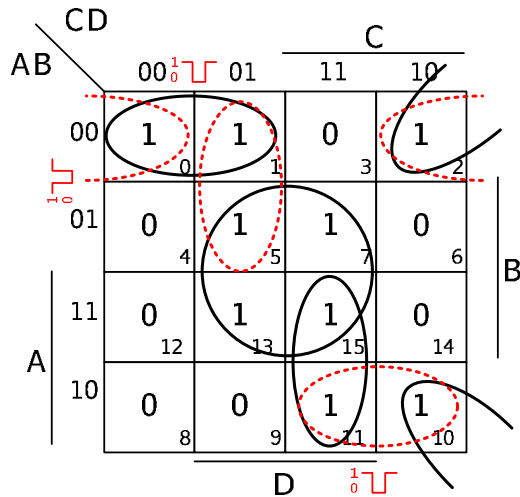
Példa: Statikus/Dinamikus hazard vizsgálata

Hazardmentes-e a következő F függvénnyel, DNF alakban megadott hálózat (statikus, és dinamikus értelemben vizsgálva)? Ha nem, hazardmentesítse, és rajzolja fel a hazardmentes elvi logikai kapcsolási rajtot! Továbbá azt is vizsgálja meg a megvalósításhoz szükséges a *felhasznált kapuk száma* alapján, hogy az F függvény Karnaugh tábla szerint felírt KNF alakja segítségével nem érdemesebb-e a hazardmentesítést elvégezni!

$$F^{n=4} = \sum_{i=0}^{15} (0,1,2,5,7,10,11,13,15)$$

Megoldás: DNF

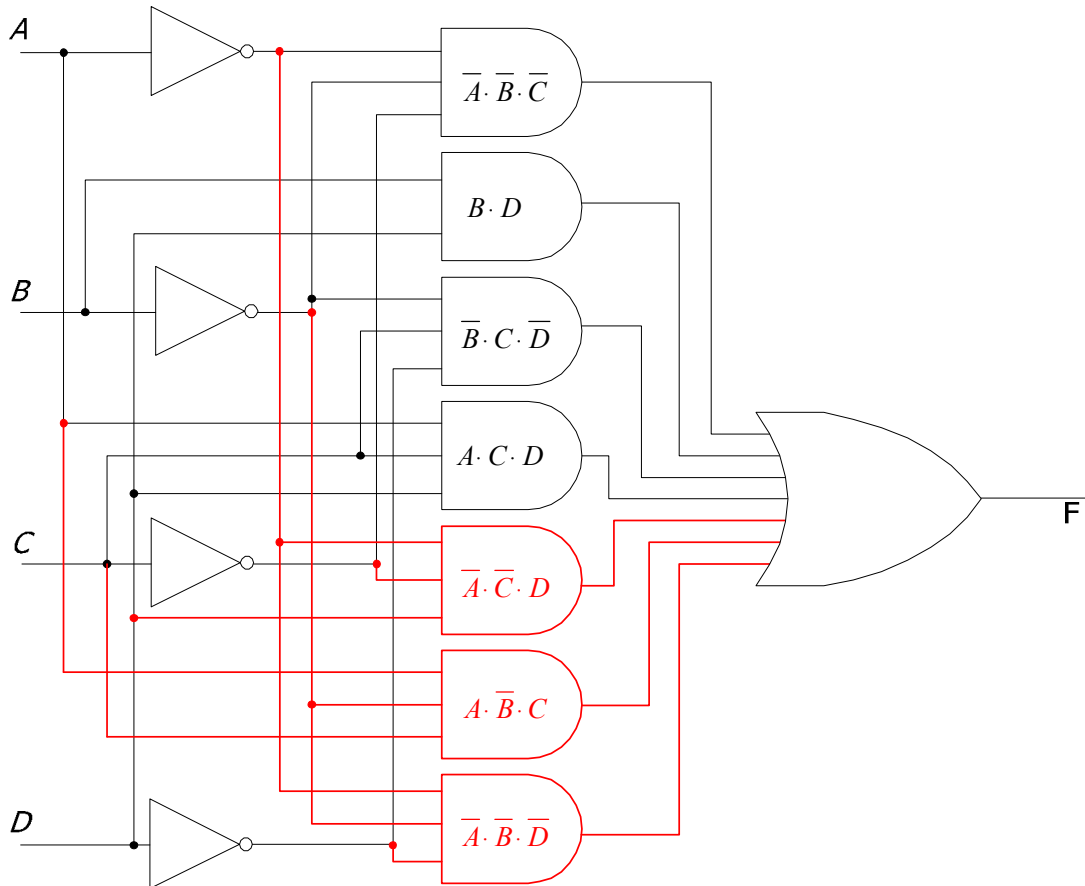
Az F függvényre a következő Karnaugh tábla adható meg (egy lehetséges összevonás szerint):



