# Digitális technika 1

Tantárgykód: VIIIA105
Villamosmérnöki szak, Bsc. képzés

*Készítette: Dudás Márton*

## Bevezető:

A jegyzet a BME VIK első éves villamosmérnök hallgatóinak készült a *Digitális technika 1* tárgyhoz. Tartalma nagyjából lefedi az első félévben elsajátítandó tudást, segítséget nyújt annak megismerésében, és mintapéldákon keresztül mutatja be azt. A jegyzet hibákat tartalmazhat. Kellő forráskritikával olvassuk, és ha gyanús doldokat vélünk felfedezni, járjunk alaposan utána a valóságnak!
A jegyzet nem helyettesíti a tankönyvet, előadásokat és a gyakorlatokat, utóbbiakon az aktív részvétel erősen ajánlott.

## Tartalomjegyzék:


1. fejezet: Ismerkedés a tárggyal ..... 3. oldal
   - A jelfeldolgozás alapjai..... 4. oldal
   - Számrendszerek ..... 5. oldal
   - Boole algebra, logikai függvények..... 8. oldal
   - Kanonikus függvényalak..... 9. oldal
   - Termek..... 10. oldal
   - A kombinációs és sorrendi hálózatok bevezetése: ..... 12. oldal

2. fejezet: Kombinációs hálózatok ..... 13. oldal
   - Függvény felírása szöveges feladat alapján..... 14. oldal
   - Függvényalakok egyszerűsítése ..... 18. oldal
   - Grafikus minimalizálás..... 19. oldal
   - Hazárdok kombinációs hálózatokban..... 22. oldal
   - Prímimimplikáns tábla használata ..... 27. oldal
   - Számjegyes minimalizálás..... 29. oldal
   - Hálózatépítés kész elemek felhasználásával..... 34. oldal

3. fejezet: Sorrendi hálózatok ..... 36. oldal
   - Sorrendi hálózatok bevezetése..... 37. oldal
   - Sorrendi hálózatok megismerése..... 38. oldal
   - Elemi sorrendi hálózatok (flip-flop)..... 43. oldal
   - Szinkron sorrendi hálózatok előzetes állapottáblájának felvétele..... 48. oldal
   - Aszinkron sorrendi hálózatok előzetes állapottáblájának felvétele... 53. oldal
   - Állapotösszevonások sorrendi hálózatokban..... 55. oldal
   - Aszinkron hálózat állapotainak kódválasztása ..... 62. oldal
   - Szinkron hálózat állapotainak kódválasztása ..... 66. oldal
   - Kódolt állapottáblák ..... 70. oldal
   - Aszinkron hálózatok függvényeinek felvétele ..... 71. oldal
   - Vezérlési tábla felvétele szinkron hálózatokhoz ..... 72. oldal
   - Sorrendi hálózatok megvalósítása ..... 75. oldal
   - Hazárdok sorrendi hálózatokban ..... 78. oldal
   - Sorrendi hálózatok analízise..... 79. oldal

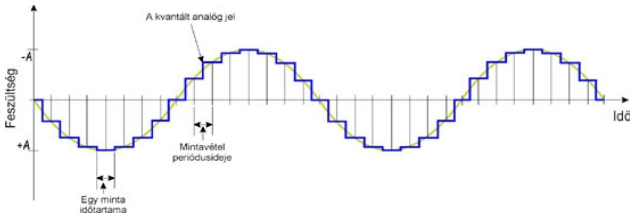

**A jegyzet szabadon terjeszthető és továbbfejleszthető, szerkeszthető formátumot a szerzőtől el lehet kérni, az esetleges hibákat a szerző felé bármikor lehet jelezni.**

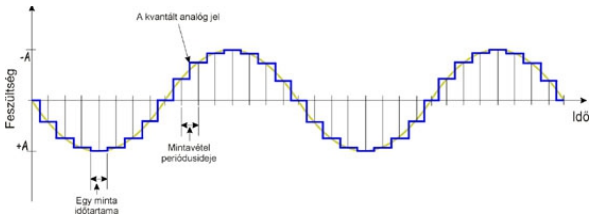
Ismerkedés a tárggyal

# A jelfeldolgozás alapjai:

Analóg technika esetén adott tartományon belül is végtelen sok állapot létezik, így mindenféle számítás pontatlan.
Ennek kiküszöbölésére a végtelen állapotot megpróbáljuk letranszformálni véges számú állapotra.
A digitális technika egy része ezzel a problémával foglalkozik.

Kvantálásnak nevezzük azt az eljárást, amikor a folytonos jelet véges számú szintet tartalmazó lépcsős görbével közelítjük. Ezeket a szinteket már pontosan meg tudjuk határozni.




A közhiedelemmel ellentétben a digitális jel pontosabb, mint az analóg jel. Analóg jellel számolni nehéz, digitális jellel azonban elérhetünk olyan finom felosztást, amely pontatlansága állandó, és amely pontatlanság az adott alkalmazásnál már elhanyagolható.

Analóg jel egyes mintáihoz egyazon kvantált érték is tartozhat.
A fentebb ismertetett módszer egy fontos paramétere a mintavételezés gyakorisága. Minél nagyobb gyakorisággal veszünk mintát, annál magasabb frekvenciát tudunk leírni és később visszaállítani.

Az analóg jelet feldolgozó áramkört ADC-nek, analóg-digitális konverternek nevezzük.

Szükség lehet azonban a rögzített analóg jel újbóli előállítására is, például ha zenét szeretnénk hallgatni egy digitális készülékről.
A digitális jelet ananlóg jellé alakító áramkört DAC-nak, digitális-analóg konverternek nevezzük.

## Számrendszerek:

A 2-es számrendszer ismerete elengedhetetlen a digitális technikában, az alábbiakban ezzel fogunk számolni.
Vizsgáljuk meg, hogyan válthatjuk át a 10-es számrendszerben megadott számot kettes számrendszerbe:

Egész:

Álltalános lépés: a számunkat osszuk 2-vel, és jegyezzük fel a maradékot.

| | |
|---|---|
| **13** | |
| **13:2=6** | **$mod_2(13)=1$** |
| **6:2=3** | **$mod_2(6)=0$** |
| **3** | **$mod_2(3)=1$** |
| **1** | **$mod_2(1)=1$** |

A számot alulról felfele olvashatjuk ki.

$mod_2(n)$: vesszük az n szám kettővel vett maradékát.

Tört:

Általános lépés: a számunkat szorozzuk kettővel, jegyezzük fel az első számjegyet, majd vegyük a törtrészt.

| | |
|---|---|
| **0.44** | **0, ...** |
| **0,44*2=0,88** | **0** |
| **0,88*2=1,76** | **1** |
| **0,76*2=1,52** | **1** |
| **0.52*2=1,04** | **1** |
| **...** | |

A számot felülről lefele olvashatjuk ki.

Az átváltás működik a már megimsert módszerrel is.
A helyiértékek sorra a kettő hatványai:

| $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

Ennek értelmében a fenti szám: $2^2 + 2^1 + 2^{-1} + 2^{-3} = 6,625$

Vizsgáljuk meg az átváltást más számrendszerek esetén:
8-as és 16-os számrendszer, valamint BCD kódolás esetén:
2-es és 8-as vagy 16-os számrendszerek között az átváltás meglehetősen egyszerű. 8-as számrendszer egy értéke 2-es számrendszerben 3 egymást követő értéknek felel meg, míg a 16-os számrendszer egy értéke 4 egymást követő számnak felel meg, melyet ki is használhatunk.

| Számrendszer | Érték | Érték | Számrendszer | Átváltott érték | Érték | Számrendszer | Átváltott érték |
|---|---|---|---|---|---|---|---|
| 2 | 00110101 | 000110101 | 8 | 065 | 00110101 | 16 | 35 |
| 8 | 127 | 127 | 2 | 001010111 | | | |
| 16 | A1 | A1 | 2 | 10100001 | | | |

A blokkokat mindig a szám végéről kezdjük el számolni. A szám elején hiányzó részt nullákkal tölthetjük fel, hiszen azok nem változtatnak semmit az eredeti számon. Pl.: 100 = 000100

**BCD számábrázolás** (binárisan kódolt decimális érték):

BCD számok átváltása hasonlóan működik ahhoz, amit a 16-os számrendszernél megfigyeltünk.
A BCD mindig tízes vagy kettes számrendszer alapú.
A tízes számrendszer beli szám kettes számrendszer beli megfelelőjét úgy kapjuk, hogy 1 számjegyet 4 bites bináris kóddal írunk le. Minden számjegynek egy négyes blokk felel meg.

Pl.: $1984_{10} = 0001\ 1001\ 1000\ 0010_{BCD}$

Figyeljünk a következőkre:
-Mindig 10-es az alap!
-Más számrendszerben megadott számot előbb 10-ez számrendszerbe váltunk, és csak ezután írhatjuk fel a BCD értéket!

Mivel véges helyiértéket használhatunk csak fel, ezért beszélnünk kell a **pontosságról**:
Könnyedén igazolható, hogy n bit törtrész esetén a pontosság $2^{-n}$.

MP: A 0,34 kettes számrendszerben egy elég hosszú tört: 0,01010111000...
Mi történik, ha ezt csak 4 biten ábrázolhatjuk:
0,0101 → 0,3125
0,0110 → 0,375
**$2^{-4}$ = 0,0625**
Azt tapasztaljuk, hogy a pontosság 0,0625, azaz ez a legkisebb lépcső, amivel lépkedhetünk a számaink közt.

**Törtek ábrázolása** a gyakorlatban kétféle módon fordulhat elő:

**Fix pontos törtképzés**: az egészrész és a törtrész leírására pontosan megszabott darabszámú bit áll rendelkezésre.

**Lebegőpontos számolás**: megszabott darabszámú bit közül a törtvessző bárhova kerülhet annak megfelelően, hogy kis számokat akarunk ábrázolni nagy pontossággal, vagy nagy számokat akarunk ábrázolni kis pontossággal.

**Negatív számok ábrázolása**:
A gyakorlatban szükséges, hogy negatív számokat is ábrázolni tudjunk, és ezek mellett azt is elvárjuk, hogy a kivonások helyes eredményt adjanak.

A megoldást a **kettes komplemens** használata jelenti.

A számegyenes második felét fordítsuk az első végéhez.

| 0-255 → | -128 … | 0 … | 127 |
|---|---|---|---|
| | 10000000 | 00000000 | 01111111 |

-A matematikai műveleteink működnek.
-Egy bittel többet kell használnunk.

Kettes komplemens esetén nem hozunk létre előjelbitet szánt szándékkal, azonban a képzés módja miatt az első bit mégis előjelbit lesz. Amennyiben az első bit 1-es, negatív számról beszélhetünk, ha azonban az első bit 0, a számunk pozitív.

A kettes komplemens használata:

Tízes számrendszerben adott szám átváltása kettes komplemensbe:

- Pozitív számot egyszerűen átválthatjuk kettes számrendszerbe. Figyeljünk arra, hogy legalább annyi bit szükséges, hogy az első
  bit 0 maradhasson.

- Negatív számot átváltjuk kettes számrendszerbe, mintha pozitív lenne.
  Az átváltás után, mivel negatív számot szerenténk kapni, ezért a számot bitenként invertáljuk, majd hozzáadunk 1 bitet. Fontos, hogy 1 bitet adjunk hozzá, tehát tört számok esetén is az utolsó bit-hez kell egyet adni.

  Pl.:
  1. invertálás: 01010101 → 10101010
  2. +1 bit: 10101010 → 10101011

n bit-en, 2-es komplemens esetében max és min érték:
max: $2^n/2 - 1$
min: $- 2^n/2$
|max|: $2^n/2$ Figyeljünk a kérdésre. A legnagyobb abszolútértékű a $-2^n/2$. Az előjelet ne feledjük!

Megkeressük a legkisebb negatív számot, ami: 1000 … 000
Invertáljuk, hozzáadunk 1-et, és megkaptuk a legnagyobb abszolút értékű számot.

Pl.: 6 bit egész, 2 bit tört, 2-es komplemens:

Számtartomány: -32 … -0,25 ; 0 ; 0,25 … 31,75

Legkisebb: 100000.00 = -32
Legnagyobb: 011111.11 = 31,75
Legnagyobb abszolútértékű: -32

Kettes komplemensben adott szám átváltása tízes számrendszerbe:

- Ha az első számjegy 0, akkor mint azt már megfigyeltük pozitív számról van szó.
  Ebben az esetben egyszerűen át kell váltanunk a kettes számrendszerben levő számunkat tízes számrendszerbe.
  Pl.: 011010.10=26.5

- Ha az első számjegyünk 1-es, akkor mint azt már megfigyeltük negatív számról van szó.
  Ebben az esetben invertáljuk bitenként a számunkat, adjunk hozzá 1 bitet, majd váltsuk át az így kapott számunkat, mint bármely kettes számrendszer beli számot. Ne felejtsük az előjelet!
  Pl.: 100101.10 → 011010.01 → 011010.10 =26.5 → -26.5

Nézzünk egy példát az összeadásra:

26.5 + (-26.5) = 0

```
  011010.10
+ 100101.10
 1000000.00
```

Jól látható, hogy nem 0-t kaptunk. Azonban az első bit már túlcsordul, így azt figyelmen kívül kell hagynunk, tehát mégis 0-t kaptunk.

# Boole algebra, logikai függvények:

Feladataink megoldása során fontos lesz, hogy a szöveggel megadott példákat valahogy le tudjuk írni.
Erre úgynevezett logikai függvényeket fogunk konstruálni.
A logikai függvényeknek igazságértékük van, azaz értékük vagy 0=hamis vagy 1=igaz.

Logikai kapcsolatok:

| | |
|---|---|
| ÉS kapcsolat: | A * B * … * X = 1, ha minden tényező 1 |
| VAGY kapcsolat: | A + B + … + X = 1, ha valamelyik tényező 1 |
| Negálás: | $\overline{A}$ = 1, ha A=0 és $\overline{A}$ = 0 ha A=1 (Negálást / jellel is jelölhetjük.) |
| NOR kapcsolat: | $\overline{A+B}$ = 1, ha minden tényező 0 |
| NAND kapcsolat: | $\overline{A*B}$ = 1, ha valamelyik tényező 0 |
| XOR kapcsolat: | A⊕B = 1, ha pontosan az egyik tényező 1 (Más néven *kizáró vagy.*) |
| XNOR kapcsolat: | A⊕B = 1, ha pontosan az egyik tényező 0 |

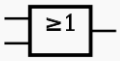
Azonosságok:

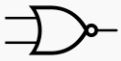
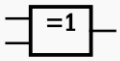
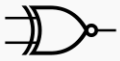
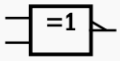
A*A ... *A = A
A+A ... +A = A
A*0=0
A+0=A
$\overline{0}$=1
$\overline{1}$=0

De Morgan tétel:

$\overline{(A+B)} = \overline{A} * \overline{B}$
$\overline{(A*B)} = \overline{A} + \overline{B}$

Logikai kapcsolatokat megvalósító kapuk és igazságtáblázatuk:

| Kapu | hagyományos jel | szögletes jel | művelet | Igazságtábla |
|---|---|---|---|---|
| AND (és) |  |  | $A \cdot B$ | bemenet A B / kimenet A AND B: 0 0 → 0; 0 1 → 0; 1 0 → 0; 1 1 → 1 |
| OR (megengedő vagy) |  |  | $A + B$ | bemenet A B / kimenet A OR B: 0 0 → 0; 0 1 → 1; 1 0 → 1; 1 1 → 1 |
| NOT (negálás) |  |  | $\overline{A}$ | bemenet A / kimenet NOT A: 0 → 1; 1 → 0 |

| Kapu | hagyományos jel | szögletes jel | művelet | Igazságtábla |
|---|---|---|---|---|
| NAND (negált *és*) |  |  | $\overline{A \cdot B}$ | bemenet A B / kimenet A NAND B: 0 0 → 1; 0 1 → 1; 1 0 → 1; 1 1 → 0 |
| NOR (negált *vagy*) |  |  | $\overline{A + B}$ | bemenet A B / kimenet A NOR B: 0 0 → 1; 0 1 → 0; 1 0 → 0; 1 1 → 0 |
| XOR, EXOR vagy MOD2 (kizáró vagy, antivalencia) |  |  | $A \oplus B$ | bemenet A B / kimenet A XOR B: 0 0 → 0; 0 1 → 1; 1 0 → 1; 1 1 → 0 |
| XNOR vagy EXNOR (negált kizáró vagy, ekvivalencia) |  |  | $\overline{A \oplus B}$ | bemenet A B / kimenet A XNOR B: 0 0 → 1; 0 1 → 0; 1 0 → 0; 1 1 → 1 |

## Kanonikus függvényalak:

Egy-egy feladatot végtelen számú helyes függvénnyel írhatunk le. Az egyszerűség kedvéért keresnünk kell egy olyan függvényalakot, ami tulajdonságainál fogva egyedülálló.

Z=(/a * /b * /c) + (a * /b * /c) + (a * /b * c) + (a * b * /c) + (a*b*c) … + (a*b*c)

Jól látható, hogy a fentebbi függvény végére bármennyiszer odaírhatnánk az (a*b*c) tagot, hiszen már egyszer szerepel a függvényben, vagy kapcsolatok állítják elő a függvényt, így ha annak értéke 1, akkor az ugyan olyan tagok nem változtatják a végeredményt.

**Kanonikus diszjunktív normálalak** (=DNA) (A felső szinten + (diszjunktív) kapcsolat van.)

1. ÉS kapcsolatok VAGY (diszjunkt) kapcsolata.
2. Minden ÉS kapcsolat tartalmazza az összes változót.
3. Mindegyik ÉS kapcsolat csak egyszer fordul elő.

A fentebbi kritériumrendszernek egyetlen egy függvény fog megfelelni.

Alkalmazzuk a DeMorgan azonosságot a Z függvényen (Z függvényt a lap tetején találjuk):

$Z = //Z = \overline{(/a * /b * c)} * \dots * \dots$

Z = (a + b + /c) * (a + /b + c) * (a + /b + /c)

Ez is kanonikus alak.

**Kanonikus konjunktív normál alak (**= KNA) (A felső szinten * (konjunktív) kapcsolat van.)

1. VAGY kapcsolatok ÉS (konjunkt) kapcsolata.
2. Minden VAGY kapcsolat tartalmazza az összes változót.
3. Mindegyik VAGY kapcsolat csak egyszer fordul elő.

A fentebbi példáknál 2 szintű függvényekkel foglalkoztunk, ami azt jelenti, hogy két szinten jelennek meg logikai kapcsolatok. De természetesen léteznek 1, 3 és több szintű függvények is.

Pl.: Z=(/a * /b * /c)⊕[(a * /b * /c) + (a * /b * c)]

## Termek:

A logikában azon szimbólumokat, melyeket konstansokból, változókból, vagy függvényekből állítunk elő, **termek**nek nevezzük. Amennyiben egy nyelv összes függvényszimbólumának a leírása elérhető, előállítható az adott nyelven értelmezett összes term, a konstansok és változók behelyettesítésével.

**Minterm**:

Azon logikai függvények szabályos alakjának független változóit hívjuk így, amelyek között ÉS kapcsolat áll fenn.

Jelölése: $m^n_i$
m: minterm
n: független változók száma
i: a minterm sorszáma (indexszáma)

A minterm sorszámát a bináris kód alapján a term változóiból képezzük. A változókat jobbról balra, növekvő sorrendű bináris helyértéknek tekintjük, majd az igaz változókat logikai 1-nek, a tagadott változókat logikai 0-nak tekintve a keletkezett bináris számot decimálissá alakítjuk.

Az igazságtábla egy-egy sorát és a normálalakok egy-egy összetevőjét egy m betűvel, alsó és felső indexel is jelölhetjük. A mintermeket diszjunkt alak esetén használjuk.

Z = (/A * /B * /C) + (A * /B * /C) + (A * /B * /C) + (A * B * /C) +(A*B*C)

Példa átalakításra:
$m^3_0 \rightarrow 000 \rightarrow$ (/A*/B* /C)

Alsó index a helyi érték szerint keletkezik.
A felső index a változószám.

Tehát a fentebbi Z függvényt a következő módon is leírhatjuk:
$Z = m^3_0 + m^3_4 + m^3_5 + m^3_7$

Ezen formát a következőképpen is szokás jelölni:
$Z = \sum^3 (0, 4, 5, 7)$

**Maxterm**:

Azon logikai függvények szabályos alakjának független változóit hívjuk így, amelyek között VAGY kapcsolat áll fenn. A mintermekkel ellentétben a maxtermeket konjunktív normálalak esetén használjuk.

Jelölése: $M^n_j$
M: maxterm
n: független változók száma
j: a maxterm sorszáma (indexszáma)

A maxtermek sorszámát a mintermekéhez hasonlóan számítjuk ki.

Z = (A + B + /C) * (A + /B + C) * (A + /B + /C)

$Z = M^3_6 * M^3_5 * M^3_4$

A mintermekhez hasonlóan ezt az alakot is írhatjuk másként:
$Z = \prod^3 (6, 5, 4)$

A felírások egyértelműek, hiszen egy-egy minterm vagy maxterm alsó indexét egy-egy decimális szám határozza meg, ami egyértelműen meghatároz egy bináris számsort.
A minterm és maxterm indexek használatához tudnunk kell, hogy mely változó milyen helyiértéket képvisel!

Maxtermek és mintermek negáltjait oda-vissza módosíthatjuk:

$/m^n_i = M^n_{(2^n - 1 - i)}$

$/M^n_i = m^n_{(2^n - 1 - i)}$

A minterm indexekből a maxterm indexeket egyszerűen számolhatjuk. A függvényben nem szereplő összes minterm indexét a maximális indexből (2 változó esetén 3, 3 változó esetén 7, 4 változó esetén 15 ...) kivonva megkapjuk a maxterm indexeket.
A módszer táblázatos formában is remekül látható. Ha betartjuk a táblázat képzési szabályát, melyet később részletezünk, akkor a minterm indexek felülről lefelé nőnek, míg a maxtermek felülről lefelé csökkennek.
Figyeüljünk arra, hogy az indexelés 0-val indul!
A függvényben <u>nem</u> szereplő minterm indexekkel szomszédos maxterm indexeket kell kiválasztanunk.

| A (4) | B (2) | C (1) | Z | Minterm | Maxterm |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | $m^3_0$ | $M^3_7$ |
| 0 | 0 | 1 | 0 | $m^3_1$ | $M^3_6$ |
| 0 | 1 | 0 | 0 | $m^3_2$ | $M^3_5$ |
| 0 | 1 | 1 | 0 | $m^3_3$ | $M^3_4$ |
| 1 | 0 | 0 | 1 | $m^3_4$ | $M^3_3$ |
| 1 | 0 | 1 | 1 | $m^3_5$ | $M^3_2$ |
| 1 | 1 | 0 | 1 | $m^3_6$ | $M^3_1$ |
| 1 | 1 | 1 | 1 | $m^3_7$ | $M^3_0$ |

A színekkel szemléltetett példa:

$Z = \sum^3 (0, 3, 4, 5)$

$Z = \prod^3 (6, 5, 1, 0)$

<u>Bitek helyiértéke:</u>
Mint az fentebb már meg lett említve, bizonyos esetben fontos, hogy tudjuk, melyik bit milyen helyiértéket képvisel.
A legnagyobb helyiértékű bitet MSB-nek (Most significant bit) nevezzük.
A bemeneteket az ABC betűivel jelöljük általában, és a helyiértékek is rendszerint ABC sorrendben követik egymást, azonban figyeljünk arra, hogy egyes forrásokban az A változó a legkisebb helyiérték, míg más forrásokban az A a legnagyobb helyiérték.
Mi utóbbi jelölést fogjuk választani, de más jelölések nem okoznak jelentős eltéréseket.

Pl.: 3 bemenet esetén:

A változó az MSB: $A=2^2$, $B=2^1$, $C=2^0$

C változó az MSB: $C=2^2$, $B=2^1$, $A=2^0$

<u>A fentebb látható igazságtáblának képzési módja:</u>
Helyiérték sorrendben, a legnagyobbal kezdve írjuk fel a bemeneteket. A táblázatot helyiértékenként fentről lefele haladva töltjük ki. Mindig nulákkal kezdünk.
A 0-k és 1-ek a helyiértéknek megfelelően váltogatják egymást.

Pl.: a $2^0$=1 helyiérték esetén egyesével váltogatják egymást az értékek, azaz: 01010101

Pl.: a $2^3$=8 helyiérték esetén nyolcasával váltogatják egymást az értékek, azaz: 0000000011111111

| A (4) | B (2) | C (1) | Z | Minterm | Maxterm |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | $m^3_0$ | $M^3_7$ |
| 0 | 0 | 1 | 0 | $m^3_1$ | $M^3_6$ |
| 0 | 1 | 0 | 0 | $m^3_2$ | $M^3_5$ |
| 0 | 1 | 1 | 0 | $m^3_3$ | $M^3_4$ |
| ... | | | | | |

## A kombinációs és sorrendi hálózatok bevezetése:

A továbbiakban kombinációs és sorrendi hálózatokkal fogunk foglalkozni. Ezeket fogjuk megtervezni. Ezeket a későbbiekben részletesen is ismertetjük, csak bevezető jelleggel írunk róluk, hogy a továbbiakban beszélhessünk ezekről.

Egy feladat megvalósításának első lépése, hogy kiválasztjuk, hogy milyen hálózatra lesz szükségünk.

**Kombinációs hálózat:**
Számos feladatnál olyan áramkörre van szükség, amely számos bemenetet és kimenetet tartalmaz, valamint a pillanatnyi bemenetek egyértelműen meghatározzák az aktuális kimenetet. Ezeket az áramköröket kombinációs hálózatoknak nevezzük.
Ilyen hálózatok például a multiplexerek vagy komparátorok.

Feladat megvalósításának menete kombinációs hálózat esetén:
- -Függvény elkészítése.
- -Függvény minimalizálása.
- -Feladat megvalósítása kapuk és/vagy előre legyártott építőelemek segítségével.
  - -Elvi logikai rajz
  - -Kapcsolási rajz

**Sorrendi hálózat:**
Számos feladatot nem tudunk úgy megoldani, hogy nem emlékszünk az előző állapotra. Például egy lift vezérlésnél szükséges, hogy tudjuk, melyik emeleten áll a lift. A bemenet az adott emeleten megnyomott gomb, viszont ez a lift szempontjából rengeteg dolgot jelenthet (például 4 emelet felfele vagy 2 emelet lefele) attól függően, hogy a lift jelenleg hol tartózkodik. Ebben az esetben szükséges, hogy tudjuk az előző állapotot.
Sorrendi hálózatoknál a kimeneti kombinációt a bemenetek aktuális értékei, valamint a korábban fennállt értékei is befolyásolják.