A lehetséges összevonások alapján a következő egyszerűsített DNF alakot kapjuk:

$$F(A, B, C, D) = \bar{A} \cdot \bar{B} \cdot \bar{C} + B \cdot D + A \cdot C \cdot D + \bar{B} \cdot C \cdot \bar{D} + \bar{A} \cdot \bar{C} \cdot D + A \cdot \bar{B} \cdot C + \bar{A} \cdot \bar{B} \cdot \bar{D}$$

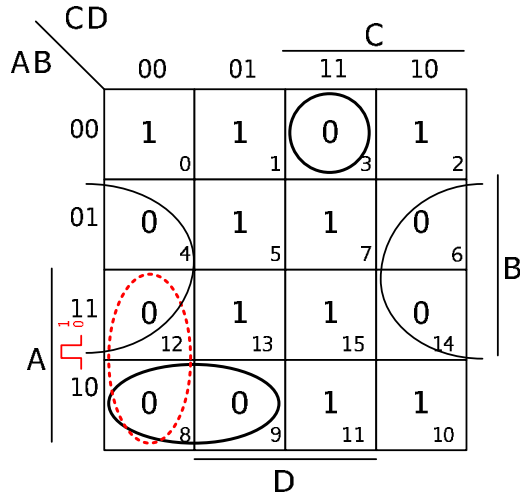
Mivel a teljes hálózat kétszintű ($l=2$) ÉS-VAGY logikai kapcsolással realizálható, ezért a vizsgálat szempontjából csak statikus hazárd alakulhat ki. Azokat a $1 \rightarrow 0 \rightarrow 1$ átmeneteket, amelyeknél statikus hazárd lehetséges, piros szaggatott vonallal jelöltük. A fenti DNF szerinti képletben pedig pirossal vannak jelölve az extra primimplikáns tagok, amelyek a hazárdmentesítő hurkokat definiálják.



11.11 ábra Statikus hazárd megszüntetése: F függvény kapcsolási rajza a késleltetési viszonyok figyelembe vételével és a hazárdmentesítéssel (DNF alak esetén)

KNF:

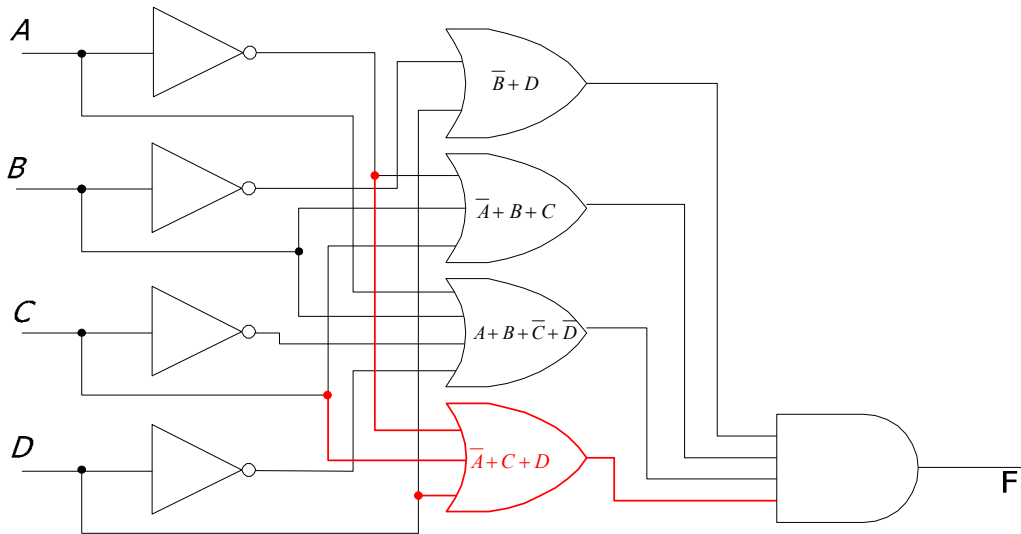
Most pedig vizsgáljuk meg, hogy nem optimálisabb-e KNF alakkal megvalósítani a logikai hálózatot a statikus/dinamikus hazárdok kiküszöbölése, megszüntetése szempontjából.



A lehetséges összevonások alapján a következő egyszerűsített DNF alakot kapjuk:

$$F^{n=4}(A, B, C, D) = (\bar{B} + D) \cdot (\bar{A} + B + C) \cdot (A + B + \bar{C} + \bar{D}) \cdot (\bar{A} + C + D)$$

Mivel a teljes hálózat kétszintű (l=2) VAGY-ÉS logikai kapcsolással realizálható, ezért a vizsgálat szempontjából csak statikus hazárd alakulhat ki. Azokat a 0 -> 1 -> 0 átmeneteket, amelyeknél statikus hazárd lehetséges, piros szaggatott vonallal jelöltük. A fenti KNF szerinti képletben pedig pirossal vannak ismételten jelölve az extra prímisszorzók, amelyek a hazárdmentesítő hurkokat definiálják (kapcsolási rajzon is pirossal van jelölve).