Feladat megvalósításának menete sorrendi hálózat esetén:
- -Előzetes állapottábla felvétele
- -Állapotok összevonása
- -Állapotok kódolása
- -Kódolt állapottábla felírása
- -Vezérlési tábla elkészítése
- -Kimeneti és szekunder változók függvényeinek elkészítése
- -Feladat megvalósítása kapuk és flip-flopok segítségével
  - -Elvi logikai rajt
  - -Kapcsolási rajz

A továbbiakban a kétféle hálózatot, és azok megvalósításához szükséges lépéseket is részletesen tárgyaljuk.

Kombinációs hálózatok

Függvény felírása szöveges feladat alapján:

MP: Tekintsünk egy 3 bemenetű (A, B, C) és 1 kimenetű (Z) kombinációs hálózatot, melynek működése a következő:

Ha AB-n előálló bináris szám decimális értéke nagyobb vagy egyenlő, mint BC-n előálló bináris szám decimális értéke, akkor Z=1, egyébként Z=0.

A bemeneti variációk száma $2^3$
A kimeneti variációk száma 2, hiszen Z=1 vagy Z=0.

Konstruáljuk meg az igazságtáblát:

| A ($2^2$) | B ($2^1$) | C ($2^0$) | F | Minterm | /F | Maxterm |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | $m^3_0$ | 0 | $M^3_7$ |
| 0 | 0 | 1 | 0 | $m^3_1$ | 1 | $M^3_6$ |
| 0 | 1 | 0 | 0 | $m^3_2$ | 1 | $M^3_5$ |
| 0 | 1 | 1 | 0 | $m^3_3$ | 1 | $M^3_4$ |
| 1 | 0 | 0 | 1 | $m^3_4$ | 0 | $M^3_3$ |
| 1 | 0 | 1 | 1 | $m^3_5$ | 0 | $M^3_2$ |
| 1 | 1 | 0 | 1 | $m^3_6$ | 0 | $M^3_1$ |
| 1 | 1 | 1 | 1 | $m^3_7$ | 0 | $M^3_0$ |

Írjuk fel a logikai függvényeket, az igazságtábla alapján:

DNA:

Z = (/A * /B * /C) + (A * /B * /C) + (A * /B * C) + (A * B * /C)

KNA:

Z = (/A+/B+C) * (/A + B + /C) * (/A + B + C)

MP: A, B, C kapcsolókkal szeretnénk vezérelni egy berendezést. A működés a következő módon történjen:
A berendezés akkor működik, ha az A és B kapcsolót egyszerre nyomjuk le, vagy ha lenyomjuk az A kapcsolót, de a C kapcsolót nem. Az A változó az MSB.

Értelmezzük a feladatot és készítsük el a működésnek megfelelő logikai függvényt:
F = A és B vagy A és /C
F=A*B + A*/C

Ez azonban nem kanonikus normálalak.
Bővítsük ki:

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

DNA:

A fentebb bővített függvény már kanonikus. Írjuk fel a mintermes alakját:
$F= m^3_4 + m^3_6 + m^3_7 = \Sigma^3 (4, 6, 7)$

KNA:

-A konjunktív normálalak megalkotásához az F függvény negáltjának a negáltját vesszük, és alkalmazzuk a deMorgan azonosságokat.
KNA felírásához azokat a bemeneteket kell vennünk, ahol az eredeti kimenet 0, és minden tagot az eredeti értékének negáltjaként kell tekintenünk.
Tehát: 0 0 0 → A B C

$F= \Sigma^3 (4,6,7)$
$/F= \Sigma^3 (0,1,2,3,5)$ → Itt fontos a változóink száma, 0-val kezdődnek a minterm indexek!

$F= \Pi^3 (7\text{-}0, 7\text{-}1, 7\text{-}2, 7\text{-}3, 7\text{-}5) = \Pi^3 (7, 6, 5, 4, 2)$

A minterm indexeket kivonjuk a legnagyobb indexből, hiszen a maxtern indexeket így képezzük.

-Gondolkodás nélkül is átalakíthatjuk a mintermeket maxtermekké, ahogy fenn ismertettük:
Leírjuk az igazságtábla fordítottját, és abból írjuk fel az F függvényt.

$F= M^3_7 * M^3_6 * M^3_5 * M^3_4 * M^3_2 = \Pi^3 (2,4,5,6,7)$

DE: $M^3_7$ esetén a 7-es index A+B+C alapján számolódik, de a maxterm jelentése /A + /B + /C.

| A | B | C | F | Minterm | /F | Maxterm |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $m^3_0$ | 1 | $M^3_7$ |
| 0 | 0 | 1 | 0 | $m^3_1$ | 1 | $M^3_6$ |
| 0 | 1 | 0 | 0 | $m^3_2$ | 1 | $M^3_5$ |
| 0 | 1 | 1 | 0 | $m^3_3$ | 1 | $M^3_4$ |
| 1 | 0 | 0 | 1 | $m^3_4$ | 0 | $M^3_3$ |
| 1 | 0 | 1 | 0 | $m^3_5$ | 1 | $M^3_2$ |
| 1 | 1 | 0 | 1 | $m^3_6$ | 0 | $M^3_1$ |
| 1 | 1 | 1 | 1 | $m^3_7$ | 0 | $M^3_0$ |

MP: Legyen 4 bemenetünk (A, B, C és D) és 3 kimenetünk ($F_1$, $F_2$, $F_3$).
Az $F_1$ kimenet értéke legyen 1-es, ha az AB-n előálló bináris szám egyenlő a CD-n előálló bináris számmal.
Az $F_2$ kimenet értéke legyen 1-es, ha az AB-n előálló bináris szám kisebb, mint a CS-n előálló bináris szám.
Az $F_3$ kimenet értéke legyen 1-es, ha az AB-n előálló bináris szám nagyobb, mint a CS-n előálló bináris szám.
Az A változó az MSB.

Bemenetek: $x_1$=A, $x_0$=B, $y_1$=C, $y_0$=D
Kimenetek: $F_1$, $F_2$, $F_3$

$F_1 \rightarrow x=y$
$F_2 \rightarrow x<y$
$F_3 \rightarrow x>y$

| $A=x_1$ | $B=x_0$ | $C=y_1$ | $D=y_0$ | $F_1$ (x=y) | $F_2$ (x<y) | $F_3$ (x>y) | minterm | maxterm |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 15 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 14 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 13 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 3 | 12 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 4 | 11 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 5 | 10 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 6 | 9 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 7 | 8 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 8 | 7 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 9 | 6 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 10 | 5 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 11 | 4 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 12 | 3 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 13 | 2 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 14 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 15 | 0 |

Ekkor a kimeneti függvények:

$F_1 = \Sigma^4(0, 5, 10, 15) \rightarrow /F_1 = \Sigma^4(1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14) \rightarrow F_1 = \Pi^4(14, 13, 12, 11, 9, 8, 7, 6, 4, 3, 2, 1)$

Példa kedvéért vizsgáljuk meg, mit jelent 1-1 minterm vagy maxterm:

$M^4_1$ : A + B + C + /D  $m^4_0$: (/A*/B*/C*/D)
$M^4_6$ : /A + B + C + /D  $m^4_5$: (A*/B*/C*D)
$M^4_9$ : A + /B + /C + D  $m^4_{10}$: (A*/B*C*/D)

$F_2 = \Sigma^4(1, 2, 3, 6, 7, 11) \rightarrow /F_2 = \Sigma^4(0, 4, 5, 8, 9, 10, 12, 13, 14, 15) \rightarrow F_2 = \Pi^4(15, 11, 10, 7, 6, 5, 3, 2, 1, 0)$

$F_3 = \Sigma^4(4, 8, 9, 12, 13, 14) \rightarrow /F_3 = \Sigma^4(0, 1, 2, 3, 5, 6, 7, 10, 11, 15) \rightarrow F_3 = \Pi^4(0, 4, 5, 8, 9, 10, 12, 13, 14, 15)$

MP: Teljes összeadó megvalósítása. 3 bemenete van. Kettőn bitenként fogad 1-1 számot, a harmadikon pedig az előző elemből jövő túlcsordulást jelző bit érkezik. Egy darab S kimenete van. A működése a következő: Összeadja a 3 bemenet értékét, és kiadja S-en, ha van túlcsordulás, akkor azt $C_{out}$-on jelzi.
Ilyeneket összefűzve bitenként adhatunk össze számot. Ekkor az első elem carry bemenetét 0-ra kell állítanunk. Az A változó az MSB.

| A | B | $C_{in}$ = carry | S | $C_{out}$ | minterm | Maxterm |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 7 |
| 0 | 0 | 1 | 1 | 0 | 1 | 6 |
| 0 | 1 | 0 | 1 | 0 | 2 | 5 |
| 0 | 1 | 1 | 0 | 1 | 3 | 4 |
| 1 | 0 | 0 | 1 | 0 | 4 | 3 |
| 1 | 0 | 1 | 0 | 1 | 5 | 2 |
| 1 | 1 | 0 | 0 | 1 | 6 | 1 |
| 1 | 1 | 1 | 1 | 1 | 7 | 0 |

KNA:
$S = \Pi^3 ( 7, 4, 2, 1)$ $C_{out} = \Pi^3 ( 7, 6, 5, 3)$

DNA:
$S = \Sigma^3 (1, 2, 4, 7)$ $C_{out} = \Sigma^3 ( 3, 5, 6, 7)$

Egyszerűsítések után $C_{out}$-ra a következőt kaphatjuk: $C_{out} = AB + AC + BC$ Ez a következő példa miatt kell.

NAND és NOR kapuk használata:

Itt nem részletezett okok miatt sok esetben előnyös, ha NAND vagy NOR kapukat használunk.

Nézzük meg mi történik, ha **NAND kaput** szeretnénk használni:

$C_{out} = AB + AC + BC$

Ha kétszer negáljuk a C-t: $//C = \overline{( /(A*B) * /(A*C) * /(B*C) )}$

Az így kapott függvény megvalósítható csak NAND kapuk felhasználásával!

Nézzük mi történik, ha **NOR kapukat** szeretnénk használni:

$S = \Pi^3 (7,4, 2,1) = (/A+/B+C)*(A+/B+/C)*(/A+B+/C)*(/A+/B+C)$

Ezt is invertáljuk kétszer: $S = \overline{( /(/A+/B+C) + /(A+/B+/C) + \dots )}$

Az így kapott függvény megvalósítható csak NOR kapuk felhasználásával!

Foglaljuk össze:
Az ilyen két szintű diszjunkt függvények megvalósíthatók kizárólag NAND kapukkal.
**Mindkét szinten NAND kapu = első szinten ÉS kapcsolat második szinten VAGY kapcsolata!**
Az ilyen két szintű konjunkt függvények megvalósíthatók kizárólag NOR kapukkal.
**Mindkét szinten NOR kapu = első szinten VAGY kapcsolat, második szinten ÉS kapcsolat!**

Fontos: A legtöbb ilyen függvénynél semmit nem kell változtatni, csak a kapukat cserélni, de azon változók esetén, amik közvetlenül a 2. szintre csatlakoznak, nem jelölünk kaput. Az ilyen átírásnál viszont szükséges egy egy bemenetű NAND kaput beiktatnunk az útjukba, ami működésben egy inverternek felel meg.

## Függvényalakok egyszerűsítése:

Fentebb láthattuk, hogy hogyan készíthetjük el a logikai függvények kanonikus alakját. Ezek azonban a bemenetek számát illetően elég pocsékolók.
A továbbiakban az lesz a célunk, hogy minél egyszerűbb alakot találjunk a függvényeinknek a következő szempontok szerint:

-Legkevesebb változó.
-2 szintűség megőrzése.
-Legkevesebb kapubemenet.

Az egyszerűsítésre az ad lehetőséget, hogy léteznek **szomszédos minterm/maxtermek**.
Egy-egy mintermet/maxtermet szomszédosnak nevezünk, ha egyetlen egy tagban különböznek.
Egyértelmű, hogy A+/A=1 és A*/A=0. Ezt felhasználva egyszerűsíthetünk.

Nézzük például az F= (A*B*C)+(A*B*/C) függvényt.
Ha jól látható, hogy a két minterm 1 változóban, C-ben különbözik, hiszen az az egyik helyen ponált, a másik helyen negált értékként szerepel. Ha az A és B értéke 1, akkor a két minterm közül pontosan az egyik 1 és a másik pedig ekkor 0. Mivel közöttük vagy kapcsolat van, ekkor az F biztosan 1.
Ha az A és B változó közül azonban az egyik 0, akkor egészen biztosan 0 a függvény értéke.
Látható tehát, hogy a függvény kimenete ebben az esetben független a C változótól, az A és B változó egyértelműen meghatározza az értékét.
A függvényünket tehát a következő módon is felírhatjuk:
F=A*B

Nézzük például az F= (A+B+C)*(A+B+/C) függvényt.
Ha jól látható, hogy a két maxterm 1 változóban, C-ben különbözik, hiszen az az egyik helyen ponált, a másik helyen negált értékként szerepel. Ha az A és B értéke 0, akkor a két minterm közül pontosan az egyik 1 és a másik pedig ekkor 0. Mivel közöttük és kapcsolat van, ekkor az F biztosan 0.
Ha az A és B változó közül azonban az egyik 1, akkor egészen biztosan 1 a függvény értéke.
Látható tehát, hogy a függvény kimenete ebben az esetben független a C változótól, az A és B változó egyértelműen meghatározza az értékét.
A függvényünket tehát a következő módon is felírhatjuk:
F=A*B

Összegezve: Ha találunk két szomszédos szorzatot/összeget, akkor őket egy darabbal helyettesíthetjük, mely nem tartalmazza az eltérő változót. Ezeket a szorzatokat/összegeket **implikánsok**nak nevezzük.

A fent ismertetett eljárást nem csak egyszer alkalmazhatjuk, hanem egymás után többször is, amíg szomszédos szorzatokat/összegeket találunk. A folyamatnak ott van vége, ahol: nem marad, csak 1 változó, vagy nincs szomszédos szorzat/összeg. A folyamat végén kapott szorzatokat/összegeket **prímimplikánsok**nak nevezzük.

A prímimplikánsok két csoportra oszthatóak:

-**Lényeges prímimimplikánsok:** ha létezik olyan term, amit egyedül a vizsgált prímimimplikáns fed le, akkor ez a prímimplikáns lényeges prímimplikáns, és nekünk mindeképpen meg kell valósítanunk.
-**Nem lényeges prímimplikáns:** nincs olyan term, amit egyedül a vizsgált prímimplikáns fed le, akkor ez a prímimplikáns nem lényeges prímimplikáns. Ebből azonban még nem derül ki, hogy meg kell-e valósítanunk, hiszen előfordulhat olyan eset, hogy egy termet csak nem lényeges prímimimplikánsok fednek le. Ha egyiket se valósítanánk meg, hibát követnénk el.

Az implikánsokhoz hasonlóan a termeket is csoportokra oszthatjuk:

**Megkülönböztetett termeknek** azok a termek, amiket csak egy prímimplikáns fed le.
Tehát elmondható, hogy minden olyan prímimplikáns, ami lefed megkülönböztetett termet, lényeges prímimplikáns, és amennyiben egy implikáns prímimimplikáns, lefed legalább egy megkülönböztetett termet.

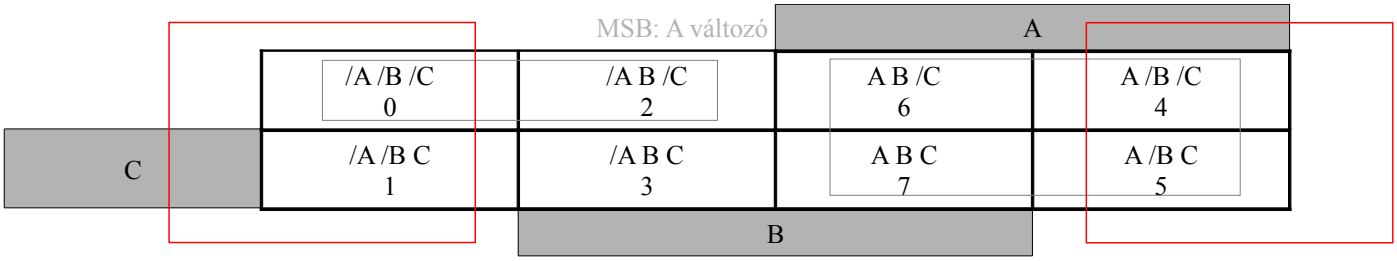
**„Dont care" értékek:**

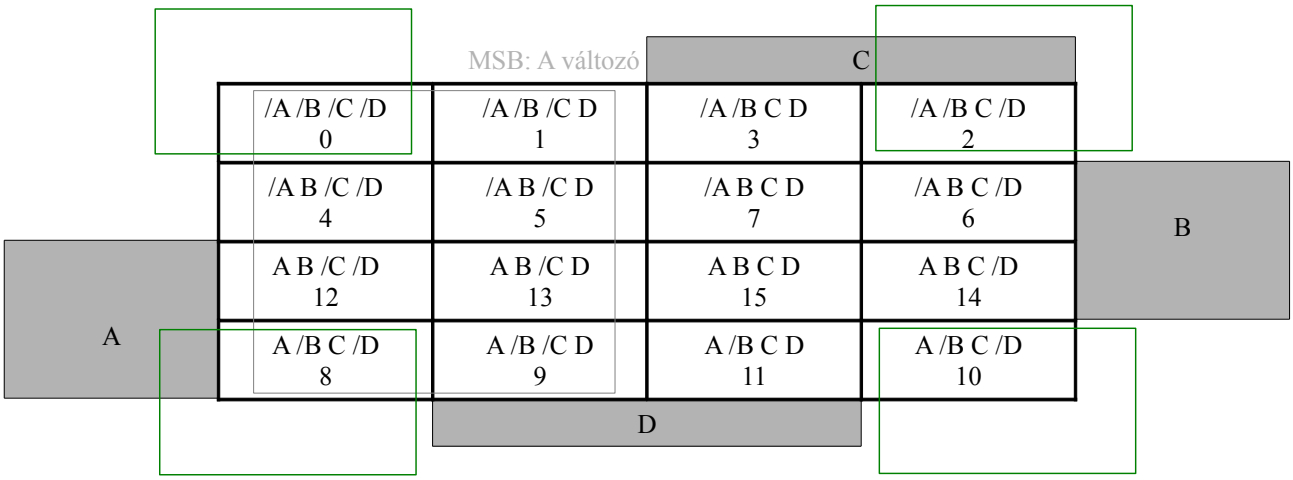
Vannak olyan bemeneti kombinációk, amik valamiért nem fordulhatnak elő, vagy ha előfordulnak, akkor nem érdekel minket a hozzájuk tartozó kimenet. Ezekben az esetekben a kimeneteket tetszés szerint rögzíthetjük.
A dont care értékeket – vagy x jelöli általában.

## Grafikus minimalizálás:

A fentebb tárgyalt prímimplikánsok megtalálására létezik kétféle algoritmus is, melyek megkönnyítik a dolgunkat. Az egyik a **számjegyes minimalizálás**, melyet később tárgyalunk, a másik a **grafikus minimalizálás**, melyet most részletezünk.

A grafikus minimalizáláshoz egy olyan táblázatot kell készítenünk, amelyben a szomszédos termek egymás mellett foglalnak helyet. Az elvárásainknak megfelelő táblázatot **Karnaugh-táblának** nevezzük.
A tábla minden egyes cellája 1-1 termet jelenít meg. Helyes peremezés esetén a szomszédos termek a táblában is egymás mellé kerülnek, így könnyedén észrevehetjük az összevonásokat.
A peremezés határozza meg az egyes maxtermek és mintermek helyét, és kiderül belőle a változók száma is. Figyeljünk a változók helyiértékére is. Minden változóhoz a tábla fele tartozik. Azok a sorok/oszlopok, amelyek mellett/felet a változó jelölve van azt jelentik, hogy ott az adott változó ponálva szerepel.
Amennyiben a peremezést egyféle módon készítjük, akkor a minterm és maxterm indexek minden esetben ugyan abba a cellába kerülnek, így az indexekkel adott függvényeket könnyedén átírhatjuk.




A minimalizálás menete:
A táblában 1-eseket vagy 0-kat vonunk be 2-es, 4-es, 8-as vagy 16-os hurokba.
Mindig **2 hatványának megfelelő termet kell egy hurokba vennünk**, tehát 9-es hurkokat NEM készíthetünk például.
A hurkokra a táblákban találunk példákat.
A szürke hurkok egyértelműek. **A leggyakoribb hiba a módszernél, hogy a fal menti szomszédságot nem figyelik a diákok. A zöld vagy a piros hurkok egy-egy hurkot jelölnek!**
A 4 sarokban levő termek is szomszédosak például, de a falak peremein át képezhetünk 2-es vagy 8-as hurkokat is!

A hurkok függvényei könnyedén leolvashatóak:
Mintermek esetén (amennyiben 1-eseket fedjük le hurkokkal) az első táblán például a piros hurok jelentése /B, a második táblán a zöld hurok jelentése /B*/D.
Maxtermek esetén (amennyiben a 0-kat fedjük le hurkokkal) a hurkokat ugyan úgy olvassuk le, majd változónként tagadunk. Első táblán tehát a piros hurok jelentése B, a második táblán a zöld hurok jelentése B+D.

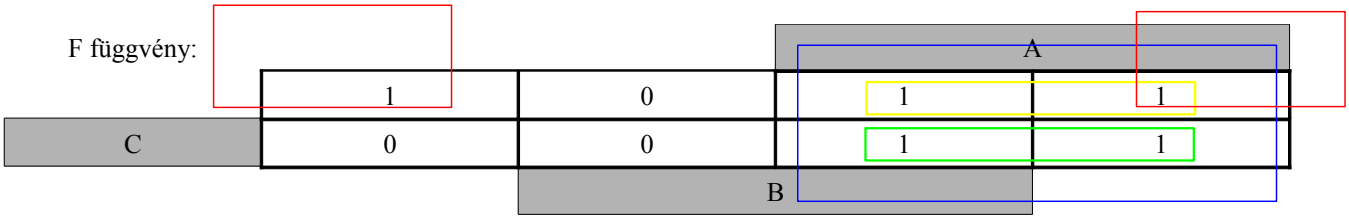
Grafikus minimalizálás 3 és 4 változóra könnyedén működik. 5 változó esetén két darab 4x4-es táblával kell dolgoznunk. A táblák csak 4 változót tartalmaznak, az 5 változó a két táblánál jelenik meg. Az egyik táblánál az 5. változót ponáltnak vesszük, a másik változót negáltnak vesszük. 5 változó felett a számjegyes minimalizálást használhatjuk majd.

MP: Kanyarodjunk vissza egy korábban említett példánkra, melyet igazságtáblázatával adunk meg:

| A (4) | B (2) | C (1) | F | Minterm | Maxterm |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | $m^3_0$ | $M^3_7$ |
| 0 | 0 | 1 | 0 | $m^3_1$ | $M^3_6$ |
| 0 | 1 | 0 | 0 | $m^3_2$ | $M^3_5$ |
| 0 | 1 | 1 | 0 | $m^3_3$ | $M^3_4$ |
| 1 | 0 | 0 | 1 | $m^3_4$ | $M^3_3$ |
| 1 | 0 | 1 | 1 | $m^3_5$ | $M^3_2$ |
| 1 | 1 | 0 | 1 | $m^3_6$ | $M^3_1$ |
| 1 | 1 | 1 | 1 | $m^3_7$ | $M^3_0$ |

DNA: $F = m^3_0 + m^3_4 + m^3_5 + m^3_6 + m^3_7$

Mintermek esetén:

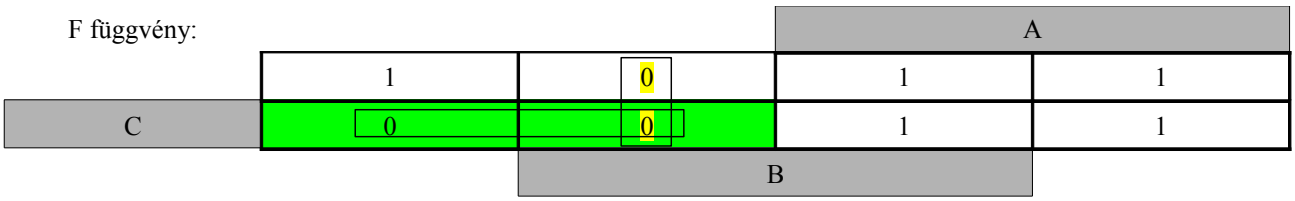



Implikánsok tehát:
: A*/C
: A*C
: A
: /B*/C

Jól látható, hogy a sárga és a zöld nem prímimplikáns. A kék és a piros viszont prímimplikáns.
Mindkettő lényeges prímimplikáns, hiszen a 0-s mintermet csak a piros fedi le, a kékhez pedig 3 darab olyan minterm is van, amelyet egyedül fed le (a sima implikánsokat nem tekintjük).

Tehát az F függvény egyszerűsítésére azt kaptuk, hogy:
F= A + /B/C

Maxtermek esetén:

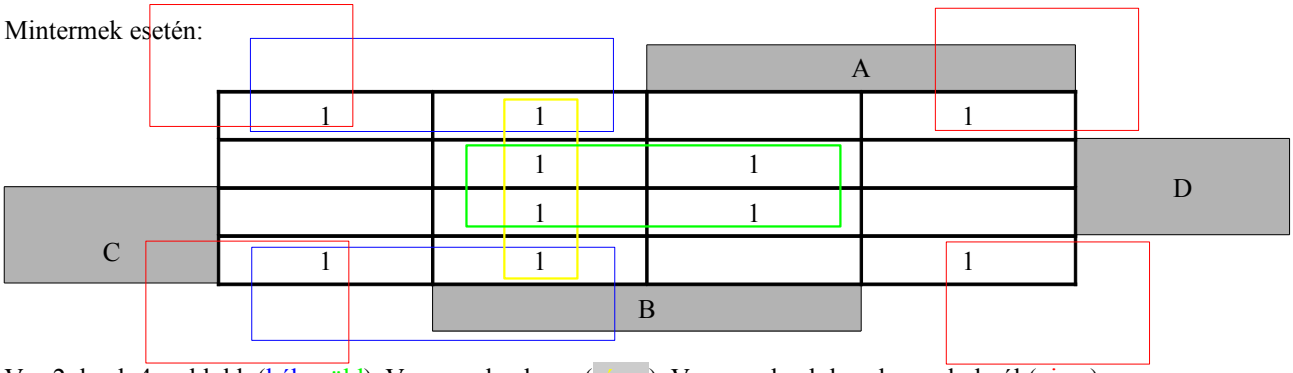



Implikánsok tehát:
: A+/B
: A+/C

Maxtermek esetén tehát az F függvényre az alábbi egyszerűsítés adódik:
F= (A+/B)*(A+/C)

MP: Karnaugh táblájával adott az alábbi példa:

Mintermek esetén:




Van 2 darab 4-es blokk (kék, zöld). Van egy darab sor (sárga). Van egy darab hurok a sarkoknál (piros).
Jól látható, hogy vannak olyan megkülönböztetett mintermek, amiket csak egy prímimimplikáns érint. Ezek lényeges prímimimplikánsok (piros, zöld). A sárgából és a kékből viszont elég egyet választanunk, hiszen átfedésben vannak.