11.12 ábra Statikus hazárd megszüntetése: F függvény kapcsolási rajza a késleltetési viszonyok figyelembe vételével és a hazárdmentesítéssel (KNF alak esetén)

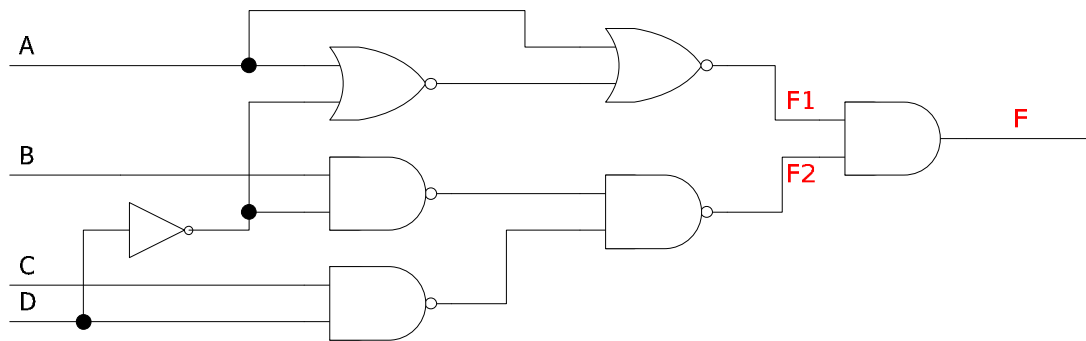
Konklúzió: a lehetséges DNF, KNF alakok közül a statikus hazárd szempontjából (mivel dinamikus nem volt), és az elvi logikai hálózat megvalósíthatóságát szem előtt tartva a **KNF alak használata jobb megoldást eredményez!** (amennyiben nincs egyéb tervezési megszorítás, például a kapubemenetek számának maximumát illetően – terhelhetőség, fan-out stb.)

Példa: Statikus/Dinamikus hazárd vizsgálata

Hazárdmentes-e az alábbi ábrán megadott $F(A,B,C,D)$ logikai kombinációs hálózat (statikus, és dinamikus értelemben vizsgálva)?

a.) Rajzolja fel az F_1 , F_2 , valamint F függvények Karnaugh tábláit is, és jelölje be rajtuk az esetlegesen kialakuló hazárdokat!

b.) Amennyiben a hálózat nem hazárdmentes, adja meg a hazárdmentes alakokat külön-külön F_1 , F_2 , majd pedig F -re is!



11.13 ábra Az F függvény kapcsolási rajza

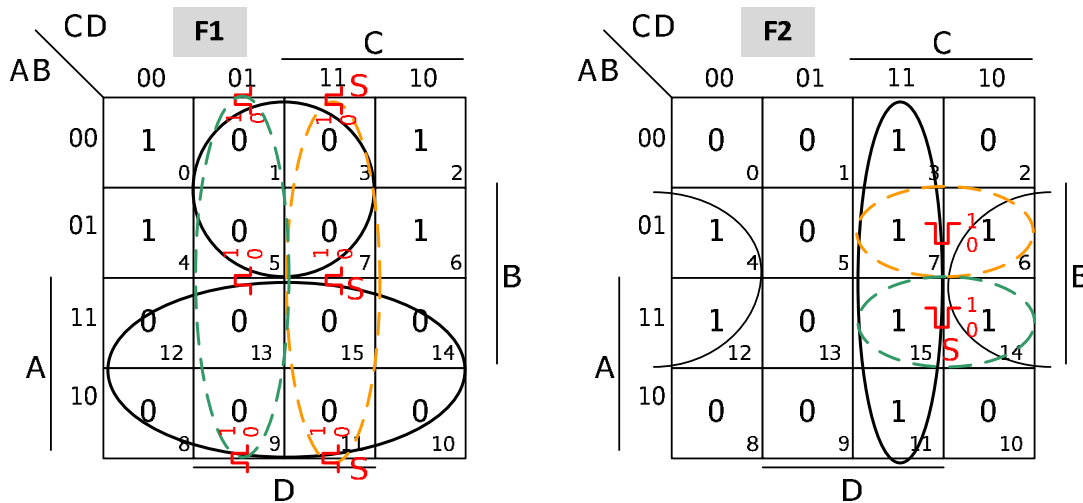
Az F_1 , F_2 részfüggvények, illetve az F kimeneti függvények algebrai alakjai, illetve Karnaugh táblái a következők:

$$0 = F_1 = \overline{\overline{(A + \overline{D})} + A} \stackrel{DeMorgan}{=} (A + \overline{D}) \cdot \overline{A} \quad \text{KNF alak}$$

$$1 = F_2 = \overline{\overline{B\overline{D}} \cdot \overline{CD}} \stackrel{DeMorgan}{=} B\overline{D} + CD \quad \text{DNF alak}$$