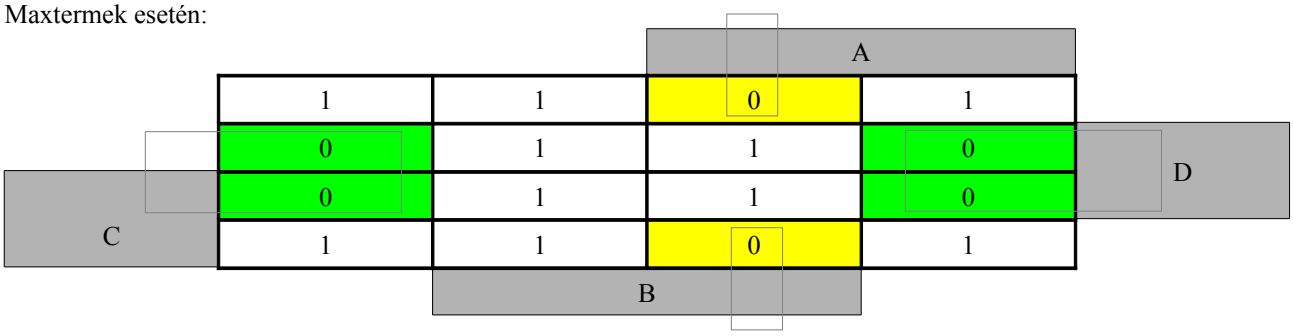
: /AB
: BD
: /B/D
: /A/D

A következő egyszerűsítás adódott:
Z= BD + /B/D + /A/D
+ /AB
Ezek közül egyet kell választanunk, nincs szükség mindkettőre!

Az így kapott alakot **diszjunktív minimálalaknak** (DMA) nevezzük.

Az általunk megadott függvény megvalósításához 9 darab kapubemenet szükséges.
Ennél kevesebb kapubemenettel is megoldhatnánk a feladatot a következő módon:
Z= B(D+/A) + /B/D
Így csak 8 kapubemenetre volna szükség, de elvesztenénk a kétszintűséget, ami számunkra nem jó!

Maxtermek esetén:




: B+/D
: /A+/B+D

Minden maxterm kitüntetett maxterm és mindkét prímimimplikáns lényeges prímimimplikáns.
A következő egyszerűsítés adódott tehát:
Z= (B+/D)*(/A+/B+D)

Az így kapott alakot **konjunktív minimálalaknak** (KMA) nevezzük.

Az általunk megadott függvény megvalósításához 7 darab kapubemenet szükséges.

## Hazárdok kombinációs hálózatokban:

Bár konkrétan még nem jelentettük ki, de valószínűleg már mindenki számára világos, hogy elektromos alkatrészekkel szeretnénk megvalósítani a hálózatainkat. A logikai értékeket feszültségszinteknek feleltetjük meg, így a 0 és 1 állapot közti átmenet nem történhet "idő nélkül", hiszen ekkor végtelen teljesítményről beszélnénk. Így minden elemnek van egy felfutási ideje, azaz egy $\Delta t$ késleltetése.

Ez a késleltetés több tényezőtől függ, egyrészt a kapuk felépítéséből, másrészt a kábelek használatából következi, mivel a kábelek modellje szerint soros RL kör és párhuzamos kondenzátorként modellezhetjük kábeleinket. A négyszögjel tehát a kábelen is torzul, és ez a torzulás olyan, hogy azt mi késleltetésnek érzékeljük.

A következő megállapítást tehetjük:

$$\Delta t = \Delta t_p + \Delta t_s$$

$\Delta t_p$ : propagation delay, azaz a kapukból származó késleltetés.
- -Pontosan megadható, a gyártási technológiáktól és a kivitelezéstől függ.

$\Delta t_s$ : strayed delay, azaz a szórt impedanciákból következő késleltetés.
- -Nem adható meg előre, függ a környezettől, időben változhat.

$\Delta t_p$ egy ideig elnyomta a $\Delta t_s$ -t, azonban a technológia fejlődésével az építőelemek gyorsultak, és a kapukésleltetés beesett a szórt impedanciából következő késleltetés nagyságrendjébe, így ma már nem tudjuk túlságosan pontosan megadni.

A továbbiakban a kombinációs hálózatokra jellemző 4 féle hazárdot fogunk megismerni:

- -Statikus 1 hazárd
  - -minimum két szintű kombinációs hálózat
  - -diszjunkt hálózat
  - -szomszédos bemeneti változás
- -Statikus 0 hazárd
  - -minimum két szintű kombinációs hálózat
  - -konjunkt hálózat
  - -szomszédos bemeneti változás
- -Dinamikus hazárd
  - -minimum 3 szintű kombinációs hálózat
- -Funkcionális hazárd
  - -minimum két szintű kombinációs hálózat
  - -nem szomszédos bemeneti változás

Nagyon fontos megjegyezni, hogy a hazárdok a hibáknál nagyobb gondot okoznak számunkra.
A hibák folyamatosan fennállnak, így megtalálhatóak, ezzel szemben hazárdok esetén előfordulhat, hogy a vizsgálat ideje alatt nemtapasztalunk rendellenességet!

**Statikus 1 hazárd (diszjunkt hálózat esetén):**

Statikus 1 hazárdról beszélünk, ha szomszédos bemeneti kombináció változás esetén az F függvény kimenete 1 értéket kéne, hogy tartson, azonban valamelyik késleltetésnek köszönhetően a hálózat ideiglenesen olyan állapotba kerül, ahol a kimenet 0. Ebben az esetben az F függvény kimenetén 1-0-1 átmenetet tapasztalhatunk.

Tekintsünk egy szomszédos állapotváltozást, ahol mindkettő bemeneti kombinációhoz 1 tartozik. Mivel mind a kettő esetén 1 a kimenet, nem szabadna változást észlelnünk.

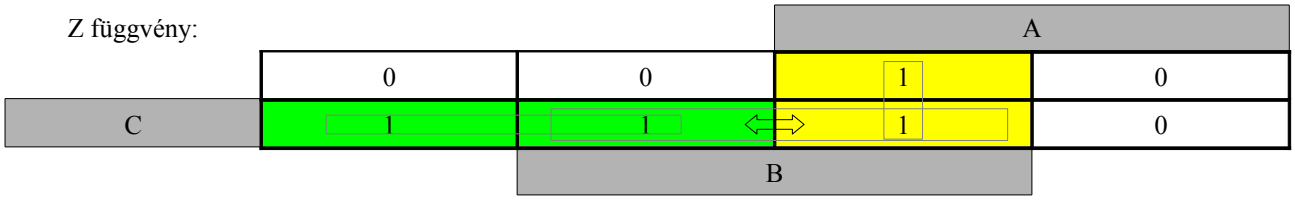
Ideális esetben ténylegesen 1-1 átmenetünk van, és nem tapasztalunk semmi rendelleneset.
Azonban az AND kapuk értékei megváltoznak, ezáltal a OR kapun szerepcsere történik. Eddig az egyik AND kapu kimenete biztosította, hogy az OR kapu kimenete 1 legyen, azonban ha ezen kapu kimenete hamarabb megy 0-ra, mint a másik kapu kimenete 1-re, akkor az OR kapu bemenetein rövid ideig csak 0-k jelennek meg, így a kimenete is 0.

A statikus 1 hazárd **kiküszöböléséhez** módosítanunk kell a függvényt úgy, hogy az összes szomszédos bemeneti változás le legyen fedve egy-egy prímimplikánssal.
TILOS kiküszöbölési szándékkal kondenzátort tenni a kimenetre. Lassítanánk és terhelnénk a hálózatunk.
Az egyik ág késleltetésével se oldhatjuk meg a problémát, hiszen oda-vissza jelentkezhet a hazárd.

A hazárdmentesített alakra új elenvezést vezethetünk be: **hazárdmentes diszjunktív minimálalak** (HMDMA).

MP:

Z függvény:




Z = A*B + C*/A

A statikus hazárd jól látható a Karnaugh-táblán.
Az ABC és /ABC bemenetek között tapasztalható.

Ebben az esetben a kiküszöbölést a szürke kerettel jelzett hurok jelzi a Karnaugh táblán, melyet hozzá kell vennünk a Z függvényhez.
A kapott függvény tehát: Z = A*B + C*/A + **B*C**

Konjunktív hálózatnál a kimenet akkor 0, ha a függvénykimenetet előállító AND kapu bemenetei közül legalább az egyik 0, tehát itt lehet szerepcsere! Statikus 0 hazard viszont ebben az esetben nem fordulhat elő, hiszen konjunk hálózat kimenete akkor 1 ha a függvénykimenetet előállító AND kapu összes bemenete 1, tehát itt nem lehetséges szerepcsere.

| | Diszjunktív hálózat | Konjunktív hálózat |
|---|---|---|
| Statikus 1 hazard | 1 | 0 |
| Statikus 0 hazard | 0 | 1 |

Fontos tisztázni, hogy a hazárd nem feltétlenül jelenik meg a kimeneten. Számunkra azonban az sem megengedett általában, hogy a lehetőség fennálljon. Ha egy hálózat működése során egyszer, de rossz pillanatban jelentkezik a hazárd, az nem megengedhető!

**Statikus 0 hazárd (konjunktív hálózat esetén):**

Statikus 0 hazárdról beszélünk, ha szomszédos bemeneti kombináció változás esetén az F függvény kimenete 0 értéket kéne, hogy tartson, azonban valamelyik késleltetésnek köszönhetően a hálózat ideiglenesen olyan állapotba kerül, ahol a kimenet 1. Ebben az esetben az F függvény kimenetén 0-1-0 átmenetet tapasztalhatunk.

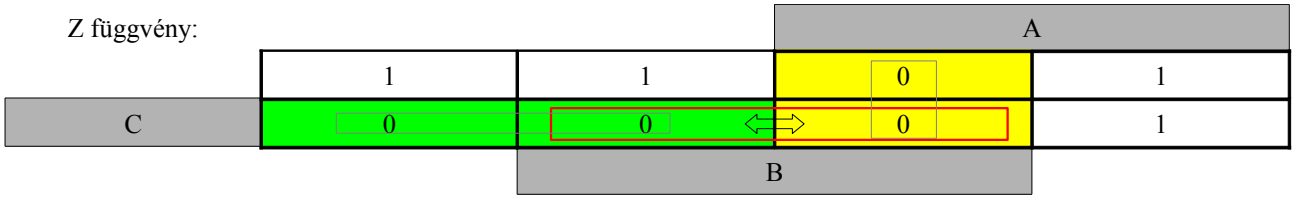
Tekintsünk egy szomszédos állapotváltozást, ahol mindkettő bemeneti kombinációhoz 0 tartozik. Mivel mind a kettő esetén 0 a kimenet, nem szabadna változást észlelnünk.

Ideális esetben ténylegesen 0-0 átmenetünk van, és nem tapasztalunk semmi rendelleneset.
Azonban az OR kapuk értékei megváltoznak, ezáltal az AND kapun szerepcsere történik. Eddig az egyik OR kapu kimenete biztosította, hogy az AND kapu kimenete 0 legyen, azonban ha ezen kapu kimenete hamarabb megy 1-re, mint a másik kapu kimenete 0-ra, akkor az AND kapu bemenetein rövid ideig csak 1-ek jelennek meg,
így a kimenete is 1.

A statikus 0 hazárd **kiküszöböléséhez** módosítanunk kell a függvényt úgy, hogy az összes szomszédos bemeneti változás le legyen fedve egy-egy prímimplikánssal.
TILOS kiküszöbölési szándékkal kondenzátort tenni a kimenetre. Lassítanánk és terhelnénk a hálózatunk.
Az egyik ág késleltetésével se oldhatjuk meg a problémát, hiszen oda-vissza jelentkezhet a hazárd.

MP:

Z függvény:




Z = (/A+/B) * (/C+A)

A statikus hazárd jól látható a Karnaugh-táblán.
Az ABC és /ABC bemenetek között tapasztalható.

Ebben az esetben a kiküszöbölést a szürke kerettel jelzett hurok jelzi a Karnaugh táblán, melyet hozzá kell vennünk a Z függvényhez.
A kapott függvény tehát: Z = (/A+/B) * (/C+A) * **(/B+/C)**

Diszjunkt hálózatnál a kimenet akkor 1, ha a függvénykimenetet előállító OR kapu bemenetei közül legalább az egyik 1, tehát itt lehet szerepcsere! Statikus 1 hazard viszont ebben az esetben nem fordulhat elő, hiszen diszjunkt hálózat kimenete akkor 0, ha a függvénykimenetet előállító OR kapu összes bemenete 0, tehát itt nem lehetséges szerepcsere.

| | Diszjunktív hálózat | Konjunktív hálózat |
|---|---|---|
| Statikus 1 hazard | 1 | 0 |
| Statikus 0 hazard | 0 | 1 |

Fontos tisztázni, hogy a hazárd nem feltétlenül jelenik meg a kimeneten. Számunkra azonban az sem megengedett általában, hogy a lehetőség fennálljon. Ha egy hálózat működése során egyszer, de rossz pillanatban jelentkezik a hazárd, az nem megengedhető!

**Dinamikus hazardok:**

Dinamikus hazárdról beszélünk, ha 0-1 vagy 1-0 átmenet helyett 0-1-0-1 vagy 1-0-1-0 átmenetek zajlanak a kimeneten. Csak többszintű hálózatoknál fordul elő, és egy alhálózat statikus hazárdja okozza.

-Ha az alhálózatban statikus hazárd van, és az F kimenet előjelet vált, az alhálózat statikus hazárdja **dinamikus hazárdként** jelenhet meg a kimeneten.

-Ha a függvényünk nem vált előjelet, akkor meg kell vizsgálnunk, hogy az utolsó kapu bemenetén milyen változások zajlanak le, és hogy az egyik kapu fedezi-e a másik kapu statikus hazárdját a kimenetet bizotsító kapu ogazságtáblázata alapján:

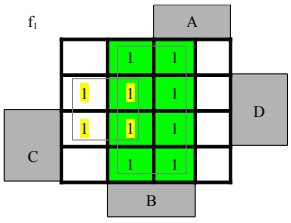
-AND kapu: Ha az egyik bemenete folyamatosan 0, akkor ez a bemenet fedezi a többi bemeneten érkező statikus hazárdot azáltal, hogy a kimenetet 0-n tartja.
-NOR kapu: Ha az egyik bemenete folyamatonsa 1, akkor ez a bemenet fedezi a többi bemeneten érkező hazárdot azáltal, hogy a kimenetet 1-en tartja.
-NAND kapu:Ha az egyik bemenete folyamatosan 0, akkor ez a bemenet fedezi a többi bemeneten érkező statikus hazárdot azáltal, hogy a kimenetet 1-n tartja.
-NOR kapu: Ha az egyik bemenete folyamatosan 1, akkor ez a bemenet fedezi a többi bemeneten érkező statikus hazárdot azáltal, hogy a kimenetet 0-n tartja.

MP: Tekintsünk egy több szintű hálózatot:

$F = \underbrace{[\ (/A*D) + (B)\ ]}_{f_1} * \underbrace{[\ (/A*B) + (A*C)\ ]}_{f_2}$

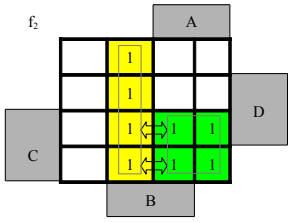
Jól láthatóan 3 szintű hálózatról beszélünk: $F= f_1 * f_2$

Az első rész 1 → 0 átmenetet fog produkálni.
A felső részen viszont statikus 1 hazard léphet fel, és ha az első rész átmenete lassú, akkor a kimeneten 1 → 0 → 1 → 0 átmenetet tapasztalhatunk az 1 → 0 helyett!
Ezt nevezzük dinamikus hazardnak.

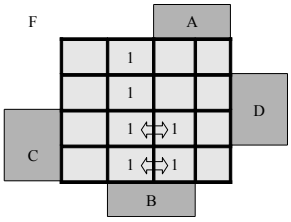



A Karnaugh táblákban jól láthatóak az alhálózatok, és be lett jelölve a két statikus 1 hazárd is az $f_2$ alhálózatban és az F hálózatban is.

Az F kimenetet egy AND kapu állítja elő ebben az esetben.
Az eredő hálózatban (F) nincs 0-1 váltás egyik statikus hazárd helyén sem, tehát emiatt nem lesz ott dinamikus hazárd.




Muszáj ellenőriznünk, hogy az $f_1$ fedezi-e az AND kapun a statikus 1 hazárdunkat.
Azt tapasztaljuk, hogy nem.
Az $f_1$ kimenete végig 1, tehát amíg az $f_2$ kimenete is 1, addig az F kimeneten 1 érték jelenik meg. Azonban miközben az $f_2$ a statikus 1 hazárdot, azaz egy rövid 0-t produkál, addig az AND kapun egy 0 áll elő, tehát ez a statikus 1 hazárd kijut a kimenetre.

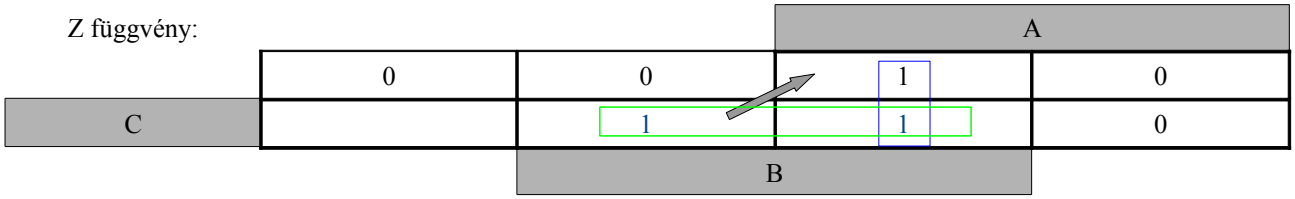



Ha az $f_1$ hálózat kimenetét invertálnánk, abban az esetben a statikus 1 hazárdot produkáló bementi változásoknál végig 0 lenne az $f_1$ értéke, így az AND kapu egyik bemenete, ami ezáltal végig 0 értéket produkálna, és így a hazárdunk nem jutna ki.

Dinamikus hazárd **kiküszöbölése**: az alhálózatokban nem engedhetünk meg statikus hazárdot.

**Funkcionális hazárd:**

Eddig szomszédos változásokat tekintettünk a Karnaugh táblában.
Most tekintsünk egy nem szomszédos változást!
Amennyiben nem szomszédos bemeneti változás esetén az egyik változás gyorsabban történik meg mint a másik, előfordulhat, hogy ideiglenesen olyan kimenetet ad a rendszer, ami nem felel meg az eredeti szándékunknak.




Z= CB + AB

Tekintsük a nyíllal jelölt, nem szomszédos bemeneti változást:
/ABC → AB/C
0 1 1 → 1 1 0

Jól látható, hogy több bemeneti változó változott meg egyszerre.
Ha A gyorsabban változik, mint C, akkor 111 bemeneti kombinációt érzékelünk rövid ideig, amihez 1 kimenet tartozik, tehát ebben az esetben nem észlelünk hazárdot, mivel 1-1 változást várunk, és azt is kapunk.
Azonban ha C változik gyorsabban mint A, akkor 010 bemeneti kombinációt érzékelünk rövid ideig, amihez viszont 0 kimenet tartozik, ebben az esetben tehát érzékeljük a hazárdot, mivel 1-1 változást vártunk, de helyette 1-0-1 változást kaptunk.

Jelen példánkban, ha az 111 felé megy a változás, akkor rendeltetés szerűen működik (A*B*C is megfelel a függvényünknek). Azonban ha a 010 felé megyünk, akkor kilépünk a Z függvényből, és egy 1 hazard lép fel a kimeneten!

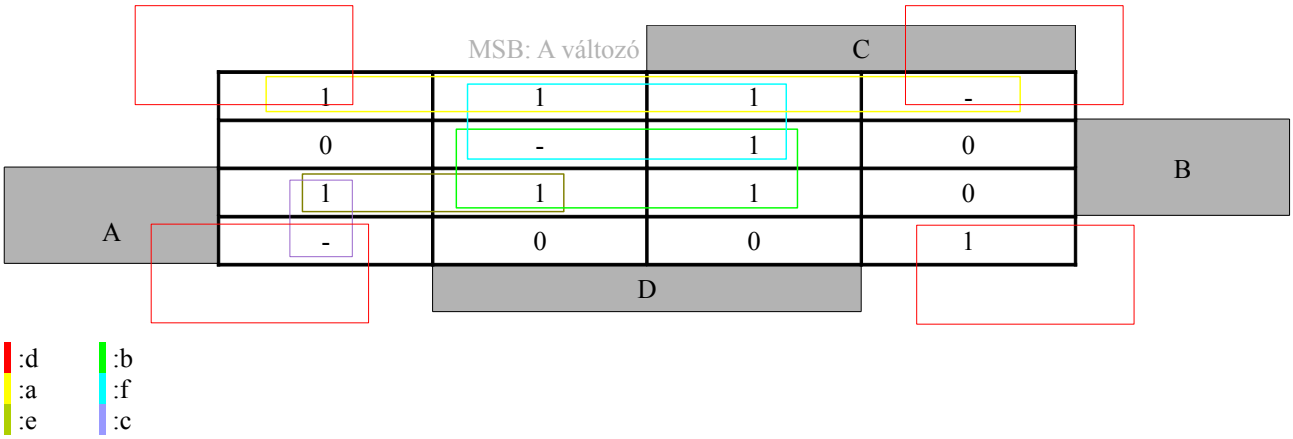
Funkcionális hazárdok **kiküszöbölése**:
A kiküszöbölés egyik módja, hogy tiltjuk a nem szomszédos bemeneti változásokat. Ezt azonban nem tehetjük meg minden esetben.
Másik lehetséges megoldás, hogy órajellel szinkronizálunk.
Harmadik lehetséges megoldás, hogy **hand-shake** rendszert alkalmazunk: minden rendszer ad egy visszajelzést a megelőző szintnek, hogy stabil a kimenete.

## Prímimplikáns tábla használata:

Grafikus minimalizálásnál, kevés változónál sok esetben egyértelmű, hogy melyik a legegyszerűbb hazárdmentes alak. Ez azonban több változó esetén nem látszik grafikusan, és nem is egyértelmű, hogy mely alak a legegyszerűbb.

A legegyszerűbb hazárdmentes alak megtalálásában a prímimimplikáns tábla lesz segítségünkre, melyet a következő példán keresztül szemléltetünk:

Adott a következő függvény: $F = \sum^4 [ (0, 1, 3, 7, 10, 12, 13, 15)+(2, 5, 8) ]$




Most először jelentek meg dont care értékek, ez azonban nem zavar minket semmiben.
Az F függvény termes megadásánál a dont care értékek külön zárójelben, + jel után vannak felsorolva, ahogy azt fent is láthatjuk.
A prímimimplikánstábla használatához meg kell határoznunk az össze prímimplikánst. Ez történhez grafikus minimalizálással, vagy a későbbiekben említett számjegyes minimalizálással is.

Ha meghatároztuk az összes prímimplikánst, akkor a következő lépés, hogy vegyünk fel egy olyan tábláta ami lentebb látható. Az oszlopokazokat a termeket jelentik, amelyek fix értékek. A dont care értékekkel nem kell foglalkoznunk.
A sorok az egyes prímimplikánsokat tartalmazzák, és egy cellába x kerül, ha az adott cellához tartozó prímimimplikáns tartalmazza az ugyan ezen cellához tartozó termet.

| term→<br>prímimplikáns | 0 | 1 | 3 | 7 | 10 | 12 | 13 | 15 |
|---|---|---|---|---|---|---|---|---|
| a | x | x | x | | | | | |
| b | | | | x | | | x | x |
| c | | | | | | x | | |
| d | x | | | | x | | | |
| e | | | | | | x | x | |
| f | | x | x | x | | | | |

Válasszuk ki a megkülönböztetett termeket, azaz azokat, amelyeket csak 1 prímimimplikáns fed le. Ezeket onnan ismerhetjük fel, hogy az oszlopukban egy darab x található. A táblázatban pirossal jelöltük.
Zöld színnel jelöljük meg azokat a prímimimplikánsokat, amelyeket biztosan be kell vennünk a függvényünkbe.
Az ezen prímimimplikánsokhoz tartozó mintermek bizotsan le lesznek fedve. Ezeket sárgával jelöltük.
Látható, hogy még nincs minden lefedve. Írjunk fel egy segédfüggvényt, melynek értéke 1, ha minden term le van fedve:
S= d * (a+f) * (a+f) * b * d * (c+e) * b * b = d * b * (a+f) * (c+e) = b*d*a*c + b*d*a*e + b*d*f*c + b*d*f*e

A segédfüggvény úgy készül, hogy oszloponként megnézzük, hogy mely prímimplikáns fedhetné le az adott oszlopot. Ha az adott oszlopban van lényeges prímimimplikáns, akkor azt az oszlopot már mással nem kell lefednünk.
Ilyen módon találtunk négy minimális lefedést, ez azonban még nem hazárdmentes.

A hazárdmentesítéshez be kell vennünk újabb prímimplikánsokat is. Ehhez fel kell írnunk a szomszédos prímimplikánsok kombinációit. Olyan párokat kell keresnünk, melyek szomszédosak. A dont care értékekkel továbbra sem törődünk.

Az alábbi táblázatot kapjuk a fentebbi példához, a megadott feltételek alapján:

| term → | 0 | 1 | 3 | 7 | 13 | 12 |
|---|---|---|---|---|---|---|
| prímimplikáns | 1 | 3 | 7 | 15 | 15 | 13 |
| a | x | x | | | | |
| b | | | | x | x | |
| c | | | | | | |
| d | | | | | | |
| e | | | | | | x |
| f | | x | x | | | |

Tudjuk, hogy a b és d prímimimplikánsok lényeges prímimimplikánsok, tehát azokat mindenképpen be kell majd vennünk.
A fentebb már ismeretett módon most is megnézzük, hogy melyek a kitüntetett termpárok, azaz azok a párok, amelyeket csak egyetlen prímimimplikáns fed le, és most is elkészítjük a segédfüggvényt.
A már fentebb is használt színeket használtuk a jelölésekhez.
Jól látható, hogy az *a, e* és *f* prímimimplikánsok is lényegesek lettek a hazárdmentesítés szempontjából, tehát azokat is mindeképpen meg kell valósítanunk.
A segédfüggvényt ebben az esetben úgy kell felírnunk, hogy a két táblát egy, hosszabb táblaként kezeljük, mintha a hazárdmentesítéshez használt tábla az előző folytatása volna.

Az S segédfüggvényre hazárdmentes alak esetén a következőt kapjuk (ne feledjük, hogy hazárdmentesítés szempontjából lényegessé váltak prímimplikánsok, tehát ezeket elég egyszer bevenni, és az általuk lefedett oszlopokat utána nem kell lefedni, így a segédfüggvénynél sokkal kevesebb olyan oszlop lesz, ahol választási lehetőségünk van!):

S = a*b*d*e*f

Azt kaptuk tehát, hogy a fentebbi példánál egyedül a c prímimimplikánst hagyhatjuk el.
Használjuk ki, hogy kevés változó van, és meg tudjuk vizsgálni grafikusan is a végeredményünket.
Ha nem feledkeztünk meg róla, hogy a dont care értékeket nem kell figyelni, akkor azt tapasztalhatjuk, hogy a megoldás helyes.

Az F függvényünk tehát a következő módon áll elő:
F = a+b+d+e+f

Megj.: Bár az S segédfüggvény és az F függvény esetén is ugyan azt az a betűt használtuk jelölésre, a kettő jelentése nem egyezik. Az F függvénynél a már megszokott dolgot jelenti, de a segédfüggvény esetén az a szimbólumot jelölhetnénk $v_a$-val, melynek értéke 1, ha az a prímimplikáns le van fedve, és értéke 0, ha az a prímimplikáns nincs lefedve.

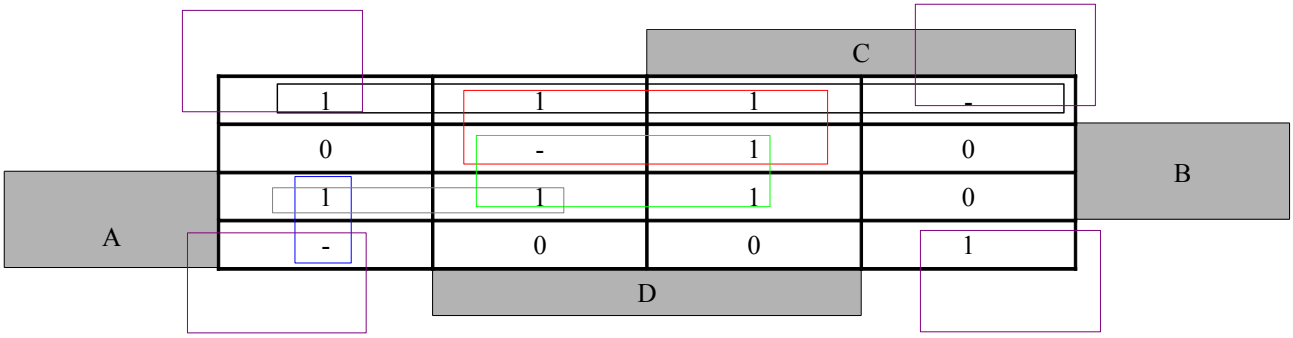
## A számjegyes minimalizálás:

A már oly sokat emlegetett számjegyes minimalizálás akkor nyújt segítséget számunkra, ha a grafikus módszerek már nem működnek megfelelően, mivel túl sok a bemeneti változó.
Ennek ellenére kis változószámmal mutatjuk be a módszert, mivel így szemléletesen ellenőrizhető annak helyessége, és a lépések száma is egy kisebb számhoz konvergál.

Szükségünk lesz azonban egy új definícióra, mivel fogunk rá hivatkozni:
Bináris szám **Haming-súlyán** az adott bináris szám egyeseinek számát értjük.

A következő példán keresztül mutatjuk be a számjegyes minimalizálást:

$F= \Sigma^4 [ ( 0, 1, 3, 7, 10,13, 15) + (2, 5, 8) ]$




Pontosan tudjuk, hogy minek kell kijönnie, hiszen ezt a példát már alaposan átnéztük, így könnyen ellenőrizhető.

A logikai függvények minterm (maxterm) indexét úgy képezzük, hogy mintermben ABC sorrendben szereplő változókhoz ponált esetben 1-et, a negált esetben 0-át rendelünk, majd az így kapott bináris számot decimálissá alakítjuk.

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

Tehát az egy bitben eltérő számokat kell megkeresni, és az eltérő bit helyéhez rendelt változót elhagyni.

| | | | | A | B | C | D |
|---|---|---|---|---|---|---|---|
| | | | | 0 | 0 | 0 | 0 |
| | | | | 0 | 0 | 0 | 1 |
| 0 | 0 | - | 1 | 0 | 0 | 1 | 0 |
| | | | | 0 | 0 | 1 | 1 |
| | | | | 0 | 1 | 0 | 0 |
| | | | | 0 | 1 | 0 | 1 |

…

A szomszédos termeket gyorsabban meg lehet találni, ha először a bináris minterm indexeket a bennük levő 1-esek száma (bináris, vagy haming-súlyuk) alapján sorbarendezzük. Ekkor már csak a súlyuk szerint egymás melletti indexeket kell egymáshoz hasoníltani.

Számjegyes minimalizálás lépései:

-Haming-súly szerinti osztályzás:
-Azért, hogy könnyedén vizsgálhassuk a szomszédosságot.

-Kettes hurkok képzése:
-Elkezdjük hízlalni az implikánsainkat. Ehhez a szomszédos haming-súlyú termekből képezünk párokat, ha azok különbsége 2 hatványa.

-Ha képeztünk egy párt, akkor mögéjük írjuk oda zárójelben a különbségüket. Pl.: 2-10 → 2,10 (8)

-Azokat a termeket, amelyeket már bevettünk legalább egy párba, kipipáljuk, ezzel jelezve, hogy ők le vannak már fedve legalább 1 implikánssal (táblázatban zöld kiemeléssel jelöltük).
Ha a fázis végén olyan termet találunk, ami mellett nincs pipa, akkor az önmagában egy prímimplikáns lesz, hiszen nem vonható be egyetlen nagyobb hurokba sem.

-Egy termet sok párba is bevehetünk, sőt, be is kell vennünk!

-Gyakori hiba, hogy a Haming-súlynál ha egyáltalán nincs például 2-es súlyú term, akkor az 1-es és a 3-as súlyú termeket hasonlítja össze a diák. Ne kövessük el ezt. Ha egy csoport hiányzik, akkor azt jelöljük, és figyeljünk, hogy csak ténylegesen szomszédos termeket hasonlítsunk.

-Négyes hurkok képzése:
-Az előző fázist kell vizsgálnunk, azonban szükséges tisztázni, hogy most mely csoportokat hasonlítjuk össze: a kettes hurkok képzésénél a 0-1 és az 1-2 Haming-súlyú termeket hasonlítottuk. Most a két összehasonlításra adódó párokat kell hasonlítanunk, tehát a 0-1 és az 1-2 lesz szomszédos. Ha az 1-2 csoportban nem találtunk párt, akkor **NEM** hasonlíthatjuk a 0-1-et a 2-3 csoporthoz!
**Nagyon figyeljünk oda, hogy mindig csak valóban szomszédos csoportokból építkezzünk!**

-Az előző fázisban kapott párokat hasonlítjuk. A feltétel, hogy szomszédos csoportban legyenek, a zárójelben jelzett számok megegyezzenek, és az első tagok különbsége 2 hatványa legyen (ekkor a második tagok különbsége is ugyan annyi)

-Ha egy párt legalább 1 párba bevettünk már, akkor pipáljuk ki, mint az előbb. Azok a párok, amik nem kaptak pipát a fázis végén, prímimimplikánsok.

-A pár mögé a zárójelbe mostmár két szám kerül, az eredetileg is ott lévő szám, és az első tagok különbsége.

-Nyolcas és nagyobb hurkok képzése:
-Az előző logikát kell követnünk a továbbiakban is, minden nagyobb hurok esetén.

-A zárójelben levő számok folyamatosan bővülnek a különbségekkel. Minden fázisban egyel bővülnek, hiszen úgy választjuk mindig a tagokat, hogy az n. tagok különbsége megegyezzen minden n-re.

Figyeljünk a sorrendre. A párokban mindig az első tag a kisebb Haming-súlyú, és ezt a nagyobb hurkoknál is kövessük.
Ha egy-egy hurok ugyan azokat az indexeket más sorrendben tartalmazza, akkor elég csak az egyikkel tovább számolni, hiszen csak a sorrendjük különbözik, de a két hurok egy és ugyan az.

A példát mintermek esetén mutatjuk be, de maxtremekkel is működik az algoritmus:

Haming-súly szerint osztályozzuk az össze termet, ami F-ben szerepel. A dont care értékeket is, hiszen azokat is felhasználhattuk egyszerűsítésre. A csoportosításra az alábbi táblázatot kapjuk:

| Haming-súly | Minterm index |
|---|---|
| 0 | 0 |
| 1 | 1 |
| | 2 |
| | 8 |
| 2 | 3 |
| | 5 |
| | 10 |
| | 12 |
| 3 | 7 |
| | 13 |
| 4 | 15 |

A II. fázis, azaz a párok létrehozása:

| Haming -súly | I. fázis | II. fázis |
|---|---|---|
| 0 | 0 | |
| 1 | 1 | 0,1 (1) |
| | 2 | 0,2 (2) |
| | 8 | 0,8 (8) |
| 1 | 1 | |
| | 2 | 1,3 (2) |
| | 8 | 1,5 (4) |
| 2 | 3 | 2,3 (1) |
| | 5 | 2,10 (8) |
| | 10 | 8,10 (2) |
| | 12 | 8,12 (4) |
| 2 | 3 | |
| | 5 | 3,7 (4) |
| | 10 | 5,7 (2) |
| | 12 | 5,13 (8) |
| 3 | 7 | 12,13 (1) |
| | 13 | |
| 3 | 7 | |
| | 13 | 7,15 (8) |
| 4 | 15 | 13,15 (2) |

Következő lépésben tekintjük a párokat a **szomszédos csoportokban**.

| Haming -súly | I. fázis | II. fázis | III. fázis |
|---|---|---|---|
| 0 | 0 | | |
| 1 | 1 | 0,1 (1) | 0,1,2,3 (1,2) |
| | 2 | 0,2 (2) | |
| | 8 | 0,8 (8) | 0,2,1,3 (2,1) |
| 1 | 1 | | |
| | 2 | 1,3 (2) | 0,8,2,10 (8,2) |
| | 8 | 1,5 (4) | ... |
| 2 | 3 | 2,3 (1) | |
| | 5 | 2,10 (8) | |
| | 10 | 8,10 (2) | 0,8,1,5 |
| | 12 | 8,12 (4) | |

…

Most négyeseket képeztünk úgy, hogy fogtuk a szomszédos csoportokat, kivettük azokat a párokat, amiknél a különbségek (zárójelbe írt számok) megegyeznek, majd megnéztük, hogy a két első és a két második tag közti különbség 2 hatvány-e!

Pl.: 0,1,2,3 (1,2) stimmel
DE: 2,10,5,13 NEM stimmel, hiszen a zárójelbe írt számok sem stimmelnek.
Ha a zárójelbe írt számok stimmelnének, a 2,10,5,13 akkor sem stimmelne, hiszen a két első tag, jelen esetben a 2 és az 5 különbsége 3, ami nem kettő hatvány.

Szomszédos csoportokban továbbra is képezhetünk adott esetben még implikánsokat a fentebb ismertetett módszerrel. Ha a különbségek stimmelnek (**sorrend ITT NEM számít, tehát (2,8) és (8,2) párba állítható!**) és a mintermek közti különbségek sorban 2 hatványai.

A számjegyes minimalizálás akkor ér véget, ha már nem tudunk a következő fázisban nagyobb hurkot létrehozni.

A számjegyes minimalizálás ilyen módon az összes prímimplikánst megtalálja. Azok az implikánsok, amelyek nem kapnak pipát, prímimplikánsok!
Nevezzük el őket a, b, c, d … -nek.

A fentebbi példa a következő príimplikánsokat eredményezi:
a: 8, 12
b: 12, 13
c: 0, 1, 2, 3
d: 0, 2, 8, 10
e: 1, 3, 5, 7
f: 5, 7, 13, 15

Hasonlítsuk össze, a grafikus módszerrel találtakkal. Láthatjuk, hogy ugyan azt kaptuk.

A legegyszerűbb alakot a már ismertetett prímimplikánstáblával fogjuk megkapni.

Vegyük fel a prímimplikánstáblát:

| | 0 | 1 | 3 | 7 | 10 | 12 | 13 | 15 |
|---|---|---|---|---|---|---|---|---|
| a | | | | | | x | | |
| b | | | | | | x | x | |
| c | x | x | x | | | | | |
| d | x | | | | x | | | |
| e | | x | x | x | | | | |
| f | | | | x | | | x | x |

Keressük meg a megkülönböztetett termeket és a lényeges prímimplikánsokat, és vegyük fel a segédfüggvényt.

A segédfüggvényre a következő adódik:

S = d * (c+e) *f * (a+b)

Végezzük el a segédfüggvény beszorzását:

S=acdf + adef + bcdf + bdef

<u>Hazárdmentes alakot is kereshetnénk a korábban már megtárgyalt kiegészítéssel!</u>

A lényeges prímimplikánsok minden tagban szerepelnek.
Ezeken tagok közül kell választanunk egyet a hálózat megvalósításához.

A példa kedvéért válasszunk egyet:

F = a + c + d + f

A példa kedvéért írjuk vissza az algebrai alakot:

Vegyük a–t:

a = 8, 12 (4) → 8-as és 12-es minterm, és 4-es miatt a B (B változó értéke $2^2$) változó esik ki:

a = A * /C * /D

Vegyük f-et:

f= 5, 7, 13, 15 (2, 8) → Az 8-as miatt az A változó esik ki, a 2-es miatt a C változó esik ki.

f = B * D

...

Mint azt már fentebb említettük a módszer működik maxtermekkel is, csak az indexek képzéséből eredő eltérésekre kell figyelnünk a Haming súlyok megállapításánál és az algebrai alak visszaírásánál.

## Hálózatépítés kész elemek felhasználásával:

Léteznek olyan problémák, amik bonyolultak, de nagy gyakorisággal előfordulnak, ezért ezeket előre legyártott elemekként vásárolhatjuk meg. Ilyen elemeket kis átalakítással használhatunk a feladataink megvalósítására.
Ezt fogjuk látni a következőkben.

**Szimmetrikus függvények:**
Még nem ejtettünk szót a szimmetrikus függvényekről, melyek legyártott kész elemekként beszerezhetőek a boltokban.
Szimmetrikus függvényeknek nevezzük azokat a logikai függvényeket, amelyek a független változók tetszőleges páronkénti felcserélése esetén változatlanok maradnak.
**Szimmetria szám**:
A szimmetriaszám azt határozza meg, hogy a szimmetrikus függvényünkben hány változó rendelkezik ponált értékkel.
Minden szimmetrikus függvényhez legalább egy ilyen szimmetriaszám megadható.
Jelölése: $S^{n}_{k,l,j...}$
A felső index a változók számát jelöli, míg az alsó indexbe a szimmetriaszámok kerülnek.

Pl.: $S^{4}_{0,3}$ = /A*/B*/C*/D + A*B*C*/D + A*B*/C*D + A*/B*C*D + /A*B*C*D
Jól látható, hogy vagy 0 vagy 3 változó van ponálva.

<u>Hálózatépítés megadott elemekkel:</u>

Karnaugh-táblával adott a következő feladat:

Megvalósítandó hálózat:

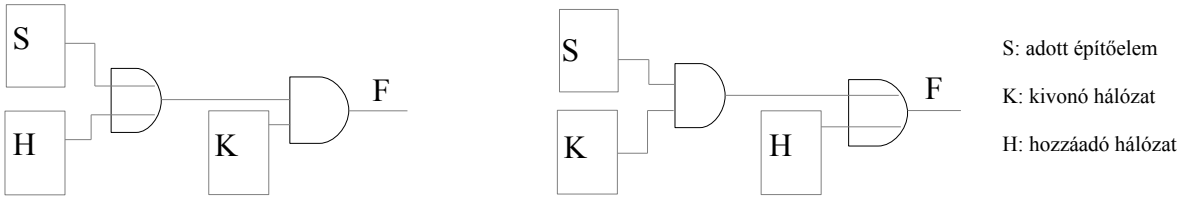
| A \ B | C: 0 | C: 0 | C: 1 | C: 1 |
|---|---|---|---|---|
| | D: 0 | D: 1 | D: 1 | D: 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

A megvalósításhoz segítségünkre lesz egy szimmetrikus függvény, ezt kell felhasználnunk:

$S^{4}_{0,2}$

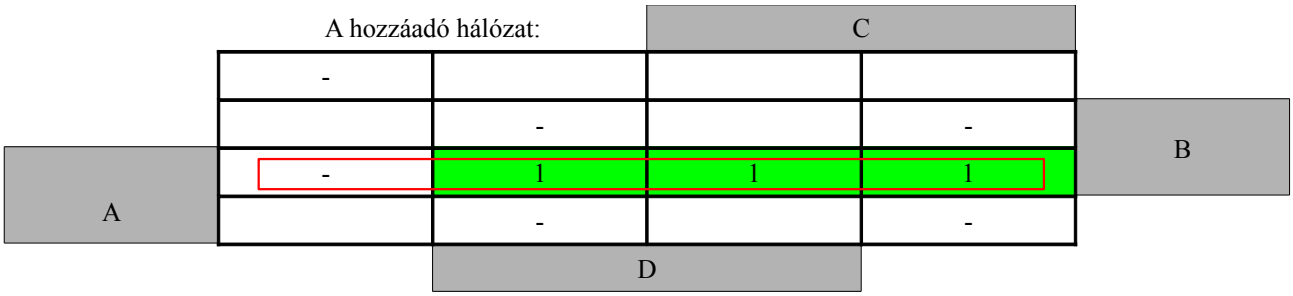
| A \ B | C: 0 | C: 0 | C: 1 | C: 1 |
|---|---|---|---|---|
| | D: 0 | D: 1 | D: 1 | D: 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |

A megvalósítást alapvetően AND és OR kapukkal fogjuk végrehajtani.
AND kapu segítségével el tudunk távolítani a szimmetrikus függvényünkből 1-eseket (1*0=0), míg OR kapu segítségével hozzá tudunk adni egyeseket (0+1=1).
Két megoldást képzelhetünk el, és attól függően, hogy melyiket választjuk különböző egyszerűsítések adódnak.



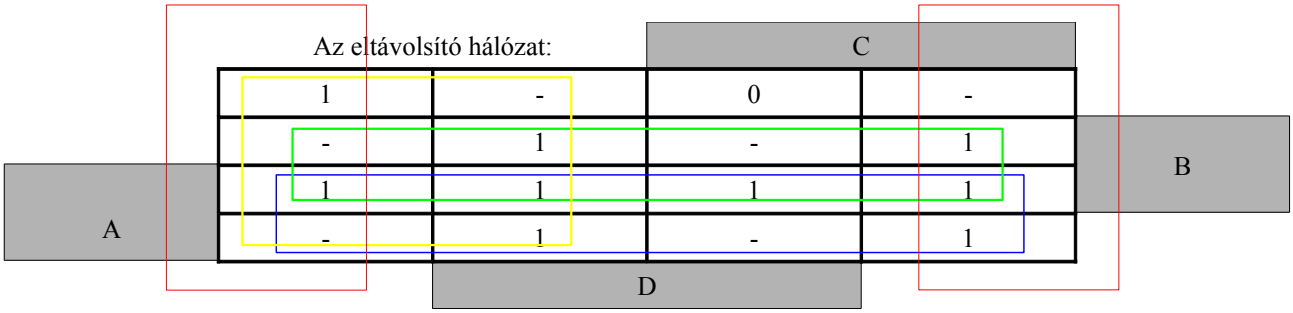

A továbbiakban a K és H hálózatok megtervezése lesz a feladatunk.

Jelen példánkban az első változatot valósítjuk meg:




Azokon a helyeken kell 1-est fixálnunk, ahol a megvalósítandó hálózatunk 1-est ad, de az építőelemünk nem, hiszen ezekre a helyekre egészen biztos, hogy 1-eseket kell hozzáadnunk, és 0+1=1.
0-kat kell írnunk azokra a helyekre, ahol nem akarunk hozzáadni a hálózatunkhoz. Ez okozza a feladat bonyolultságát, ugyanis ha ezt nem tennénk meg, hanem a kivonó hálózattal vonnánk ki az itt hozzáadott 1-eket, akkor a lehetséges variációk száma nagyon gyorsan nőne, és a legegyszerűbb kiválasztásához az összeset végig kéne néznünk.
Azokra a helyekre, ahol az építőelemünk és a végső hálózatunk is 1-est ad, dont care értékeket vehetünk fel, hiszen OR kapcsolat esetén nem fognak számítani. 1+0=1 és 1+1=1

Ezek alapján a hozzáadó hálózatra a következő adódik: H=A*B




Az eltávolító hálózatot úgy kell megépítenünk, hogy azt kapcsolatba hozva a meglévő elemekkel, azokat az 1-eseket, amik benne vannak a hálózatban még, de a végső hálózat kimenetén értékek 0 kell, hogy legyen.
1-eseket kell írnunk azokra a helyekre, ahol a végső hálózatnak 1-est kell adnia, hiszen csak az 1*1 ad 1-es kimenetet, és mi AND kaput akarunk használni.
Dont care értékeket helyezhetünk azokra a helyekre, ahol a megvalósítandó hálózat 0-t ad.

Ezek alapján a hozzáadó hálózatra a következő adódik: K=A+B+/C+/D $\rightarrow \bar{K}$=/A*/B*C*D

<u>Másik változat megvalósítása:</u>
Miután a hozzáadást és az eltávolítást 2 szinten valósítjuk meg, ezért az első szinten vagy hozzáadás vagy elvétel szerepel, míg a második szinten a másik. Fentebb láthattuk, hova rakhatunk dont care-t abban az esetben, ha előbb a hozzáadást valósítottuk meg, most nézzük, hova rakhatnánk dont care-t, ha fordítva csinálnánk:

Az első szinten AND kapcsolat van, tehát itt valósítjuk meg a kivonást:
Ha az első szinten valósítjuk meg az elvételt, akkor azokra a helyekre, ahol a hozzáadó hálózat 1-est fog adni a második szinten, dont care értékeket rakhatunk.
A második szinten, az OR kapcsolatnál, azaz a hozzáadó hálózatnál oda rakhatunk dont care értékeket, ahol a megelőző hálózat már 1-est ad.

<u>Összetettség:</u>
Használhatnánk más kapukat is, a hozzáadó és kivonó függvények invertáltjait is kapcsolhatnánk a kapuk bemeneteire, és az egyik szinten akár az összes helyre rakhatnánk dont vare értékeket. Mivel a fentebbi megkötések nélkül a lehetőségek száma a bemenetek számával nagyon gyorsan nő, ezért a legegyszerűbb alakot csak “tenyésztett” példákon fogjuk észrevenni. Az egyszerűbb hálózatot úgy választhatjuk ki, ha mind a két esetet felírjuk, azonban általában a feladat kiköti, hogy melyiket kell használnunk.

Sorrendi hálózatok

## Sorrendi hálózatok bevezetése:

Kombinációs hálózatok esetén a kimenetet egyértelműen meghatározta a bemenet, sorrendi hálózatok esetén viszont a kimenet meghatározásához az előző állapot ismerete is szükséges.
Végtelen számú előző állapot lehet, de egy-egy feladatunkhoz véges számú állapotot is elég definiálnunk.

Az állapotok leírásához úgynevezett **szekunder változókat** használunk.

A kombinációs hálózatoktól eltérően nem egy függvénnyel fogunk dolgozni, ugyanis nem elég a kimenetet előállítani, a következő állapotot is meg kell mondanunk.
A függvényeink nem csak

2 függvényre lesz szükségünk, egyik megmondja a kimenetet a bemenet és az előző állapot ismeretében, a másik előállítja az újabb állapotot.

Az állapotokat két csoportba soroljuk:

$f_y(x_0, y_i) = Y_j$ → **instabil állapot**: adott bemenethez következő állapotot rendel
$f_y(x_0, y_n) = Y_n$ → **stabil állapot**: adott bemenethez az aktuális állapotot rendeli

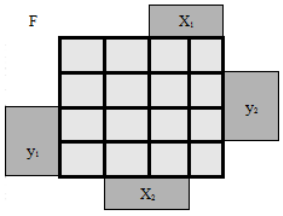
**Normál hálózatról** beszélünk, ha két stabil állapot közt legfeljebb 1 instabil állapot van.
Továbbiakban a célunk az lesz, hogy lehetőleg normál hálózatot hozzunk létre.

A sorrendi hálózatok megadására **állapottáblát** vagy **állapotgráfot** használhatunk.
Állapottábla esetén a sorok az állapotokat, az oszlopok a bemeneteket jelentik.
A bemenetek sorrendjét szándékosan fel fogjuk cserélni annak érdekében, hogy az állapottáblából könnyedén Karnaugh táblát készíthessünk, és így a grafikus minimalizálást alkalmazhassuk.
Ahhoz, hogy az állapottábla könnyedén átírható legyen, az állapotkódoknak is a megadott sorrendben kell követniük egymást. Figyeljünk a sorrendekre a továbbiakban.
Az állapotgráf csúcsai állapotok, az irányított élek pedig azt jelentik, hogy mely bemenetre ugrunk egyik állapotról a másik állapotra.

Állapottábla:

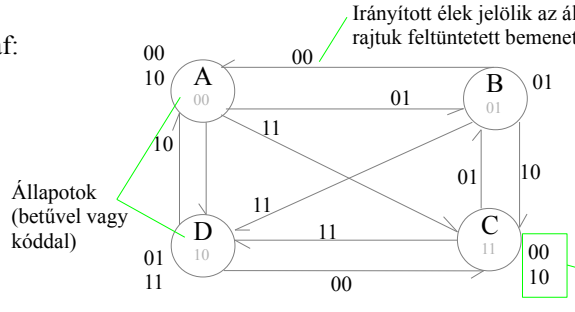
| | $X_1X_2$ = 00 (Bemenetek) | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 00, 0 | 01, 0 | 11, 0 | 00, 0 |
| 01 | 00, 1 | 01, 1 | 10, 1 | 11, 1 |
| 11 | 11, 1 | 01, 1 | 10, 1 | 11, 1 |
| 10 | 11, 0 | 10, 0 | 10, 0 | 00, 0 |

Állapotok: $y_1$ $y_2$ (szekunder változók)
Következő állapot: $Y_1$ $Y_2$
Kimenet: Z




A szekunder változókat $y_i$-vel jelöljük. Nem kódolt állapottáblán a szekunder változók az ABC nagybetűivel jelölendők. Figyeljük meg a sorrendeket az állapotoknál és a bemeneteknél. Ilyen jelöléssel a már megismert Karnaugh tábla adódik majd.

Állapotgráf:




Az állapotgráf és az állapottábla ekvivalensek egymással, akár a kimeneteket is feltüntethetjük.