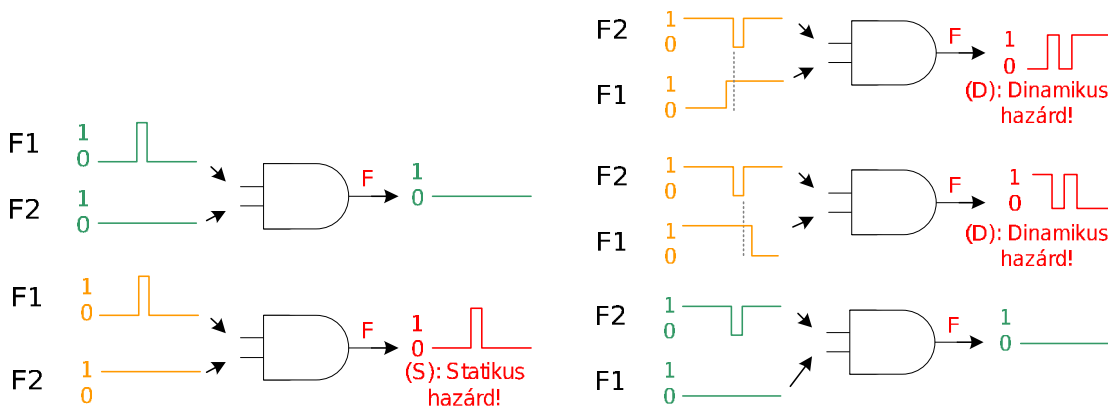
$$F = F_1 \cdot F_2 \quad \text{szorzat függvény}$$

Karnaugh táblák (pirossal jelölve a lehetséges hazárdokat az átmeneteknél, amelyeket meg kell vizsgálni:



11.14 ábra Az F1 és F2 függvény szerinti Karnaugh táblák, jelölve a lehetséges átmeneteket, illetve hazárdokat. A két függvényt külön-külön, illetve együttesen is vizsgálni kell az F kimeneti függvényben

Hazárd vizsgálatok mindkét F1, F2 részfüggvényre, valamint F kimeneti függvényre a következők:



11.15 ábra Az F1 és F2 függvény vizsgálata

A fenti hazárd vizsgálatok alapján az F kimeneti függvény Karnaugh táblája a következő (pirossal jelölve azokat a helyeket ahol S: Statikus hazárd, illetve D: dinamikus hazárd van!):

AB \ CD		F=F1*F2			
		00	01	11	10
A	00	0	0	0	0
	01	1	0	0	1
	11	0	0	0	0
	10	0	0	0	0
		C		D	

The table includes several annotations:

- Red dashed lines and arrows labeled 'S' indicate static hazards at transitions between minterms 3 and 7, 7 and 11, and 11 and 15.
- Yellow dashed lines and arrows labeled 'D' indicate a dynamic hazard at the transition between minterms 5 and 6.
- Gray shading highlights the cells (3,0), (7,0), (11,0), (15,0) and (5,1), (6,1).

11.16 ábra Az F függvény Karnaugh táblája, jelölve a kialakult S: Statikus illetve D:dinamikus hazárdokat

Hazárd-mentesítések F1, F2 algebrai alakjainak megadásával, majd pedig F függvényre is:

$$0 = F1 = \overline{A + D} = \overline{DA}$$

$$1 = F2 = \overline{B\overline{D}} + CD + BC = \overline{\overline{\overline{B\overline{D}} \cdot \overline{CD} \cdot \overline{BC}}}$$

$$F = \overline{AB\overline{D}}$$

Irodalomjegyzék

[ALTERA] Altera hivatalos weboldala: <http://www.altera.com>

[ATMEL] Atmel hivatalos weboldala: <http://www.atmel.com>

[ARATÓP] Dr. Arató Péter: Logikai rendszerek tervezése (Műegyetemi Kiadó, 1992)

[KERESZTP] Dr. Keresztes Péter: Digitális Hálózatok (Széchenyi Egyetem, 2006)

[LATTICE] Lattice Semiconductor hivatalos weboldala: <http://www.latticesemi.com>

[SN74LS00] TI 74LS00 Adatlap (weboldal) <http://www.ti.com/lit/ds/symlink/sn74ls00.pdf>

[SN7447] TI 7447 Adatlap (weboldal): <http://www.ti.com/lit/ds/symlink/sn7447a.pdf>

[XILINX] Xilinx hivatalos weboldala: <http://www.xilinx.com>