## Sorrendi hálózatok megismerése:

A továbbiakban **aszinkron és szinkron sorrendi hálózatokról** fogunk beszélni.
Az alapvető különbség a kettő között, hogy előbbinek a változást mindig a bemenetek megváltozása jelenti, míg utóbbinak a változást az órajel fogja jelenteni.

Aszinkron hálózat:
Aszinkron hálózat esetén nem szabad megengednünk statikus hazárdot, mert ekkor nem megfelelő állapotban stabilizálódhat a hálózatunk. Funkcionális hazárdokat így nem küszöbölhetjük ki. Tehát ezek problémájának megoldására **SETUP és HOLD idők**et definiálunk, azaz megszabjuk, hogy két jel egyszerre való megváltozásának esetén mennyi időnek kell eltelnie a két jel megváltozása közt.
Ütemezett aszinkron sorrendi hálózatok esetén a visszacsatolásra egy **ütemezett tartó áramkört** helyezünk el, ami az órajel ideje alatt tartja a bemenetét a kimeneten. Ha az órajel frekvenciáját kisebbre választjuk, mint a hazárd változásának sebessége, akkor a hazárdot kiküszöböltük.
Aszinkron hálózat esetén nem szomszédos bemeneti változásokat sem engedhetünk meg.

Annak, hogy egy állapottábla aszinkron módon értelmezhető legyen feltételei vannak:

- Legyenek stabil állapotok.
- Minden bemeneti változásra a hálózat jusson stabil állapotba, azaz ne legyen oszcilláció.

Szinkron hálózat:
Szinkron sorrendi hálózatok esetében a hálózatnak a változást nem a bemenet változása jelenti, hanem az órajel megérkezése, így teljesen normális, ha ugyan ahhoz a bemenethez különböző kimenet tartozik.
Szinkron működésnek nincsen semmi feltétele, minden állapottábla értelmezhető szinkron módon.

A sorrendi hálózatok modelljei:
**Mealy modell** szerint a kimenet függ a bemenettől, és az éppen aktuális állapottól is.
**Moore modell** szerint a kimenet az aktuális állapot ismeretében egyértelműen meghatározható.
Moore modell szerint működő hálózatot onnan ismerhetünk fel állapottábla alapján, hogy soronként csak 1-féle kimeneti kombináció lehet. Ha egy sorban többféle kimeneti kombináció is van, akkor Mealy modell szerint működő hálózttal van dolgunk, azonban ha ezzel a kérdéssel találkozunk, akkor a válaszadásnál a modellek definíciójára hivatkozzunk!

Szinkron hálózatok vezérlése:
Szinkron hálózatokat flip-flopok segítségével valósítunk meg (aszinkron hálózatokat is megvalósíthatunk flip-flopokkal, de azoknál nem szükségesek ezek), melyekről a következőkben részletesen is olvashatunk, ezek a flip-flopok pedig 3 féle módon működhetnek:

**Élvezérelt** működés: az ilyen működésű flip-flop felfutó (vagy lefutó) élre vesz mintát és ugyan itt -, de garantáltan később, mint a mintavétel vége,- mutatja meg kimenetét.
Működéséből eredő problémák miatt felhasználása erős kompromisszumokat von maga után.

**Master-slave** működés: az ilyen működésű flip-flopok két flip-flopból épülnek fel. A master végzi a mintavételt, és ezt továbbítja a slave-nek, ami a kimenetre másolja az értéket. Az ilyen flip-flop az órajel 1-es (vagy 0-s) értéke alatt végig érzékeny, és a kimenet a lefutó (felfutó) élen jelenik meg. Tiltanunk kell a bemenet változását, amíg a flip-flop érzékeny, ez ugyanis hibás működést okoz.

**Data lock out** működés: széleskörűen elterjedt működés, mely két élvezérelt flip-flopként érthető meg.
Egy ilyen flip-flopot két élvezérelt flip-flop valósít meg, melyek közül az első a normális órajelet kapja, míg a második egy inverteren keresztül kapja azt, így pont a másik órajelélre lesz érzékeny. Ezzel azt érjük el, hogy a mintavétel a bemenetekről felfutó élre történik, míg a billenés lefutó élen megy végbe.

Tapasztalat, hogy vizsgán a különböző működéseknek megfelelő grafikonok felrajzolását elrontják a diákok. Ha nem szeretnénk ezen pontokat veszíteni, gyakoroljuk be ezt a feladatot.
Az ismeretett működések elméleti hátterét és a problémákat a D flip-flop felépítésénél részletesebben is tárgyaljuk majd, de a jegyzet célja alapvetően a gyakorlati részek bemutatása. Akit a teljes elméleti háttér érdekel, utána járhat más forrásokban is.

MP: Nézzünk konkrét példákat a fentebb ismertetett dolgokra:

| Aszinkron@Moore | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 | B, 0 | C, 0 | A, 0 |
| B | A, 1 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 1 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

Lehet-e az állapottáblájával adott hálózat aszinkron?

- Stabil állapotok mindenhol vannak, ezeket zölddel jelöltük.
- Az összes szomszédos bemeneti változásra a hálózat stabil állapotba jut minden állapotból.
  - A táblázatban egy példát jelöltünk: *A* állapotból indulunk és a 00 → 01 állapotátmenetet vizsgáljuk. A 00 → 01 bemeneti változásra a hálózat A állapotból indul, B állapotba kerül, majd ott stabilizálódik.

Lehet aszinkron, de vajon normál hálózattal van dolgunk?
Azt láthattuk, hogy *A* állapotból indulva a 00 → 01 bemeneti változásra egyetlen nem stabil köztes állapot van. Azonban *A* állapotból indulva az 10 → 11 bemenetváltozásra először a hálózat C állapotba ugrik egy instabil állapotra, majd innen ugrik D állapotba. A táblázatban piros színnel jelöltük. Tehát a hálózat nem normál.

Milyen modell szerint működik a hálózat?
A hálózat Moore modell szerint működik, mivel a kimenet közvetlenül csak az aktuális állapottól függ.
(Közvetetten természetesen a bemenettől is, de nem ez alapján ismerjük fel.)

Vizsgáljuk meg a következő bemenetsorozatot a kimenet szempontjából: $x_1$, $x_2$ : 00, 10, 11, 01, 00, 01

| Aszinkron@Moore | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 START | B, 0 | C, 0 | A, 0 |
| B | A, 1 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 1 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

A táblázatban a piros nyilakat nyomonkövetve láthatjuk a bemeneteket és az állapotokat.

Mivel Moore modell szerint működik a hálózat, ezért a kimenet csak az állapotok megváltozásánál változik.

A kimenetet az alábbi táblázat tartalmazza:

| $x_1$, $x_2$ | 00 | 10 | 11 | 01 | 00 | 01 |
|---|---|---|---|---|---|---|
| Y ($Y_0$ = A) | A | A | C → D → D | D | C → C | B → B |
| Z | 0 | 0 | 0 → 1 → 0 | 0 | 0 → 1 | 1 → 1 |

Az 11 átmenetnél megjelenő változás 1-es hazárdot jelent. A többi átmenet nem zavaró, hiszen a kimeneten nem okoz változást.

MP: Nézzük az előző példát, de most szinkron hálózatként értelmeve:

| Szinkron@Moore | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 | B, 0 | C, 0 | A, 0 |
| B | A, 1 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 1 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

Szinkron működést feltételezve nem kellett semmilyen megkötést tennünk.

Határozzuk meg a működést a következő módon:
Mindig az órajel felfutó élénél változzanak a szekunder változók, mert a mintavevő ekkor mutatja meg, mi az új érték.
A bemeneti kombinációk az órajel lefutó élénél változzanak.
A feltételeink pont a data lock out működésnek felelnek meg.

A kimeneti érték csak akkor olvasható le, amikor már megtörtént az állapotváltozás.

Vizsgáljuk meg a következő bemenetsorozatot a kimenet szempontjából: $x_1$, $x_2$ : 00, 10, 11, 01, 00, 01

| Szinkron@Moore | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | START A, 0 | B, 0 | C, 0 | A, 0 |
| B | A, 1 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 1 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

A táblázatban a piros nyilakat nyomonkövetve láthatjuk a bemeneteket és az állapotokat.
Figyeljünk, ugyanis egy állapotra itt már többször is ugrunk, ami a táblázatot átláthatatlanabbá teszi.
Meghatározott állapotból indulunk, megérkezik a bemenet, érzékeljük, adott állapotban a megérkezett bemenethez tartozó cellára ugrunk, majd az órajelimpulzus következő változásánál állapotot váltunk.

Mivel Moore modell szerint működik a hálózat, ezért a kimenet csak az állapotok megváltozásánál változik.

A kimenetet az alábbi táblázat tartalmazza:

| $x_1$, $x_2$ | 00 | 10 | 11 | 01 | 00 | 01 |
|---|---|---|---|---|---|---|
| Y ( $Y_0$ = A) | A | A | C | B | A | B |
| Z | 0 | 0 | 1 | 1 | 0 | 1 |

Létezik egy alternatív logika is az állapotváltásokra, amely szerint úgy értelmezhetjük az állapottáblát, hogy a következő állapot már az új bemenettel találkozik a visszacsatolódás után, és ez szerint határozhatjuk meg a következő oszlopot és sort.
Például az A állapotban 11-es bemenetnél, amikor a C állapot visszacsatolódik, akkor ő már a 01-es bemenettel találkozik, így a következő állapot a 12-es oszlopban és a a C sorban lesz.

MP: Módosítsunk a kimeneteken és vizsgáljuk meg a működést Mealy modellt feltételezve, először aszinkron módon:

| Aszinkron@Mealy | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 | B, 0 | C, 1 | A, 0 |
| B | A, 0 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 0 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

Lehet-e az állapottáblájával adott hálózat aszinkron?

- Stabil állapotok mindenhol vannak, ezeket zölddel jelöltük.
- Az összes szomszédos bemeneti változásra a hálózat stabil állapotba jut minden állapotból.
  - A táblázatban egy példát jelöltünk: *A* állapotból indulunk és a 00 → 01 állapotátmenetet vizsgáljuk. A 00 → 01 bemeneti változásra a hálózat A állapotból indul, B állapotba kerül, majd ott stabilizálódik.

Lehet aszinkron, de vajon normál hálózattal van dolgunk?
Azt láthattuk, hogy *A* állapotból indulva a 00 → 01 bemeneti változásra egyetlen nem stabil köztes állapot van. Azonban *A* állapotból indulva az 10 → 11 bemenetváltozásra először a hálózat C állapotba ugrik egy instabil állapotra, majd innen ugrik D állapotba. A táblázatban piros színnel jelöltük. Tehát a hálózat nem normál.

Milyen modell szerint működik a hálózat?
A hálózat Moore modell szerint működik, mivel a kimenet nem függ a bemenettől, csak az aktuális állapottól.

Vizsgáljuk meg a következő bemenetsorozatot a kimenet szempontjából: $x_1$, $x_2$ : 00, 10, 11, 01, 00, 01

| Aszinkron@Mealy | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 START | B, 0 | C, 1 | A, 0 |
| B | A, 0 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 0 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

A táblázatban a piros nyilakat nyomonkövetve láthatjuk a bemeneteket és az állapotokat.

Mivel Mealy modell szerint működik a hálózat, ezért a kimenet a bemenet megváltozásakor rögtön megváltozik, majd az állapot megváltozásakor is változik.

A kimenetet az alábbi táblázat tartalmazza:

| $x_1$, $x_2$ | 00 | 10 | 11 | 01 | 00 | 01 | 01 |
|---|---|---|---|---|---|---|---|
| Y ($Y_0$ = A) | A | A | C → D → D | D | C → C | B → B | |
| Z | 0 | 0 | 1 → 0 → 0 | 0 | 0 → 1 | 1 → 1 | |

MP: A módosított kimenetekkel vizsgáljuk meg szinkron módon is, Mealy modellt feltételezve:

| Szinkron@Mealy | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 | B, 0 | C, 1 | A, 0 |
| B | A, 0 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 0 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

Szinkron működést feltételezve nem kellett semmilyen megkötést tennünk.

Határozzuk meg a működést a következő módon:
Mindig az órajel felfutó élénél változzanak a szekunder változók, mert a mintavevő ekkor mutatja meg, mi az új érték.
A bemeneti kombinációk az órajel lefutó élénél változzanak.
A feltételeink pont a data lock out működésnek felelnek meg.

A kimenet értéke ebben az esetben akkor változik, amikor vagy a szekunder változók vagy a bemenetek változnak.
Így egy bemeneti kombinációhoz, és egy szekunder változóhoz is két érték tartozik.
Hogy döntjük el, hogy melyik az igazi?
Ha Mealy modell szerint tervezünk, akkor a követőhálózatnak meg kell tudnia különböztetni a két különböző kimeneti kombinációt, és el kell dobnia a parazita kimenetet.

Vizsgáljuk meg a következő bemenetsorozatot a kimenet szempontjából: $x_1$, $x_2$ : 00, 10, 11, 01, 00, 01

| Szinkron@Moore | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 START | B, 0 | C, 0 | A, 0 |
| B | A, 1 | B, 1 | D, 1 | C, 1 |
| C | C, 1 | B, 1 | D, 1 | C, 1 |
| D | C, 0 | D, 0 | D, 0 | A, 0 |

A táblázatban a piros nyilakat nyomonkövetve láthatjuk a bemeneteket és az állapotokat.
Figyeljünk, ugyanis egy állapotra itt már többször is ugrunk, ami a táblázatot átláthatatlanabbá teszi.
Meghatározott állapotból indulunk, megérkezik a bemenet, érzékeljük, adott állapotban a megérkezett bemenethez tartozó cellára ugrunk, majd az órajelimpulzus következő változásánál állapotot váltunk.

A kimenetet az alábbi táblázat tartalmazza:

| $x_1$, $x_2$ | 00 | 10 | 11 | 01 | 00 | 01 | 01 |
|---|---|---|---|---|---|---|---|
| Y( $Y_0$ = A) | A | A | C | B | A | B | B |
| Z | 0 → 0 | 0 → 0 | **1** → 0 | 1 → 1 | 0 → 0 | **0** → 1 | 1 → 1 |

Az eredményt azonnal megtudjuk, de meg kell jegyeznünk, mert utána kapunk egy fals értéket is!

A mintapéldában szereplő bemeneti kombinációk nem mutatnak meg minden lehetőséget.
A táblázatos formában megadott kimenetek nem tartalmazzák azt a fontos információt, hogy az órajelhez képest mikor változik a kimenet, és mikor változik az állapot. Ahogy már fentebb említettük, ezt a részt könnyedén be lehet gyakorolni, amivel vizsgán biztos pontokhoz juthatunk!

Sajnos szövegszerkesztőben nem egyszerű grafikonokat rajzolni. Ez az oka, hogy ez a része kimarad a tananyagnak.

## Elemi sorrendi hálózatok (flip-flop):

Sorrendi hálózatok megvalósításához kész elemeket használhatunk. Léteznek úgy nevezett elemi sorrendi hálózatok, melyektől az alábbi pár tulajdonságot várjuk el:

- Két állapot.
- Maximum két bemenet.
- Egyetlen kimenet.
- Moore modell szerinti működés.
- Kimenetük és állapotuk megegyezik.

Az alábbi tulajdonságnak nagyjából 30 különféle flip-flop felel meg, azonban mi csak 5 darabbal foglalkozunk, ami viszont elég minden feladatunk megvalósításához.

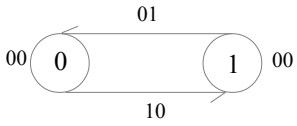
### Set-reset flip-flop (SR flip-flop)

Az S-R flipflopnak két bemenete van, a Set és a Reset. Egy darab kimenete van, melynek értéke az aktuális állapot.

A flip-flop működése a következő:
A Set bemeneten érkező 1-es a flip-flopot 1-es állapotba billenti.
A Reset bemeneten érkező 1-es a flip-flopot 0-s állapotba billenti.
Az 11 bemenet nem definiált, gondoskodnunk kell arról, hogy ne kapjon ilyen bemenetet a flip-flop!

Lássuk a működés alapján az állapottáblát és az állapotgráfot:

| y | SR | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | - | 1 |
| 1 | | 1 | 0 | - | 1 |




A 00 bemenet az aktuális állapotban tartja a flip-flopot.

A kimenetet nem tüntettük fel, hiszen az mindig megegyezik az aktuális állapottal.
Gyakoroljuk az állapottábla-állapotgráf átírást, vizsgán ugyanis könnyedén előfordulhat ilyen feladat!

Mivel nincs oszcilláció, így az S-R flip-flop aszinkron és szinkron módon is létezik.

Tervezzük meg a flip-flopunkat:

Most végre kihasználhatjuk a bináris sorrendek felcserélését, és egyből felrajzolhatjuk a Karnaugh-táblát:

| | | S | |
|---|---|---|---|
| 0 | 0 | - | 1 |
| y 1 | 0 | - | 1 |
| | R | | |

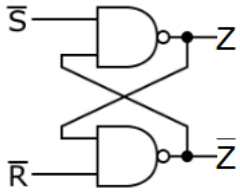
Grafikus minimalizálással a következőt kaptuk:
Y= S + y*/R

Ezt nevezzük a flip-flop **karakterisztikus egyenletének**.

Építsük is meg a flip-flopunkat:

A fent kapott logikai függvényt megvalósíthatjuk egy kombinációs hálózattal, majd ezt a kombinációs hálózatot kell visszacsatolnunk. A Z kimenet a visszacsatolásból kivezethetjük, azonban ha szinkron módon valósítjuk meg a flip-flopot, akkor a buffer utáni állapotot kell kivezetnünk. Ezt a tulajdonságoknál megszabott Z=y miatt tehetjük meg.

Az SR flip-flop egy NAND kapus realizáció esetén a következőképpen néz ki:




/Z kivezetésénél figyelnünk kell arra, hogy a hálózat ne kapjon 11 kombinációt. Z kivezetésénél az 11-es bemeneti kombinációhoz azonban 1-et rendeltünk, így ott nem fordul elő probléma (csak teljesen specifikáltuk a flipflopunk működését).

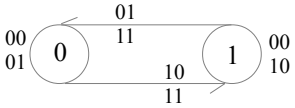
Szinkron hálózat esetén a szinkronizálást jelentő elemeket a visszacsatolásba kell beiktatnunk.

**J-K flip-flop:**

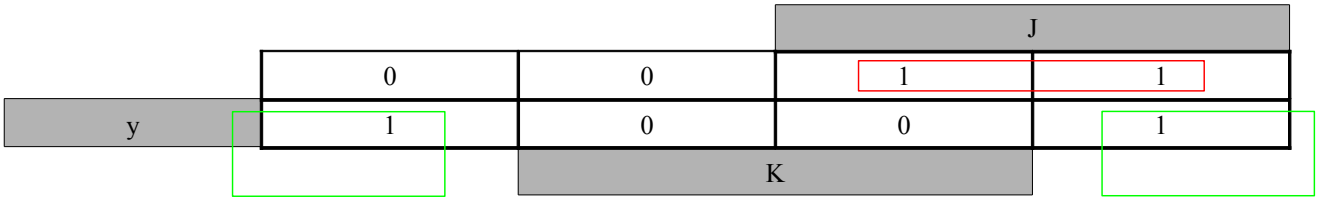
J-K flip-flop annyiban különbözik a S-R-től, hogy az 11 kombinációt megengedjük, és specifikáljuk is.
A flip-flop működése tehát a következő:
A J bemeneten érkező 1-es a flip-flopot 1-es állapotba billenti.
A K bemeneten érkező 1-es a flip-flopot 0-s állapotba billenti.
11 kombináció esetén változtassa meg a mindenkori állapotát.

Állapotábra:

| y | JK | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | 1 | 1 |
| 1 | | 1 | 0 | 0 | 1 |

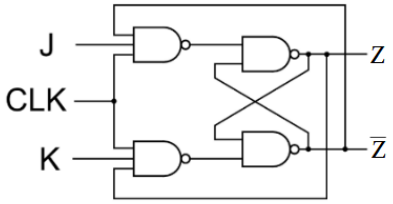



Ezt azonban csak szinkron módon valósíthatjuk meg, mivel az 11 bemeneti kombináció esetén nincs stabil állapot!




Y= J*/y + y*/K

Mivel szinkron módban működik, ezért hazárd nem léphet fel, nem kell hazárdmentes alakot keresnünk.
JK flip-flop egy NAND kapus realizációja:



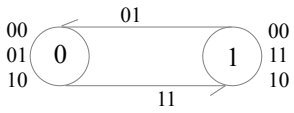
**D-G flip-flop**
(Data-Gate)

A D-G flipflopnak két bemenete van, a Data és a Gate. Egy darab kimenete van, melynek értéke az aktuális állapot.

A flip-flop működése a következő:
Amennyiben a G=1, azaz a gate nyitva van, a következő állapot (=kimenet) a Data értéke.
Amennyiben a G=0, azaz a gate csukva van, a következő állapot az aktuális állapot.

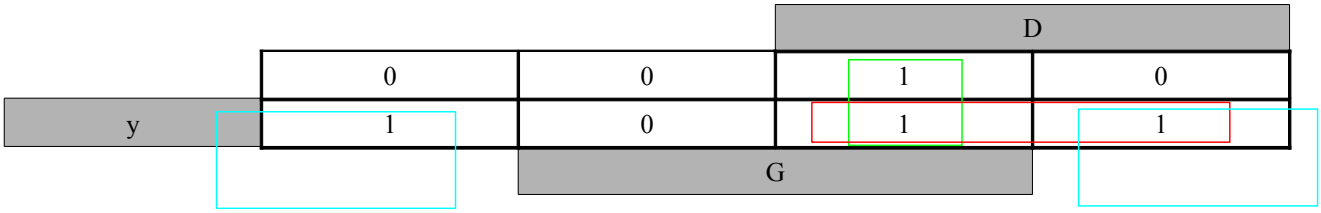
Lássuk a működés alapján az állapottáblát és az állapotgráfot:

| y | DG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | 1 | 0 |
| 1 | | 1 | 0 | 1 | 1 |




Stabil állapot van és nem oszcillál, tehát működhet aszinkron módban. Szinkron módon is működne, de a gyakorlatban csak aszinkron módon valósítják meg. **Latch** néven is hivatkoznak rá (latch=retesz).

A DG flip-flop karakterisztikus egyenletét keressük meg ismét grafikus minimalizálással:




Y= D*G + /G*y + D*y

Aszinkron megvalósításnál (amivel a gyakorlatban találkozunk) hazárdmentes alakot kell felírnunk!

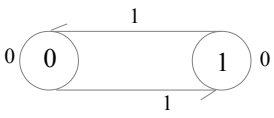
**T flip-flop**
(Trigger)

Egyetlen bemenete és egy kimenete van.

A flip-flop működése a következő:
Amennyiben a T=0, a következő állapot az aktuális állapot.
Amennyiben a T=1, a flip-flop állapotot vált.

Lássuk a működés alapján az állapottáblát és az állapotgráfot:

| y | T | 0 | 1 |
|---|---|---|---|
| 0 | | 0 | 1 |
| 1 | | 1 | 0 |




A karakterisztikus egyenlet: Y= y*/T + /y*T

Mivel a flip-flop T=1 bemenetre oszcillál, így csak szinkron módon létezik.

T flip-flop JK flip-floppal könnyedén megvalósítható, amennyiben a J és K bemeneteket összekötjük.

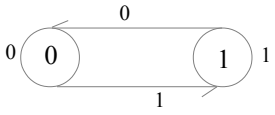
**D flip-flop**
(Data)

A D flip-flopnak 1 bemenete és 1 kimenete van.
A működése pedig a következő:
Az órajelre a D értéke megjelenik a kimeneten.

Bár nem oszcillál, a működéséből következően csak és kizárólag szinkron módon létezik!
Az aszonkron D flip-flopot vezetéknek nevezhetnénk.

Lássuk a működés alapján az állapottáblát és az állapotgráfot:

| y | D | 0 | 1 |
|---|---|---|---|
| 0 | | 0 | 1 |
| 1 | | 0 | 1 |




$Y=f_D(D,y)$ karakterisztkus egyenletet keressük:

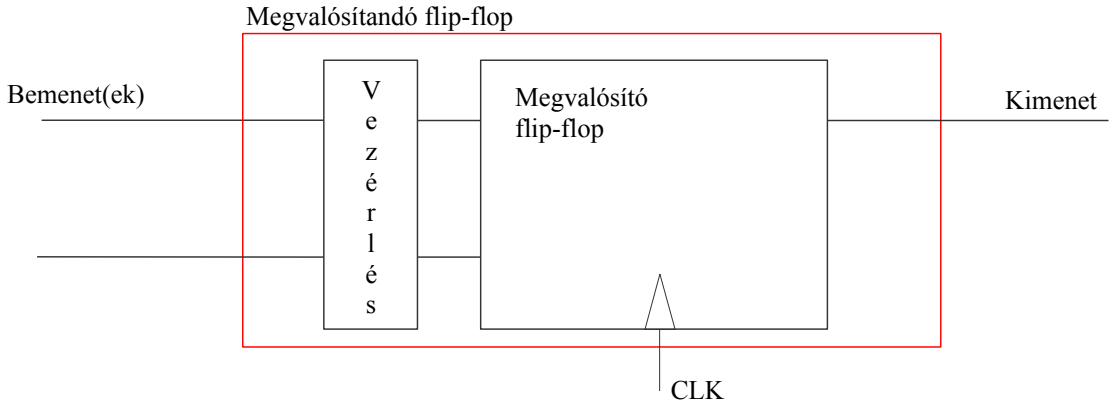
| | D | |
|---|---|---|
| | 0 | 1 |
| y | 0 | 1 |

Y=D

A D flip-flop tehát maga a szinkron hálózatok esetén feltételezett visszacsatoló elem!

**Flip-flopok megvalósítása adott flip-flop és vezérlőhálózat segítségvel:**

Flip-flopok egymással is megvalósíthatóak, amit a következő módon tehetünk meg:




A problémát a vezérlőhálózat megtervezése jelenti a magvílósítandó és a megvalósító flip-flopok ismeretében.
Ez a hálózat különböző bonyolultságú lehet, például T flip-flopot J-K flipfloppal úgy valósíthatunk meg, hogy a J és K bemeneteket összekötjük.
Az órajel nem minden esetben szükséges.
A visszacsatolás vagy a vezérlésnél, vagy a megvalósító flip-flopon belül jelenik meg.

MP: Valósítsunk meg J-K flip-flopot T flip-flop felhasználásval:

A feladat megvalósításához szükséges, hogy ismerjük a felhasználni kívánt flip-flopok állapottábláját.

J-K flip-flopot szeretnénk megvalósítani:

| y | JK | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | 1 | 1 |
| 1 | | 1 | 0 | 0 | 1 |

T flip-flop áll a rendelkezésünkre:

| y | T | 0 | 1 |
|---|---|---|---|
| 0 | | 0 | 1 |
| 1 | | 1 | 0 |

A vezérlőhálózat elkészítéséhez úgy nevezett **vezérlési táblát** használunk, melyet a későbbiekben, sorrendi hálózatok realizációjánál is használni fogunk.
A vezérlési táblát a következő elveket követve kell megkonstruálnunk:
A külső szemlélőnek J-K flip-flopot kell látnia, aminek 2 bemenete van.
Ezt a két bemenetet kell átalakítanunk 1 bemenetre, mivel a T flip-flopnak csak 1 bemenete van.
Az átalakítást úgy kell véghezvinnünk, hogy a megfelelő bemenetre a T flipflop olyan kimenetet állítson elő, ami megfelel a J-K flip-flop kimenetével az adott bemenethez.

A vezérlési tábla:

| y | JK | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | 1 | 1 |
| 1 | | 0 | 1 | 1 | 0 |

Nézzük meg az első cellába mi kerül:
0 állapotban vagyunk, 00 a bemenet, ebben az esetben a J-K flip-flopunk 0 értéket kell, hogy előállítson a kimenetén.
A T flip-flop kimenete 0 állapotban akkor lesz 0, ha a bemenetére 0 érkezik.
Abban az esetben, ha egy két bemenetű flipflopot vezérelnénk, akkor dontcare értékek is előfordulhatnának.
Például ha a fentebbi példában T flip-flop helyett DG flip-flopot használnánk, akkor 0 állapotban a 00, 01 és az 10 bemenet is a megfelelő kimenetet eredményezné.

A példa kedvéért nézzük meg 00 bemenetre az 1-es állapotot is:
1-es állapotban vagyunk, 00 a bemenet, ebben az esetben a J-K flip-flopunk 1 értéket kell, hogy előállítson a kimenetén. A T flip-flop kimenete 1 állapotban akkor lesz 1, ha a bemenetére 0 érkezik.

Ezen logikát követve kitöltjük a vezérlési táblát, majd felírjuk a vezérlőfüggvényt.

Ismét kihasználjuk a felcserélt bináris sorrendeket és egyből Karnaugh-táblába írjuk az értékeket:

| | | J | |
|---|---|---|---|
| | 0 | 0 | 1 | 1 |
| y | 0 | 1 | 1 | 0 |
| | | K | |

Mivel T flip-flop csak szinkron módon létezik, ezért a hálózat biztosan szinkron lesz, így nem kell hazárdmentesítés.
Grafikus minimalizálással a következő lpogikai függvény adódik a vezérlőhálózatra:
$T=f_{T \rightarrow JK}$ (J, K, y)= Ky + /yJ

## Szinkron sorrendi hálózat előzetes állapottáblájának felvétele:

MP:
4 bites prím számokat fogadunk a bemeneten 4 ütemben. A kimeneten jelenjen meg, hogy lehet-e még prím számunk, illetve a 4 ütem végén az jelenjen meg, hogy a szám prím-e. A feladat meghatározása szabhatja meg, hogy Moore vagy Mealy modellt kell használnunk. Ha a kimenetet a bemenet után rögtön meg kell mutatnunk, akkor csak Mealy modellt használhatunk, ugyanis Moore modellnél csak az állapot megváltozásakor változhat a kimenet. Ne feledjük, hogy Mealy modell szerint tervezve a követő hálózatnak ki kell szűrnia a hibáss kimeneteket.
A hálózatnak 1 bemenete és 1 kimenete van, mely 1-es ha a szám még lehet prím, vagy ha prím.
Az MSB bit jön először.

Nézzük melyek a prím számok 4 biten. A táblázat csak segítség annak eldöntésében, hogy az adott szám lehet-e prím.

| | I | II | III | IV |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 |
| 13 | 1 | 1 | 0 | 1 |

Nézzünk egy példát a bemenetre és a hozzá tartozó kimenetre, hogy értsük, hogy is működik a hálózat:
X: 0010

0: Első ütemben a kimenet mindig 1-es, hiszen vagy 0 vagy 1 jön, tudunk olyan prímet, ami így kezdődik.
00: A kimenet még mindig 1-es, hiszen van olyan prím, ami úgy kezdődik, hogy 00.
001: A kimenet még mindig 1-es, a 2-es és a 3-as szám is megfelel a kritériumnak.
0010: A kimenet itt is 1-es, hiszen a bemeneten a 2-es szám jött, ami prím.

**Vegyük fel az állapottáblát:**
Kezdjük azzal, hogy felveszünk egy START állapotot. Az ABC betűivel jelöljük az állapotokat, így a start állapot legyen az A állapot.

Első ütem
Az első ütemben vagy 1-es, vagy 0 jön.
Ha 0 jön: van-e olyan állapotunk, ami azt kódolja, hogy 0 a bemenet? Nincs, tehát fel kell vennünk. Ez lesz a B állapot.
Tehát A állapotból 0 bemenetre B állapotba ugrunk, és a kimenet 1, hiszen a számunk még lehet prím.
Ha 1 jön: van-e olyan állapotunk, ami azt kódolja, hogy 1 a bemenet? Nincs, tehát fel kell vennünk. Ez lesz a C állapot.
Tehát A állapotból 1 bemenetre C állapotba ugrunk, és a kimenet 1, hiszen a számunk még lehet prím.

Második ütem:
Ha B állapotban voltunk, azaz eddig 0 jött, és most 0 jön: olyan állapot kell, ami a 00-t kódolja. Ilyen még nincs, ez lesz a D állapot.
01-et fogja kódolni az E állapot.
10 lesz az F állapot.
11 lesz a G állapot.
Határozzuk meg, hogy melyik állapotból melyik bemenetre milyen új állapotba kerülünk, és határozzuk meg a kimeneteket is. Töltsük ki az állapottábla megfelelő celláit.
A kimenetek meghatározásakor döntsük el, hogy Mealy vagy Moore modell szerint tervezünk.

Harmadik ütem:
Itt nem kell minden lehetőséghez új állapotot felvennünk, hiszen azokat a számokat, amik már nem elhetnek prímek, nem kell megkülönböztetnünk. A prímeket tartalmazó táblázat alapján a következőket kell külön állapotokkal leírnunk: 000, 001, 010, 011, 101, 110 és minden más egy *nem prím* állapotba kerül.

Negyedik ütem:
Itt már csak az a kérdés, hogy a számunk prím-e vagy sem?
Arra figyeljünk, hogy az N állapotot nem használhatjuk, mivel az még csak 3 ütemet kódol.

Mealy modell szerint az állapottábla:

| y | X | 0 | 1 |
|---|---|---|---|
| A (START) | | B, 1 | C, 1 |
| B (0) | | D, 1 | E, 1 |
| C (1) | | F, 1 | G, 1 |
| D (00) | | H, 1 | I, 1 |
| E (01) | | J, 1 | K, 1 |
| F (10) | | N, 0 | L, 1 |
| G (11) | | M, 1 | N, 0 |
| H (000) | | O, 0 | P, 1 |
| I (001) | | P, 1 | P, 1 |
| J (010) | | O, 0 | P, 1 |
| K (011) | | O, 0 | P, 1 |
| L (101) | | O, 0 | P, 1 |
| M (110) | | O, 0 | P, 1 |
| N (nem prím) | | O, 0 | O, 0 |
| O (nem prím) | | B, 1 | C, 1 |
| P (prím) | | B, 1 | C, 1 |

Moore modell szerint:

| y | X | 0 | 1 |
|---|---|---|---|
| A (START) | | B, 1 | C, 1 |
| B (0) | | D, 1 | E, 1 |
| C (1) | | F, 1 | G, 1 |
| D (00) | | H, 1 | I, 1 |
| E (01) | | J, 1 | K, 1 |
| F (10) | | N, 1 | L, 1 |
| G (11) | | M, 1 | N, 1 |
| H (000) | | O, 1 | P, 1 |
| I (001) | | P, 1 | P, 1 |
| J (010) | | O, 1 | P, 1 |
| K (011) | | O, 1 | P, 1 |
| L (101) | | O, 1 | P, 1 |
| M (110) | | O, 1 | P, 1 |
| N (nem prím) | | O, 0 | O, 0 |
| O (nem prím) | | B, 0 | C, 0 |
| P (prím) | | B, 1 | C, 1 |

Ha többet gondolkozunk akkor az alábbi eredményt kaphatjuk (Mealy modell szerint):

| y | X | 0 | 1 |
|---|---|---|---|
| A (START) | | B, 1 | C, 1 |
| B (0) | | D, 1 | E, 1 |
| C (1) | | F, 1 | G, 1 |
| D (00) | | H, 1 | I, 1 |
| E (01) | | H, 1 | H, 1 |
| F (10) | | J, 0 | H, 1 |
| G (11) | | H, 1 | J, 0 |
| H | | A, 0 | A, 1 |
| I | | A, 1 | A, 1 |
| J | | A, 0 | A, 0 |

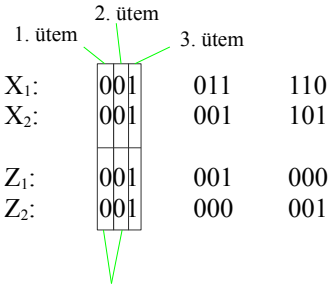
Ahelyett, hogy felveszünk még jó pár változót, gondolkodjunk:

Vegyünk fel H, I, J állapotokat. Vizsgáljuk meg, hogy melyik állapotból hova juthatunk.
H állapot jelentse azt, hogy az utolsó bit 1-es, és előtte nem 001 bemenetek érkeztek.
I állapot jelentse azt, hogy 001 bemenet érkezett, hiszen ekkor mindegy milyen a 4. bemenet.
J állapot jelentse azt, hogy nem prím a számunk.

MP:
Vegyük fel egy két bites szinkron soros komparátor előzetes állapottábláját:
3 bites számokat hasonlítsunk össze, az első két ütemben a kimenetek értéke legyen 00.
A hálózatnak két bemenete és két kimenete van.
11 a kimenet értéke, ha a két szám egyenlő.
10 a kimenet értéke, ha az $X_1$-en érkező szám nagyobb, mint az $X_2$-n érkező.
01 a kimenet értéke, ha az $X_1$-en érkező szám nagyobb, mint az $X_2$-n érkező.

**Első bit legyen az LSB, és az utolsó bit legyen az MSB!**

Nézzünk példákat különböző bemenetekre:




Első két ütemben a kimenet 0.

Az állapottábla felvétele most is egy START állapot felvételével kezdődhet.

A következő állapottáblát fogjuk kapni:

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A (START) | B, 00 | D, 00 | B, 00 | C, 00 |
| B ( 1 bit = ) | E, 00 | G, 00 | E, 00 | F, 00 |
| C (1 bit > ) | F, 00 | G, 00 | F, 00 | F, 00 |
| D (1 bit <) | G, 00 | G, 00 | G, 00 | F, 00 |
| E (2 bit =) | H, 00 | J, 00 | H, 00 | I, 00 |
| F (2 bit >) | I, 00 | J, 00 | I, 00 | I, 00 |
| G (2 bit <) | J, 00 | J, 00 | J, 00 | I, 00 |
| H (végleges =) | B, 11 | D, 11 | B, 11 | C, 11 |
| I (végleges >) | B, 10 | D, 10 | B, 10 | C, 10 |
| J (végleges <) | B, 01 | D, 01 | B, 01 | C, 01 |

Nézzük mit jelent 1-1 állapot:

B állapot azt jelenti, hogy 1 darab bit jött mindkét bemeneten, és ezek egyenlőek.
F állapot azt jelenti, hogy 2 darab bit jött mindkét bemeneten, és az $X_1$ 2 bitje nagyobb, mint az $X_2$ két bitje.
Ezért: B állapotból F állapotba ugrunk akkor, ha B állapotban voltunk, tehát az első bitek megegyeztek, és $X_1$-en nagyobb szám jön, mint $X_2$-n (10 bemenetre).

A kezdő állapotban ebben az esetben egyszer vagyunk, hiszen a H, I, J állapotoknál következő bit érkezésekor rögtön eldöntjük, hogy B, C vagy D állapotba kerül-e a hálózat.

MP:
Vegyük fel egy szinkron sorrendi hálózat állapottábláját, ami a következő módon működik:
3 ütemben fogad 3 bites számokat, és összehasonlítja őket.
A kimenete 1, ha a 3. bit után a két szám egyenlő.
Tegyük fel, hogy tudjuk, hogy hanyadik ütemben tartunk, így csak a 3. ütemben érdekel minket a kimenet.

Moore modell szerint tervezve:

Z=f(y)

A következő állapottáblát kapjuk:

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A (START) | B, - | C, - | B, - | C, - |
| B ( 1 bit = ) | D, - | E, - | D, - | E, - |
| C (1 bit ≠) | E, - | E, - | E, - | E, - |
| D (2 bit =) | F, - | G, - | F, - | G, - |
| E (2 bit ≠) | G, - | G, - | G, - | G, - |
| F (3 bit =) | B, 1 | C, 1 | B, 1 | C, 1 |
| G (3 bit ≠) | C, 0 | C, 0 | C, 0 | C, 0 |

Az állapottábla felvétele ez esetben is egy START állapot felvételével kezdődik, melyet most sem használunk, csak az első alkalommal.

Mealy modell szerint tervezve:

Z=f(X, y)

A következő állapottáblát kapjuk:

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A (START) | B, - | C, - | B, - | C, - |
| B ( 1 bit = ) | D, - | E, - | D, - | E, - |
| C (1 bit ≠ ) | E, - | E, - | E, - | E, - |
| D (2 bit =) | A ,1 | A, 0 | A ,1 | A, 0 |
| E (2 bit ≠) | A, 0 | A, 0 | A, 0 | A, 0 |

Az állapottábla felvétele ez esetben is egy START állapot felvételével kezdődik, melyet most sem használunk, csak az első alkalommal.

Amikor megjönnek a 3. bitek, akkor rögtön döntést tudunk hozni arról, hogy megegyeznek-e. Így 2 állapotot megspórolhatunk amennyiben Mealy modell szerint tervezünk.

Aszinkron sorrendi hálózat előzetes állapottáblájának felvétele:

MP:
Legyen egy 2 bemenetű (X, G) aszinkron hálózatunk, melynek a működése a következő:
Ha G=1, akkor a kimenet a 2. X felfutó élre változzon 1-re. Amint a G 0, a kimenet azonnal 0-ra ugrik.

Előzetes állapottábla: XG

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 0 | B, 0 | - | G, 0 |
| B | A, 0 | B, 0 | C, 0 | - |
| C | - | D, 0 | C, 0 | G, 0 |
| D | A, 0 | D, 0 | E, - | - |
| E | - | F, 1 | E, 1 | G, - |
| F | A, - | F, 1 | E, 1 | |
| G | A, 0 | - | H, 0 | G, 0 |
| H | - | B, 0 | H, 0 | |

**NEM engedélyezünk dupla bemenetváltozást, hiszen aszinkron módon ilyen nem lehet!**
Ezért az A,0 állapot után, ami 00 bemenetnél fordul elő, kihúzzuk az 11 bemenetet az A állapotban.

| : Ahol a kimenet megváltozik, ott az érték nem lényeges, hiszen csak minimálisan időben eltolja a kimenetváltozást, de ez a hálózat működésének sebességénél jóval kisebb nagyságrend.

Először értelmezzük, hogy mit is jelent számunkra a bemenet:
00: nincs az X-en felfutó él, és nincs engedély a Gate-en.
01: engedélyt kaptunk, de várunk egy felfutó élre.
11: engedély van, és az X-en felfutóél van.
10: engedély van, és az X-en nincs felfutóél.

Mit is jelent a felfutóél számunkra és mikor is jön:
Ha az X-en a bemenet eddig 0 volt, akkor a következő váltásra az X-en felfutóél jön.
Ha az X-en a bemenet eddig 1 volt, akkor a következő váltásra az X-en lefutó éljön, amit nekünk nem kell számolnunk, de a felfutóélek számlálásánál fontos tudnunk, hogy az X-en az előző változás mi volt (ennek tudatában kell felvennünk az állapottáblát).

A állapot: nincs engedély, 0 felfutó él jött, és felfutó él lesz a következő az X-en.
B állapot: van engedély, 0 felfutó él jött, és felfutó él lesz a következő az X-en.
C állapot: van engedélyezés, 1 felfutó él jött, és lefutó élre várunk az X-en.
D állapot: van engedély, 1 felfutó él jött, és felfutó élre várunk az X-en.
E állapot: van engedély, legalább 2 felfutóél jött, tehát a kimenet 1 lesz, amíg G=1.
F állapot: van engedély, legalább 2 felfutóél jött, tehát a kimenet 1 lesz, amíg G=1.
-Ez az állapot csak azért szükséges, hogy az állapottábla minden sorában pontosan 1 stabil állapot legyen!
G állapot: nincs engedély, 0 felfutó él jött (nincs engedély, nem is számoljuk), és egy lefutó élre várunk X-en.
H állapot: van engedélyünk, 0 felfutó él jött, és lefutó élre várunk.

**Előzetes állapottáblánál egy sorban egy stabil állapotot engedünk!**
Ha a feladat előzetes aszinkron állapottáblát kér, akkor azért is be kell vennünk új állapotokat, hogy a tábla előzetes állapottábla legyen, még akkor is, ha egyébként nem kéne.
A későbbiekben állapotokat össze fogunk tudni vonni.

MP:
Vegyük fel egy aszinkron hálózat állapottáláját, melynek két bemenete van, G és X, és két kimenete is van.
A hálózat működése a következő:
A kimenet 00, ha a G=0.
A kimenet 10, ha a G=1 és X-en páratlan számú változás történik.
A kimenet 11, ha az X-en páros számú bemenetváltozás történik.

Vigyázzunk, mert az előző feladathoz képest megfordult a Gate és az X szerepe!

Előzetes állapottábla: GX

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| A | A, 00 | E, 00 | - | B, 00 |
| B | A, 00 | - | C, -0 | B, 00 |
| C | - | E, -0 | C, 10 | D, 1- |
| D | A, -- | - | C, 1- | D, 11 |
| E | A, 00 | E, 00 | F, 00 | - |
| F | - | E, 00 | F, 1- | G, 10 |
| G | A, -0 | - | H, 1- | G, 10 |
| H | - | E, -- | H, 11 | G, 1- |

A állapot kódolja, hogy nincs engedély. Ekkor X nem érdekel minket.
B állapot kódolja, hogy engedélyünk már van, de X még nem változott egyszer sem.
C állapot kódolja, hogy van engedély, és páros számú lefutó él fog jönni legközelebb.
D állapot kódolja, hogy volt engedély, és ez páros volt, tehát a kimenetnek 11-nek kell lennie.
E állapot kódolja, hogy nincs engedély, és lefutó él lesz G-n. (ekkor mindegy, hogy X hogyan változik.)
F állapot kódolja, hogy van engedély, és lefutó élre várunk X-en.
G állapot kódolja, hogy van engedély, és páros felfutó élre várunk.
H állapot kódolja, hogy van engedély, és páratlan felfutó élre várunk.

Az előzetes állapottáblában továbbra is csak 1 stabil állapot lehet minden sorban!

: Ahol a kimenet megváltozik, ott az érték nem lényeges, hiszen csak minimálisan időben eltolja a kimenetváltozást, de ez a hálózat működésének sebességénél jóval kisebb nagyságrend. A kimenetek közül csak az egyik változik. Oda rakhatunk csak dont-care-t!

## Állapotösszevonások sorrendi hálózatokban:

Mint azt már korábban láttuk, az előzetes állapottáblában gyakran olyan állapotokat veszünk be, amelyek a feladat megoldása szempontjából nem lényegesek, bizonyos szempontból ugyan olyan állapotát kódolják a hálózatnak. Magától értetődik, hogy ezeket az állapotokat, amennyiben bizonyos feltételeket teljesítenek, összevonhatjuk.

Az alábbiakban az összevonsára egy szisztematikus módszert ismerünk meg, mely segítségével az előzetes állapottáblából egy jóval egyszerűbb állapottáblát tudunk létrehozni.

Két állapot összevonható, haminden lehetséges bemenetre ugyan azt a kimenetet szolgáltatják, és minden bemenetre a következő állapotok is összevonhatóak-e.

Nézzünk példákat:

MP:

| y | X | 0 | 1 |
|---|---|---|---|
| | | ... | |
| F | | C, 0 | E, 1 |
| H | | E, 1 | B, 0 |

Jól látható, hogy nem azonos kimenetet adnak az összes bemenetre. 0 bemenet esetén az F állapot 0, míg a H állapot 1 kimenetet ad. Innetől kezdve ezek az állapotok biztosan nem vonhatók össze.

MP:

| y | X | 0 | 1 |
|---|---|---|---|
| | | ... | |
| F | | C, 0 | E, 1 |
| H | | E, 0 | B, 1 |

Itt láthatjuk, hogy minden bemenetre azonos kimenetet adnak. Ezek az állapotok tehát akkor vonhatók össze, ha a C és E állapotok összevonhatók, **ÉS** az E és B állapotok is összevonhatók.

MP:

| y | X | 0 | 1 |
|---|---|---|---|
| | | ... | |
| F | | F, 0 | E, 1 |
| H | | H, 1 | B, 0 |

Itt F és H állapot összevonható, ha F és H állapotok összevonhatók, ami triviálisan teljesül, **ÉS** E és B állapotok is összevonhatók.

MP:

| y | X | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| | | | ... | | |
| F | | H, 0 | F, 1 | A, 1 | B, 1 |
| H | | F, 1 | H, 0 | C, 1 | E, 1 |

Itt F és H állapot összevonható, ha H és F állapot összevonható, ÉS F és H állapot összevonható, ÉS A és C állapotok összevonhatók, **ÉS** B és E állapotok is összevonhatók.

Nézzük a konkrét példánkat és magát a módszert.

MP:
Tekintsük az alábbi hálózatot, és végezzük el az állapotok egyszerűsítését:

| y | X | 0 | 1 |
|---|---|---|---|
| A | | C, 0 | E, 1 |
| B | | E, 1 | B, 0 |
| C | | D, 1 | F, 0 |
| D | | E, 1 | D, 0 |
| E | | D, 1 | E, 0 |
| F | | B, 1 | A, 0 |

Vegyünk fel egy háromszög mátrixot (lépcsős táblát) az alábbi szabály szerint:
Állapotok számánál egyel kevesebb sort és oszlopot veszünk fel, a sorokat fentről lefelé B-től számozzuk, az oszlopokat pedig balról jobbra A-tól számozzuk:

| | | | | | |
|---|---|---|---|---|---|
| B | x | | | | |
| C | x | DE, BF | | | |
| D | x | | DE, DF | | |
| E | x | DE | EF | | |
| F | x | BE, AB | BD, AF | BE, AD | AE, BD |
| | A | B | C | D | E |

Cellánként menjünk végig, és vizsgáljuk az összevonhatóságot:

Minden párra nézzük meg a kimeneteket, ha stimmelnek, és ugyan arra az állapotpárra mutatnak, akkor pipát rakunk (a táblázatban zöld szín jelöltük), mert összevonhatók, egyébként felírjuk, hogy mely állapotoknak kell összevonhatnak lenniük, hogy összevonhassuk a vizsgált két állapotot.
Találhatunk triviális esetet is, ha két ugyan olyan állapotunk van.

-Egyedül az A állapot kimenete 0 abban az esetben, ha a bemenet 0, tehát az A állapot semmivel nem vonható össze.
-B és C állapot akkor vonható össze, ha a D és E valamint a B és F állapotok párosával összevonhatóak.
-B és D állapot triviálisan összevonhatóak, hiszen egyik feltétel az, hogy B és D összevonható legyen, másik feltétel, hogy E és E összevonható legyen, és mindkét feltétel magától értetődően teljesül.
…
A fenti logika alapján töltsük ki a táblázatot.

Ennek a vizsgálatnak az eredménye a fentebbi lépcsős táblázatban található.

A következő teendő, hogy végigvizsgáljuk a lépcsős táblánkat, és megnézzük, hogy mely feltételek nem teljesülnek, és ezek miket ejtenek ki a továbbiakban.

Egyszerűsítés általános lépése: választunk egy esetet, ami biztosan nem összevonható, megnézzük, hogy feltételként hol szerepel, és azt a cellát kihúzzuk (táblázatban ezt egy X-el jelöltük a feltételek mellett). Elég, ha a cellába írt feltételek közül egy nem teljesül. A már megvizsgált és nem teljesülő feltételeket egy újabb X-el jelöljük (ezt a táblázatban piros háttérrel jelöltük), ami jelentése, hogy azt a cellát már megvizsgáltuk, és kihúztuk az összes cellát, ahol feltételként szerepelt.
Ezt addig folytatjuk, amíg mindegyik cella vagy X-et, vagy pipát vagy feltételt tartalmaz.

Ennek a végeredménye a következő oldalon látható.

| | | | | | |
|---|---|---|---|---|---|
| B | X | | | | |
| C | X | DE, BF X | | | |
| D | X | | DE, DF X | | |
| E | X | DE | EF X | | |
| F | X | BE, AB X | BD, AF X | BE, AD X | AE, BD X |
| | A | B | C | D | E |

A második X helyettesítheti a piros hátteret. ~~X~~

Azt kaptuk, hogy BD, BE és DE összevonható, tehát BDE összevonható.
Ezt a megállapítást viszont **csak teljesen specifikált hálózatban** (ahol nincsenek dont care értékek) tehetjük meg!

Teljesen specifikált tábla esetén állapotösszevonásnál **ekvivalenciaosztályokról** beszélünk.

**Nem teljesen specifikált hálózatok** esetén fordítva kell vizsgálnunk az összevonhatóságot, feltételezve, hogy minden állapot összevonható, majd a feltételeket felhasználva kizárni azokat, amelyek mégsem vonhatók össze.
Ezt azért kell megtennünk, mert a különböző dont care értékek miatt előfordulhat, hogy 3 állapotot nem tudunk összevonni a fentebbi példához hasonlóan. Érdemes ezt a módszert megjegyezni, mert minden esetben működik!
Ezt egy példán mutatjuk be:

| y | X | 0 | 1 |
|---|---|---|---|
| ... | | | |
| A | | F, - | F, 0 |
| D | | F, 1 | F, - |
| E | | F, 1 | F, 1 |

Jól látható, hogy A és D állapot összevonható, D és E állapot is összevonható, de ebből nem következik, hogy A és E állapot összevonható, sőt, jól látható, hogy ők semmilyen módon nem vonhatók össze.

Az előző feladatot tehát nem teljesen specifikált hálózat esetén a következő módon oldjuk meg:

| | | | | | |
|---|---|---|---|---|---|
| B | X | | | | |
| C | X | DE, BF X | | | |
| D | X | | DE, DF X | | |
| E | X | DE | EF X | | |
| F | X | BE, AB X | BD, AF X | BE, AD X | AE, BD X |
| | A | B | C | D | E |

Tegyük fel, hogy (ABCDEF) összevonható.

Vizsgáljuk meg sorra az állapotokat, majd a még nem vizsgált állapotok közül az összeset írjuk fel:

Az összes még nem vizsgáltat írjuk fel, nem törődve azzal, hogy a DE állapotot már felírtuk a B mellé!

A: (A)(BCDEF) → A állapot nem vonható össze semmivel, így vegyük ki a blokkból.
B: (A)(BDE)(CDEF) → B állapot BDE állapottal összevaonható, egy külön blokkot alkotnak.
C: (A)(BDE)(C)(DEF) → C állapot nem vonható össze semmivel, A-hoz hasonlóan egy külön blokk.
D: (A)(BDE)(C)(~~DE~~)(EF) → DE állapot összevonható, de már szerepel BDE-ben, így felesleges.
E: (A)(BDE)(C)(~~E~~)(F) → E-t kihúzzuk, hiszen BDE már tartalmazza

Tehát a következő állapotokat kaptuk:
(A)(BDE)(C)(F)

Nem teljesen specifikált esetben **kompatibilitási osztályokról** beszélünk.

Megkaptuk az összevont állapotokat tehát. A következő lépés, hogy a blokkokat elnevezzük új betűkkel, és felírjuk az összevont állapottáblát:

(A)(BDE)(C)(F):

G = (A)
H = (BDE)
J = (C)
K = (F)

Az összevont állapottábla felírása nem bonyolult, az eredeti táblázat állapotait kell helyettesítenünk az új állapotokkal, meghatározni a kimenetet és a következő állapotot.

Az A állapot a G állapotba olvadt. 0 bemenethez az A állapothoz 0 kimenet tartozott, a következő állapot pedig a C volt, ami a J állapotba olvadt, 1 bemenethez a kimenet 1 volt, a következő állapot pedig E, ami a H állapotba olvadt.
Az A állapot tehát a következőre módosult:

| | | |
|---|---|---|
| G | J, 0 | H, 1 |

Egy fokkal érdekesebb a BDE állapot.
BDE állapothoz 0 bemenethez 1 kimenet tartozott, a következő állapotok pedig sorra E, E és D, amik együtt megtalálhatóak a H állapotban. Fontos, hogy itt mindig találnunk kell olyan állapotot, ami együtt tartalmazza ezeket!
1 bemenethet a BDE állapothoz 0 kimenet és a B, D, E következő állapotok tartoznak, amik most is a H állapotban találhatóak.
A BDE állapotok tehát a következőre módosultak:

| | | |
|---|---|---|
| H | H, 1 | H, 0 |

Szemfülesek észrevehették, hogy a hálózatnak nincs túl sok értelme, de a példa bemutatására megfelel.

…

A logikát követve az eredeti tábla, és az állapotösszevonás alapján a következő összevont állapottáblához jutunk:

Eredeti tábla segítségként:

| y | X | 0 | 1 |
|---|---|---|---|
| A | | C, 0 | E, 1 |
| B | | E, 1 | B, 0 |
| C | | D, 1 | F, 0 |
| D | | E, 1 | D, 0 |
| E | | D, 1 | E, 0 |
| F | | B, 1 | A, 0 |

Az egyszerűsített tábla:

| y | X | 0 | 1 |
|---|---|---|---|
| G | | J, 0 | H, 1 |
| H | | H, 1 | H, 0 |
| J | | H, 1 | K, 0 |
| K | | H, 1 | G, 0 |

MP:
Adott a következő állapottábla. Végezzük el az állapotösszevonásokat:

| y | X | 0 | 1 |
|---|---|---|---|
| A | | B, 0 | H, 0 |
| B | | A, 1 | G, 1 |
| C | | -, 1 | F, - |
| D | | D, 1 | -, 0 |
| E | | C, 1 | D, - |
| F | | A, - | C, 1 |
| G | | -, - | B, - |
| H | | G, 1 | E, - |

Az állapottábla alapján a következő lépcsős tábla adódik:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | X | | | | | | |
| C | X | GF | | | | | |
| D | X | X | | | | | |
| E | X | AC, GD | FD | CD | | | |
| F | X | GC | | X | CA, DC | | |
| G | BH | | FB | | DB | BC | |
| H | X | AG, EG | EF | DG | CG, DE | AG, CE | EB |
| | A | B | C | D | E | F | G |

A lépcsős táblán elvégezve az egyszerűsítést, a következő adódik:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | X | | | | | | |
| C | X | GF | | | | | |
| D | X | X | | | | | |
| E | X | AC, GD X | FD X | CD | | | |
| F | X | GC | | X | CA, DC X | | |
| G | BH | | FB | | DB X | BC | |
| H | X | AG, EG X | EF X | DG | CG, DE | AG, CE X | EB X |
| | A | B | C | D | E | F | G |

Jól látható, hogy az állapottábla nem teljesen specifikált, mivel tartalmaz dont care értékeket.
Ebben az esetben kompatibiltási osztályokat hozhatunk csak létre, amihez a második módszer fogjuk alkalmazni.
Ez a következő oldalon látható.

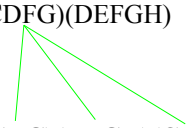
Tegyük fel, hogy mindegyik összevonható, és vizsgáljuk visszafele:

(ABCDEFGH)

A: (A)(BCDEFGH) → Az A állapot most sem vonható össze semmivel.

B: (A)(BCFG)(CDEFGH) → B állapot összevonható a CFG-vel.

C: (A)(BCFG)(CDFG)(DEFGH) → C állapot összevonható CDFG-vel, tehát a BCFG csoport maradhat, és fel kell vennünk egy CDFG csoportot is mellé.



D: (A)(BCFG)**(CDG)(DEGH)**~~**(CFG)**~~(EFGH) → C kapcsolatait már ellenőriztük, ha ABC sorrendben haladunk, mindig csak a hátrébb lévő állapotokat kell ellenőriznünk. De D nem vonható össze velük! CFG már szerepel BCFG-ben.

Nézzük meg itt alaposan, hogy hogyan esik szét egy csoport:
Tehát a kiindulási alap a CDFG csoport, és a D-t ellenőrizzük. Azt nem kell ellenőriznünk, hogy C és D összevonható-e, C-t már ellenőriztük. Csak azt kell ellenőriznünk, hogy D, F és G összevonható-e.
A D állapot E, G és H-val vonható össze, tehát F-el nem.
Így a CDFG csoport 3 részre fog szakadni:

1. Dobjuk ki D-t, mivel nem vonható össze F-el: CFG
2. Mivel előbb kidobtuk D-t, így ő egy új csaoportat is alkot azokkal az állapotokkal, amelyekkel összevonható: DEGH
3. Tartsuk meg D-t, ekkor F-et kell kidobnunk: CDG

Azok a csoportok, amikben az éppen vizsgált állapot nem szerepel, változatlanul leírandók.
Az utolsó csoport mindig csökken az éppen vizsgált állapottal.

E: (A)(BCFG)(CDG)**(DEH)(DGH)**~~**(EH)**~~(FGH) → A DEGH az előbbiek szerint ismét szétszakad az E állapot miatt.

F: (A)(BCFG)(CDG)(DEH)(DGH)~~(FG)~~(GH) → F állapot nem szerepel egy blokkban sem, de FG összevonható, így megjelenik egy újabb csoport. Azonban már szerepel egy másik csoportban, így kihúzzuk.

G: (A)(BCFG)(CDG)(DEH)~~(DG)(DH)(G)(H)~~ → Megjelenik újabb csoport, de most is húzzuk ki a már másik csoportban szereplő csoportokat.

Az állapotösszevonások után a következőket kaptuk:
(A)(BCFG)(CDG)(DEH)

Új állapotok:

| | |
|---|---|
| A: | (A) |
| B: | (BCFG) |
| C: | (CDG) |
| D: | (DEH) |

Nevezzük el őket új állapotoknak. Jelen példában A, B, C és D betűkkel neveztük el őket, míg korábban G, H, J, K-nak, azonban ennek semmi jelentőssége nincs itt. Legyünk következetesek, és ne kavarodjunk bele!

Jól látható, hogy átfedések vannak az új állapotok közt. Egyáltalán nem biztos, hogy nekünk az összeset fel kell használnunk. Vegyünk fel tehát egy lefedési táblát, ami nagyon hasonlít arra, amit a prímimplikánsok megtalálására használtunk.
A táblában azt jelöljük, hogy a régi állapotok közül melyeket fedik le az új állapotok, és akár a segédfüggvényt is felírhatjuk úgy, ahogy azt korábban láttuk.

Vegyük fel a lefedési táblát:

| | | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|---|
| A | A | x | | | | | | | |
| BCFG | B | | x | x | | | x | x | |
| CDG | C | | | x | x | | | x | |
| DEH | D | | | | x | x | | | x |

Láthatjuk, hogy C nem biztos, hogy kell, de előfordulhat, hogy annak ellenére, hogy minden állapotot lefedtünk, mégiscsak be kell vennünk. Ehhez az úgynevezett **zártságot** kell megvizsgálnunk.
Amikor állapotokat összevonunk, az adott bemeneteket tartalmazó következő állapotoknak mindig egy új állapotban kell lenniük. Ha nem találunk olyan új állapotot, ami ezeket együtt tartalmazza, akkor valahol elrontottuk a feladatot!

A zártság ellenőrzéséhez tegyük fel, hogy elég az A, B és D állapot, és próbáljuk meg felvenni az állapottáblát:

| y | X | 0 | 1 |
|---|---|---|---|
| A | | B, 0 | D, 0 |
| B | | A, 1 | B, 1 |
| D | | C, 1 | D ,0 |

De az új D állapotban a DEH állapotokat vontuk össze, melyek azonban következő állapotként a DCG állapotokra mutatnak amennyiben 0 a bemenet, amik pont az új C állapotban találhatóak meg együtt. Az új D állapot olyan összefüggő állapotra mutat tehát, amit nem vettünk be eddig, most tehát be kell vennünk.

| y | X | 0 | 1 |
|---|---|---|---|
| A | | B, 0 | D, 0 |
| B | | A, 1 | B, 1 |
| C | | C/D, 1 | B, 0 |
| D | | C, 1 | D ,0 |

Erre a bemenetre az eredeti CDG állapotokhoz (amikből az új C állapot lett) a -D- következő állapotok tartoznak (lényegében csak a D állapot a megkötés), a D állapot viszont az új C és az új D állapotban is megtalálható, tehát itt eldönthetjük, hogy hová akarunk ugrani, így mindkettőt felírjuk! Ennek az állapotkódolásnál és a kódolt állapottábla felvételénél lesz jelentőssége, mivel egy dont care értéket vehetünk majd fel.

Az állapotösszevonás ezen módszere minden sorrendi hálózatra remekül működik.
A kompatibilitási osztályokhoz tartozó módszert tanuljuk meg és értsük meg jól, ugyanos az minden esetben alkalmazható, és a valósághoz közelebb áll.

A későbbiekben részletesen tárgyalva lesznek a HT partíciók, és szisztematikus módszert is mutatunk majd, amihez szintén a lépcsős tábla lesz segítségünkre. Jól jegyezzük meg a különbségeket, ugyanis tapasztalataim azt mutatják, hogy a diákok hajlamosak összekeverni ezt a két dolgot. A most bemutatott módszer az állapotok összevonására szolgál, a HT partíciók viszont szinkron hálózatok állapotkódolására valók!

## Aszinkron hálózat állapotainak kódválasztása:

Már szó esett a szekunder változókról, és jelöltük is az állapotokat bináris kóddal, de aztán átváltottunk az ABC betűire. Eljött az idő, hogy kapcsolatot teremtsünk a kettő közt.
Az állapotokat valójában bináris kódokkal fogjuk leírni, és a feladatunk, hogy megtaláljuk ezeket a kódokat úgy, hogy az optimális legyen számunkra.
A következő elvnek kell megfelelnünk:
Az állapotok kódját úgy kell megválasztanunk, hogy csak szomszédos változások legyenek megengedve, különben versenyhelyzetek alakulnak ki, melyekről később bővebben beszélünk.

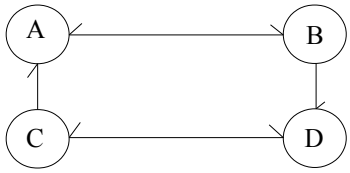
Az alábbiakban intuitív módszereket mutatunk aszinkron hálózatok állapotainak kódválasztására.

MP:
Állapottáblájával adott a következő hálózat. Kódoljuk az állapotokat úgy, hogy minden állapotváltozásnál legfeljebb 1 szekunder változó változzon.
**Az összes nem stabil állapotot is végig kell követnünk nem normál hálózat esetében!**

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | A, 0 | B, 0 |
| B | | A, 0 | D, 0 | D, 0 | B, 0 |
| C | | D, 1 | A, 0 | A, 0 | C, 1 |
| D | | D, 1 | D, 1 | D, 1 | C, 1 |

A tábla alapján kiolvasható, hogy melyek a szomszédos állapotok, ezeket állapotgráffal is megadhatjuk:

A állapotból mehetünk B állapotba.
B állapotból D állapotba, vagy A állapotba mehetünk.
C állapotból mehetünk A-ba, és D-be.
D állapotból C állapotba mehetünk.




A kódolásban újra a Karnaugh-tábla lesz segítségünkre, hiszen azt a táblát pont úgy szerkesztettük, hogy a szomszédos cellák bináris kódjai szomszédosak legyenek. Az alábbi feladat állapotgráfján remekül látszik, hogy nagyon egyszerűen kódolható, hiszen nincsen egyetlen egy olyan átmenet sem, ami megakadályozná, hogy egy az egyben beírjuk a Karnaugh-táblába az állapotokat ugyan olyan alakban, ahogy a gráfon látjuk őket.
Lényegében keresnünk kell egy rácsos elrendezést az állapotgrfában úgy, hogy nincsenek keresztbe élek, mert ekkor a Karnaugh táblába azt egyszerűen bemásolhatjuk és leolvashatjuk az állapotkódokat, mint ebben az esetben.

Karnaugh-tábla:

| | | $y_3$ | |
|---|---|---|---|
| A | B | | |
| $y_1$ C | D | | |
| | $y_2$ | | |

A Karnaugh tábla alapján a következő kódok adódnak:
A=00 B=01 C=10 D=11

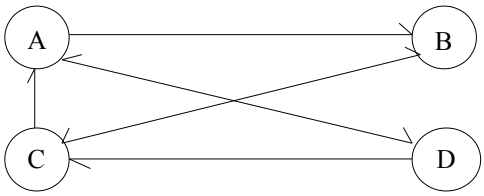
Ebben az esetben az állapotválasztás könnyű volt.

Ebből elkészíthetjük a kódolt állapottáblát, ahova az állapotok helyett már az állapotokhoz tartozó kódokat írjuk. Figyeljünk arra, hogy a kódok olyan sorrendben legyenek, hogy a karnaugh táblákat könnyedén fel tudjuk venni a megvalósításhoz utána!

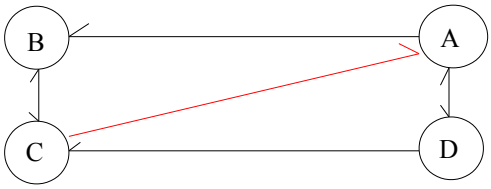
MP:
Állapottáblájával adott a következő hálózat. Kódoljuk az állapotokat úgy, hogy minden állapotváltozásnál legfeljebb 1 szekunder változó változzon.
**Az összes nem stabil állapotot is végig kell követnünk nem normál hálózat esetében!**

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | D, 0 | B, 0 |
| B | | B, 1 | B, 1 | C, 1 | B, 1 |
| C | | B, 0 | A, 0 | C, 0 | C, 0 |
| D | | A, 1 | A, 1 | D, 1 | C, 1 |

A állapotból B-be, és D-be juthatunk.
B állapotból C-be juthatunk.
C állapotból B állapotba és A állapotba juthatunk.
D állapotból A állapotba és C állapotba juthatunk.




Jól látható, hogy sehogy sem találhatunk megfelelő kódokat.
Rajzoljuk át picit a gráfot, hogy jobban látszódjon, hogy a problémát az A-C átmenet jelenti.
Keressük ezt meg a táblázatban.




Vegyük észre, hogy módosíthatjuk könnyedén a táblázatot. Ugyan oda jutunk, de kihasználhatjuk a D állapotban 01 bemenetnél levő állapotot. C állapotból ne egyből A állapotba ugorjunk, hanem előbb D állapotba, és csak onnan A állapotba. Így a problémás átmenetet kiküszöböltük, és a hálózaton is csak olyan módosítást hajtottunk végre, ami nem befolyásolja annak a működését, hiszen csak egyel több instabil állapot lett egy állapotváltásnál.

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | D, 0 | B, 0 |
| B | | B, 1 | B, 1 | C, 1 | B, 1 |
| C | | B, 0 | D, 0 | C, 0 | C, 0 |
| D | | A, 1 | A, 1 | D, 1 | C, 1 |

Így egy instabil állapotot módosítva megszüntettük a számunkra kellemetlen átmenetet.

DE: megjelent a kimeneten egy 1-es. Ez azonban eredetileg dont-care érték volt, így nyugodtan átállíthatjuk 0-ra, hiszen amikor D állapotban eredetileg ezt a cellát használjuk, akkor 1-esről a kimenet 0-ra változik a D → A állapotugrás közben. Viszont az nem lényeges, hogy ez a kimenetváltozás mikor történik meg. Erről az állapottáblák felvételénél már volt szó.

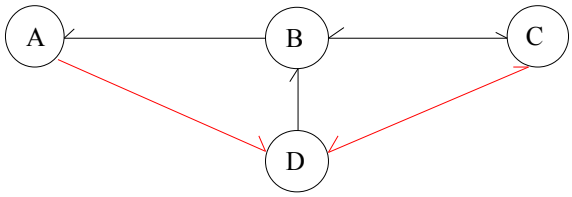
| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | D, 0 | B, 0 |
| B | | B, 1 | B, 1 | C, 1 | B, 1 |
| C | | B, 0 | D, 0 | C, 0 | C, 0 |
| D | | A, 1 | A, 0 | D, 1 | C, 1 |

Ezzel olyan módosításokat végeztünk, amik nem befolyásolják a hálózat működését, de megkönnyítik a kódválasztást.

MP:
Állapottáblájával adott a következő hálózat. Kódoljuk az állapotokat úgy, hogy minden állapotváltozásnál legfeljebb 1 szekunder változó változzon.
**Az összes nem stabil állapotot is végig kell követnünk nem normál hálózat esetében!**

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | A, 0 | D, - |
| B | | A, - | B, 1 | C, 0 | B, 0 |
| C | | D, 1 | C, 1 | C, 0 | B, 0 |
| D | | D, 1 | B, 1 | C, -0 | D, 1 |

A szomszédos D-vel.
B szomszédos A-val és C-vel.
C szomszédos B-vel és D-vel.
D szomszédos C-vel és B-vel.




Ha felrajzoljuk az állapotgráfot, jól látható, hogy az A ↔ D, és D ↔ C átmenetek problémásak számunkra.
Ha a problémás átmenetek nem lennének, akkor egyből be is írhatnánk az állapotgráfot a Karnaugh táblába.
A következő az ötletünk: vegyünk fel egy új X állapotot A és D közé, és egy Y állapotot D és C közé.

Karnaugh-tábla:

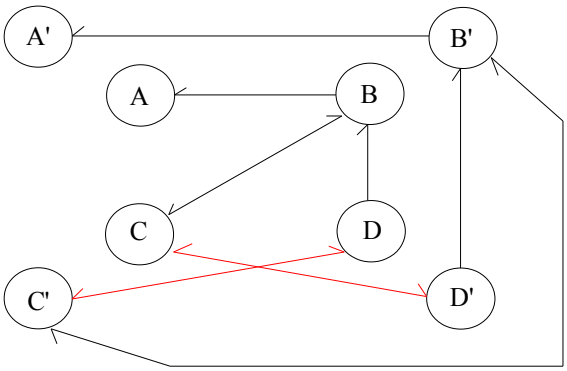
| | | | $y_3$ | |
|---|---|---|---|---|
| | A | B | C | |
| $y_1$ | X | D | Y | |
| | | $y_2$ | | |

Majd egészítsük ki az állapottáblát:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | A, 0 | X, - |
| B | | A, - | B, 1 | C, 0 | B, 0 |
| C | | Y, 1 | C, 1 | C, 0 | B, 0 |
| D | | D, 1 | B, 1 | Y, -0 | D, 1 |
| X | | - | - | - | D, 1 |
| Y | | D, 1 | - | C, 0 | - |

Egyéb ötlet:
Vegyünk fel minden állapothoz egy társállapotot és vegyünk fel még 1 bitet. Az eredeti állapotoknál az első bitet 0-ra rögzítjük, a párjuknál az első bitet 1-re.
A problémás állapotátmenetet a két különböző kör közé helyezzük, és így minden szomszédos lesz, amennyiben a külső kör bináris kódjait megfelelően elforgatjuk.
Ez a módszer is csak szigorú feltételek mellett működik!
Ne felejtsük el most is kiegészíteni az állapottáblát!

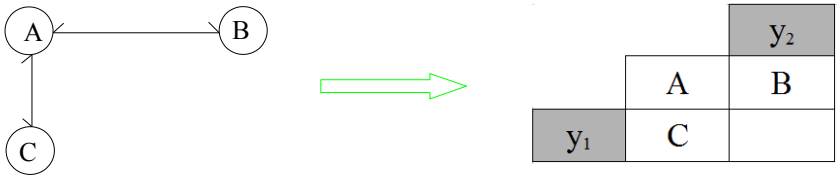
Az előző példa megoldása a következő volna az A → D átmenet nélkül ezzel a módszerrel:



Mint azt már említettük, ezek a módszerek intuitív módszerek. Ennek ellenére megfelelő mennyiségű gyakorlással könnyedén észrevehetjük, hogy a felsoroltak közül melyik az, ami az adott feladathoz megfelelő eredményt adhat.

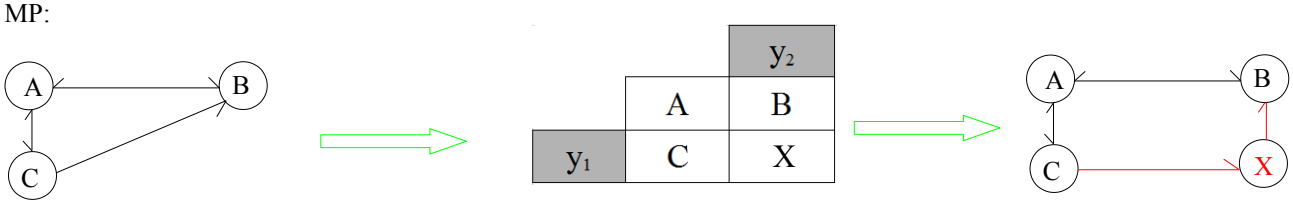
Nézzünk pár könnyen felismerhető állapotgráfot és a hozzájuk tartozó kódolást:

MP:



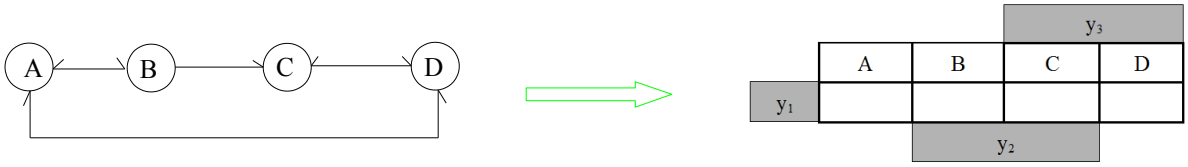

---

MP:




---

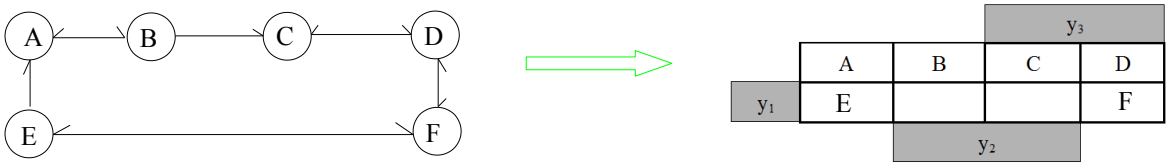
MP:




A Karnaugh-tábla alapján látható, hogy valójában 2 bit is elég volna.
Ha az állapotgráfot másként rajzolnánk fel, akkor abból adódna a két változós Karnaugh-tábla kitöltése.

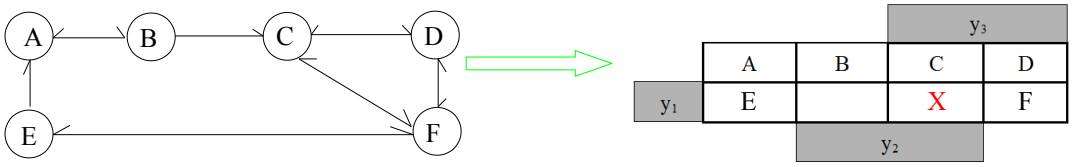
---

MP:




---

MP:




Itt is új köztes állapot segít nekünk a megfelelő kódolás megtalálásában.

---

## Szinkron hálózat állapotainak kódválasztása:

Szinkron hálózatok esetében is ugyan úgy bináris kódokkal kell leírnunk az állapotokat.
Most a feladat nem az, hogy hazárdokat és problémás működést kerüljük el, hanem olyan optimális kódokat kell találnunk, amik a hálózat megvalósításakor egyszerűsítéseket tesznek lehetővé.
Az ideális állapotkódolás NP-teljes probléma, de különböző elveket betartva nagy valószínűséggel jobb kódokat kaphatunk, minthogyha véletlenszerűen kódolnánk a hálózatot.
A módszerek elméleti hátterére most külön nem térünk ki.

**Szomszédos állapkódolás:**

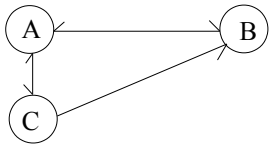
Három egyszerű szabályt kell betartanunk az állapottábla vagy az állapotgráf alapján:

1. Azokhoz az állapotokhoz, amik ugyan arra az állapotra mutatnak ***valamilyen azonos bemenet mellett***, rendeljünk páronként szomszédos kódokat.

2. Ha egy állapot több állapotra mutat, akkor ezekhez az állapotokhoz rendeljünk páronként szomszédos kódokat.

3. Az első szabály az erősebb.

MP:



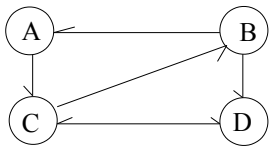

Az első szabály értelmében kapjon szomszédos kódot az A - C állapot, és a C - B állapot.

Példa a kódolásra az első szabály alapján:
A: 00
B: 11
C: 01

Figyeljünk arra, hogy azonos bemeneti kombináció szükséges az 1. szabályhoz!
A példában feltételeztük ennek a feltételnek a teljesülését.

---

MP:



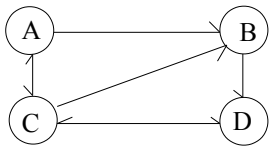

Az első szabály értelmében kapjon szomszédos kódot a B - C állapot.
A második szabály értelmében kapjon szomszédos kódot az A - D és a B - D állapot.

Példa a kódolásra az első szabály alapján:
A: 00
B: 11
C: 01
D: 10

Figyeljünk arra, hogy azonos bemeneti kombináció szükséges az 1. szabályhoz!
A példában feltételeztük ennek a feltételnek a teljesülését.

---

MP:




Az első szabály értelmében kapjon szomszédos kódot az A - C és a B - C és az A - D állapot.
A második szabály értelmében kaphatna szomszédos kódot az A - B és a A - D és a B - D állapot.

Példa a kódolásra az első szabály alapján:
A: 00
B: 11
C: 01
D: 10

Figyeljünk arra, hogy azonos bemeneti kombináció szükséges az 1. szabályhoz!
A példában feltételeztük ennek a feltételnek a teljesülését.

Az előző oldalon szereplő utolsó példánál véletlenszerűen választottunk kódolást, de a lépcsős tábla segítségünkre lehet a szomszédos kódolásnál az optimális kódolás kiválasztására.

Adott a következő hálózat:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | A, 0 | C, 0 |
| B | | A, - | B, 1 | C, 0 | B, 0 |
| C | | B, 1 | C, 1 | C, 0 | B, 0 |
| D | | D, 1 | B, 1 | D, 0 | D, 1 |

Írjuk fel a lépcsős táblát az állapotok alapján, és két számot írjunk bele. Első szám jelentse azt, hogy a cellának megfelelő párosítás hánysor szerepelt az első szabály alapján, második szám legyen az, hogy a második szabály alapján hányszor szerepelnek.
Menjünk végig az összes bemeneti kombináción, és írjuk fel mindig hogy melyik szabály alapján mi legyen szomszédos.
Első szabály alapján legyen szomszédos A és B, B és D, B és C, B és C.
Második szabály alapján legyen szomszédos A és C, A és B, A és C, B és C, B és C, D és B.

| | | | |
|---|---|---|---|
| B | 1 1 | | |
| C | 0 2 | 2 2 | |
| D | 0 0 | 1 1 | 0 0 |
| | A | B | C |

Ezek alapján azokat a szomszédokat kell preferálnunk, amelyek az első szabály szerint legtöbbször szerepelnek, és ha még mindig nem tudunk dönteni, akkor azt is figyelembe kell vennünk, hogy a második szabály szerint hányszor szerepelnek.

MP:
Állapottáblájával adott a következő szinkron hálózat. Kódoljuk szomszédosan az állapotokat:

| y | X | 0 | 1 |
|---|---|---|---|
| A | | C, 0 | B, 1 |
| B | | B, 1 | B, 0 |
| C | | B, 1 | D, 0 |
| D | | B, 1 | A, 0 |

Az első szabály alapján a következő adódik:
- Legyenek páronként szomszédosak a B-C-D állapotok.
- Legyenek páronként szomszédosak az A-B állapotok.

| y | X | 0 | 1 |
|---|---|---|---|
| A | | C, 0 | B, 1 |
| B | | B, 1 | B, 0 |
| C | | B, 1 | D, 0 |
| D | | B, 1 | A, 0 |

A második szabály alapján a következő adódik:
- Legyenek szomszédosak C és B állapotok.
- Legyenek szomszédosak B és B állapotok. Ez nem valódi feltétel.
- Legyenek szomszédosak a B és D állapotok.
- Legyenek szomszédosak a B és A állapotok.

Ebben az esetben az 1. szabály feltételei lefedik a második szabály feltételeit.

3 szekunder változóval nem tudjuk maradéktalanul teljesíteni az összes feltételt, ez azonban nem jelent problémát.

| | | | $y_3$ | |
|---|---|---|---|---|
| | B | C | D | |
| $y_1$ | A | | | |
| | | $y_2$ | | |

**HT partíciók keresése:**

Helyettesítési tulajdonságú (HT) partíciók alapján is kódolhatjuk az állapotokat.
Az állapotok halmazát olyan független részhalmazokra osztjuk, melyek uniója tartalmazza az összes állapotot.
Egy ilyen felosztást egy partíciónak nevezünk.

Egy partíció blokkokból áll, amik olyan tulajdonságúak, hogy ismerve, hogy egy állapot a partíció mely blokkjában van, a bemenet ismeretében meg tudjuk mondani, hogy a következő állapot mely blokkban lesz, de azt, hogy a blokkon belül melyik állapot az, azt nem tudjuk megmondani.

Létezik két **triviális partíció**:
- Minden állapot egy blokkban van.
- Minden állapot külön blokkban van.

A triviálistól eltérő partíciók esetén önfüggő szekunder változó csoportokat hozhatunk létre, és a célunk, hogy egy optimális kódolást találjunk.

Egy partícióhoz külön kódolnunk kell a csoportokat, és külön kódolnunk kell a csoporton belül az állapotokat is.
Az a célunk, hogy a lehető legkevesebb szekunder változót kelljen használnunk.

**Zártnak** nevezünk egy partíciót, ha a partícióból minden bemenetre olyan állapotokba ugrunk, amely állapotok közös csoportban vannak, azaz minden csoportból pontosan egy csoportba tudunk átugrani.

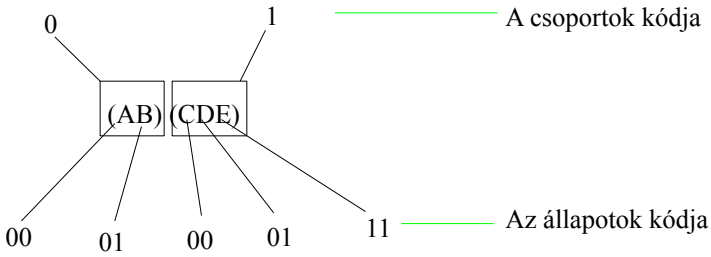
MP:
A fenti fogalmak bemutatásához vegyünk egy hálózatot 5 állapottal: A, B, C, D, E

A triviális partíciók:
- (A) (B) (C) (D) (E) → Az összes állapot külön csoportban van.
- (ABCDE) → Az összes állapot egy csoportban van.

A triviális partíciókhoz szükséges szekunder változók száma:
- (A) (B) (C) (D) (E) → Csak a csoportokat kell kódolnunk, a csoportokon belül nincs szükség szekunder változóra, hiszen mindenhol csak 1 állapot van. 5 csoport kódolásához 3 szekunder változóra van szükségünk.
- (ABCDE) → Egyetlen csoport van, nincs szükség a csoportok kódolására külön szekunder változóra. Azonban most a csoporton belül az állapotokat kódolnunk kell. Ismét 3 szekunder változóra van szükségünk.

A triviálistól eltérő partíció:
-(AB)(CDE) → Külön kell kódolnunk a csoportokat. 2 csoport esetén 1 darab változóra van szükség. Az egyik csoport kapja a 0-t a másik pedig az 1-et.

→ Külön kell kódolnunk a csoportokon belül az állapotokat. A legnagyobb csoport lesz a mérvadó, ez esetben a második. 3 állapot kódolásához 2 szekunder változó szükséges.




A következő kódokat kapjuk tehát:

A: 000
B: 001
C: 100
D: 101
E: 111

Először a csoportot kódoló bitek, majd a csoporton belüli állapotot leíró bitek következnek.

Az **önfüggő változók** a csoportokat kódoló változók. Triviális esetben ezek száma 0 és 3, míg a példában szereplő triviálistól eltérő esetben 1 (ne keverjük össze a csoportok számával).

Szisztematikus módszer HT partíciók keresésére:

HT partíciók kereséséhez a következő leírást kell követnünk:
Fogjuk a hálózatunk állapottábláját, és készítsünk egy lépcsős táblát pont úgy, ahogy azt az állapotösszevonásoknál is láthattuk. **DE** ebben az esetben a kimeneteket nem kell vizsgálnunk. Ha felvettük a lépcsős táblát, akkor minden egyes cellához megnézzük, hogy milyen partíciók alakulnak ki a feltételeknek megfelelően, megkeressük a legkedvezőbb partíciót, és kódokat választunk.
Nehéz volna érthetően megfogalmazni, hogy mit is csinálunk, póbáljuk meg követni a nyilakat, és felfedezni a rendszert a következő mintapélda megoldásában.

MP:
Tekintsük az alábbi hálózatot és kódoljuk a fentebb ismertetett módszerrel:

| y \ X | 0 | 1 |
|---|---|---|
| A | C, 0 | E, 1 |
| B | E, 1 | B, 0 |
| C | D, 1 | F, 0 |
| D | E, 1 | D, 0 |
| E | D, 1 | E, 0 |
| F | B, 1 | A, 0 |

Vegyük fel a lépcsős táblát, és írjuk fel a feltételeket, mintha állapotokat akarnánk összevonni, de a kimenetek nem számítanak!

| | | | | | |
|---|---|---|---|---|---|
| B | CE, EB | | | | |
| C | CD, EF | DE, BF | | | |
| D | CE, DE | | DE, DF | | |
| E | CD | DE | EF | | |
| F | CB, AE | BE, AB | BD, AF | BE, AD | AE, BD |
| | A | B | C | D | E |

Most minden cellára nézzük meg, hogy milyen partíció adódik:

AB: (AB)(CE)(EB) → (ABCE)(D)(F)

ABE

ABCE

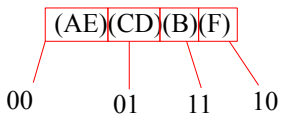
A párok nagyobb csoportokra olvadnak össze, amennyiben közös tagjuk van, és ezek a nagyobb csoportok is hízhatnak.

Kapott partíciók:

1. AC: (AC)(CD)(EF) → (ACD)(EF)(B)
2. AD: (AD)(CE)(DE) → (ACDE)(B)(F)
3. AE: (AE)(CD) → (AE)(CD)(B)(F)
4. AF: (AF)(CB)(AE) → (AEF)(CB)(D)
5. BC: (BC)(DE)(BF) → (BCF)(DE)(A)
6. BD: (BD) → (BD)(A)(C)(E)(F)
7. BE: (BE)(DE) → (BDE)(A)(C)(F)
8. BF: (BF)(BE)(AB) → (ABEF)(C)(D)
9. CD: (CD)(DE)(DF) → (CDEF)(A)(B)
10. CE: (CE)(EF) → (CEF)(A)(B)(D)
11. CF: (CF)(BD)(AF) → (ACF)(BD)(E)
12. DE: (DE) → (DE)(A)(B)(C)(F)
13. DF: (DF)(BE)(AD) → (ADF)(BE)(C)
14. EF: (EF)(AE)(BD) → (AEF)(BD)(C)

Az 1. partíciót 2 önfüggő, és 2 nem önfüggő szekunder változóval kódolhatjuk. Összesen 2+2=4 szekunder változó.
A 2. partíciót 2 önfüggő, és 2 nem önfüggő szekunder változóval kódolhatjuk. Összesen 2+2=4 szekunder változó.
…
A 6. partíciót 3 önfüggő, és 1 nem önfüggő szekunder változóval kódolhatjuk. Összesen 3+1=4 szekunder változó.

Az optimális a 3. partíció, mivel azt 2 önfüggő, és 1 nem önfüggő, azaz összesen 3 szekunder változóval kódolhatjuk.
Az adódó kódolás:




A: 000
B: 110 / 111
C: 010
D: 011
E: 001
F: 100 / 101

## Kódolt állapottáblák:

Egy-egy feladat megvalósításában a következő lépést a kódolt állapottábla felírása jelenti.

A kódolt állapottábla lényege, hogy az ABC-vel jelölt állapotokat mindenhol helyettesítjük az állapotkódolásnál kapott kódjukkal.

Ez egy könnyen emészthető anyag, a fentebbi egy sor bőven elég volna a megértéshez, de azért nézzünk egy példát:

A következő állapottáblánk van:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | | A, 0 | A, 0 | A, 0 | B, 0 |
| B | | A, 0 | D, 0 | D, 0 | B, 0 |
| C | | D, 1 | A, 0 | A, 0 | C, 1 |
| D | | D, 1 | D, 1 | D, 1 | C, 1 |

A kódolás folyamén (mindegy, hogy szinkron vagy aszinkron) a következő állapotkódokat kaptuk:
A: 00
B: 01
C: 10
D: 11

Írjuk fel a kódolt állapottáblát:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 10 | | 11, 1 | 00, 0 | 00, 0 | 10, 1 |
| 11 | | 11, 1 | 11, 1 | 11, 1 | 10, 1 |

Mint látható, semmi más nem történt, csak az állapotok betűjelét helyettesítettük a kódjukkal.
Ez azonban fontos lesz

Amennyiben a nem használt állapotkódokhoz dont care értékeket teszünk, egyszerűsítések adódhatnak a későbbiekben.

A megbízható működés érdekében előírhatjuk, hogy a nem használt állapotkódokhoz tartozó állapotokból is egy létező állapotba kerüljön a hálózat, ugyanis szerencsétlen esetben a dont care értékek miatt, akár nem használt állapotokban is ragadhat a hálózatunk.

## Aszinkron hálózatok függvényeinek felvétele:

Visszacsatolt aszinkron hálózat esetén a függvényeinket rögtön fel tudjuk írni a kódolt állapottáblából.
A hálózatnak elő kell állítania a szekunder változókat és a kimeneteket (egyszerer mindig csak 1 bitet tudunk előállítani). Flip-flopokat is használhatunk, ekkor a szinkron esetben majd ismertetett vezérlési táblákat kell felvennünk. A különbség annyi lesz, hogy a flip-flopok nem kapnak majd órajelet, és csak aszinkron flip-flopokat használhatunk majd.

Ezt egy 4x4-es állapottáblán mutatjuk meg, ugyanis ebben az esetben a legegyszerűbb a Karnaugh-táblák felírása, de ez természetesen minden hálózatra megoldható a korábban már ismert módszerekkel.

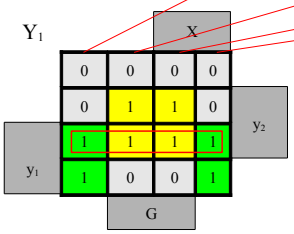
Vegyük a fentebb vizsgált kódolt állapottáblát:

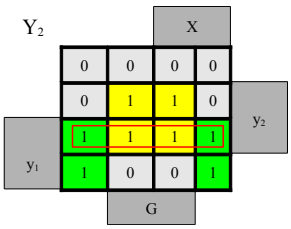
| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 10 | | 11, 1 | 00, 0 | 00, 0 | 10, 1 |
| 11 | | 11, 1 | 11, 1 | 11, 1 | 10, 1 |

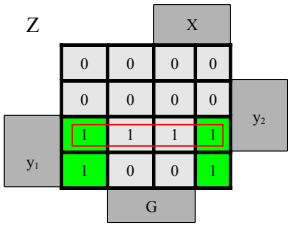
Mint azt már említettük, az ilyen 4x4-es táblázatokból nagyon könnyű Karnaugh táblákat felvenni, és a példa jó olyan szempontból, hogy figyelnünk kell a sorok sorrendjére. A bemeneteknél már felcseréltük a bináris sorrendeket, hogy a Karnaugh tábla peremezése jó legyen, most a sorokat is olyan sorrendbe kell leírnunk, hogy a peremezés itt is helyes legyen:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | | 11, 1 | 11, 1 | 11, 1 | 10, 1 |
| 10 | | 11, 1 | 00, 0 | 00, 0 | 10, 1 |

Jelenleg 3 függvényt kell leírnunk ($Y_1$, $Y_2$, $Z$):




A következő függvények adódnak:

$Y_1 = G*y_2 + /G*y_1 + y_1*y_2$

$Y_2 = G*y_2 + /G*y_1 + y_1*y_2$

$Z_1 = /G*y_1 + y_1*y_2$

Ezeket a függvényeket könnyedén megvalósíthatjuk, és a visszacsatolással válik a kombinációs hálózatunk sorrendi hálózattá.

## Vezérélési tábla felvétele szinkron hálózatokhoz:

Szinkron hálózatok tervezéséhez állapotregisztereket kell választatnunk. Eddig ezeket flip-flopoknak neveztük.
Az aszinkron hálózathoz hasonlóan, most is külön-külön minden bitet elő kell állítanunk.

A flip-flop kiválasztása hatással lesz a vezérlő hálózatra.
A helyzetünk akkor lesz a legegyszerűbb, ha D flip-flopokat használunk. Ekkor a vezérlési tábla megegyezik a kódolt állapottáblával.

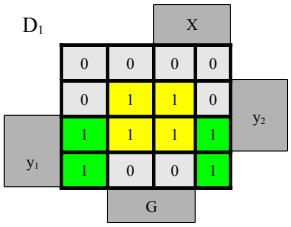
Kódolt állapottábla:

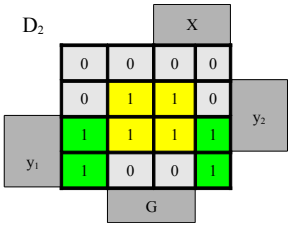
| y \ XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 10 | 11, 1 | 00, 0 | 00, 0 | 10, 1 |
| 11 | 11, 1 | 11, 1 | 11, 1 | 10, 1 |

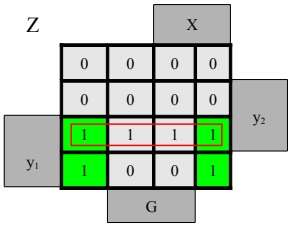
Vezérlési tábla azt tartalmazza, hogy mit kell kötnünk az adott flip-flop bemenetére (/bemeneteire) ahhoz, hogy az adott állapotban (az adott bitet előállító flip-flop állapota oldalt olvasható, $y_1$-et előállító flip-flop állapota az adott sorban $y_1$ értéke, $y_2$-t előállítóé pedig $y_2$ értéke) a kimenetén az állapottáblában lévő érték jelenjen meg.
Ehhez ismernünk kell az adott flip-flop állapottábláját.
D flip-flop esetén nagyon egyszerű a vezérlési tábla kitöltése, hiszen a kimenete megegyezik a bemenetével.
2 szekunderváltozót kell előállítanunk, így 2 D flip-flop szükséges.

A vezérlési tábla:

| y \ XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| | $D_1D_2$, Z | $D_1D_2$, Z | $D_1D_2$, Z | $D_1D_2$, Z |
| 00 | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | 11, 1 | 11, 1 | 11, 1 | 10, 1 |
| 10 | 11, 1 | 00, 0 | 00, 0 | 10, 1 |




A következő függvények adódnak, ha az aszinkron hálózatoknál megismert módszerrel:

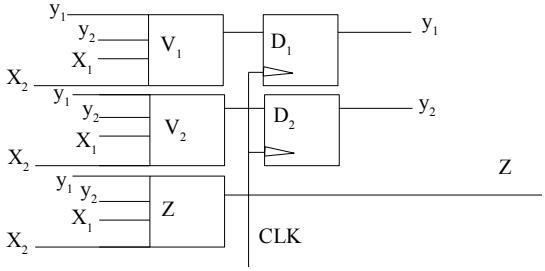
$D_1 = G*y_2 + /G*y_1$

$D_2 = G*y_2 + /G*y_1$

$Z = /G*y_1 + y_1*y_2$

**Szinkron hálózat vezérlő függvényeit nem kell hazárdmentesíteni!**

A hálózat struktúrája a következő lesz:

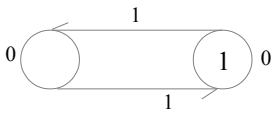


MP:
Valósítsuk meg az előző feladatot T flip-flop felhasználásával:

Kódolt állapottábla:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | | 11, 1 | 11, 0 | 11, 0 | 10, 1 |
| 10 | | 11, 1 | 00, 1 | 00, 1 | 10, 1 |

| y | T | 0 | 1 |
|---|---|---|---|
| 0 | | 0 | 1 |
| 1 | | 1 | 0 |




A megértés könnyítésének érdekében vegyükn fel egy újabb táblázatot, mely most csak az $y_1$-eket tartalmazza (a továbbiakban erre nincs szükség, jelenleg csupán segédeszköz):

| $y_1$ | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | 0 | 0 |
| 0 | | 0 | 1 | 1 | 0 |
| 1 | | 1 | 1 | 1 | 1 |
| 1 | | 1 | 0 | 0 | 1 |

Vizsgáljuk meg az első cellát (piros):
A flip-flop 0 állapotban van (bal oldalról leolvashatóak az állapotok), és a kimenetén 0-t kell adnia. Ekkor 0-val kell vezérelnünk a T flip-flopunkat (ezt az állapottáblájából olvassuk ki).

Vizsgáljunk meg egy másik cellát (zöld):
A flip-flop 1 állapotban van, a kimenetén 1-t kell adnia, tehát a T flip-flopunkat 0-val kell vezérelnünk.

A vezérlési tábla:

| $y_1$ | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| | | $D_1$ | $D_1$ | $D_1$ | $D_1$ |
| 0 | | 0 | | | |
| 0 | | | | | |
| 1 | | | 0 | | |
| 1 | | | | | |

A logikát követve kitölthetjük a teljes vezérlési táblát. (Figyeljünk arra, hogy az $y_2$-t előállító flip-flop $y_2$ állapotban van):

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| | | $D_1D_2$, Z | $D_1D_2$, Z | $D_1D_2$, Z | $D_1D_2$, Z |
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 01, 0 | 10, 0 | 10, 0 | 00, 0 |
| 11 | | 00, 1 | 00, 0 | 00, 0 | 01, 1 |
| 10 | | 01, 1 | 10, 1 | 10, 1 | 00, 1 |

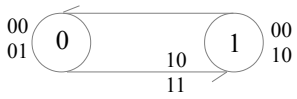
MP:
Valósítsuk meg az előző feladatot JK flip-flop felhasználásával:

Kódolt állapottábla:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | | 11, 1 | 11, 0 | 11, 0 | 10, 1 |
| 10 | | 11, 1 | 00, 1 | 00, 1 | 10, 1 |

JK flip-flop állapottáblája és állapotgráfja segítségként:

| y | JK | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | 1 | 1 |
| 1 | | 1 | 0 | 0 | 1 |




Ebben az esetben is 2 flip-flopot kell használnunk, de most mind a kettőhöz elő kell állítanunk egy J és egy K bemenetet.

Nézzük meg az első cella jelentését most is:
Az $y_1$-et előállító flip-flop 0 állapotban van, és 0 kimenetet kell adnia. Ekkor a JK flip-flop állapottáblája alapján 00-val vagy 01-el kell vezérelnünk, tehát a vezérlési táblába 0- érték kerül az első 0 helyére.
Ugyan itt az y2-t előállító flip-flop szintén 0 állapotban van, a kimenetén szintén 0-t kell előállítania, így őt is 0-értékkel kell vezérelnünk.

A vezérlési tábla:

| y | XG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| | | $J_1K_1J_2K_2$, Z | $J_1K_1J_2K_2$, Z | $J_1K_1J_2K_2$, Z | $J_1K_1J_2K_2$, Z |
| 00 | | 0-0-, 0 | 0-0-, 0 | 0-0-, 0 | 0-1-, 0 |
| 01 | | 0--1, 0 | 1--1, 0 | 1--1, 0 | 0--0, 0 |
| 11 | | -0-0, 1 | -1-1, 0 | -1-1, 0 | -1-0, 1 |
| 10 | | -1-0, 1 | -00-, 1 | -00-, 1 | -10-, 1 |

A már megismert módon most is felírhatjuk a $J_1$, $K_1$, $J_2$, $K_2$ és Z függvényeket.

## Sorrendi hálózat megvalósítása:

Ahhoz, hogy átlássuk egy feladat megvalósításának folyamatát, valósítsunk meg egy egyszerű mintapéldát:

MP:
Készítsünk egy szinkron sorrendi hálózatot, amely számlálóként működik.
1 darab X bemenete, és 2 darab kimenete van, a működése pedig a következő:
Ha X=1, akkor a számláló felfelé számlál egyesével. Ha X=0, akkor a számláló lefelé lépked.
A számláló a minimum és a maximum értékeknél megáll.
A kimeneten az aktuális érték jelenik meg.

Vegyük fel az előzetes állapottáblát:

| y | X | 0 | 1 |
|---|---|---|---|
| A (0) | | A, 00 | B, 00 |
| B (1) | | A, 01 | C, 01 |
| C (2) | | B, 10 | D, 10 |
| D (3) | | C, 11 | D, 11 |

Végezzük el az állapotösszevonásokat:

Jelenleg a példa egyszerűsége miatt az előzetes állapottábla is minimális számú állapotot tartalmaz, így ezt a fázist átugorjuk.

Kódoljuk az állapotokat:

Használjuk a szomszédos kódolást:
Az első szabály alapján legyen szomszédos az A-B és a C-D állapot.
A második szabály alapján legyen szomszédos az A-B, A-C, B-D, C-D állapotok.

A következő állapotkódok adódnak:

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

Írjuk fel a kódolt állapottáblát:

| y | X | 0 | 1 |
|---|---|---|---|
| 00 | | 00, 00 | 01, 00 |
| 01 | | 00, 01 | 10, 01 |
| 10 | | 01, 10 | 11, 10 |
| 11 | | 10, 11 | 11, 11 |

Cseréljük fel a C és D állapotot, hogy könnyebben fel tudjuk majd írni a Karnaugh-táblákat a grafikus minimalizáláshoz:

| y | X | 0 | 1 |
|---|---|---|---|
| 00 | | 00, 00 | 01, 00 |
| 01 | | 00, 01 | 10, 01 |
| 11 | | 10, 11 | 11, 11 |
| 10 | | 01, 10 | 11, 10 |

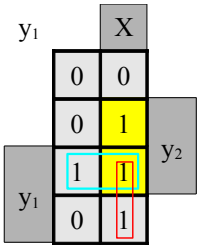
Válasszunk flip-flopokat állapotregiszterként:

Az egyszerűség kedvéért válasszuk a D flip-flopot:

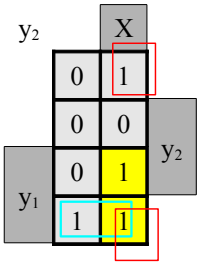
Vezérlési tábla:

| y | X | 0 | 1 |
|---|---|---|---|
| 00 | | 00, 00 | 01, 00 |
| 01 | | 00, 01 | 10, 01 |
| 11 | | 10, 11 | 11, 11 |
| 10 | | 01, 10 | 11, 10 |

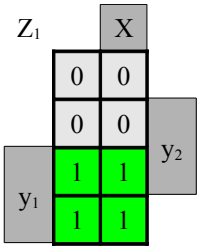
Határozzuk meg a függvényeket grafikus minimalizálással:



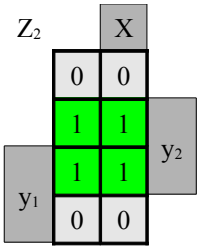

$y_1 = y_2*X + y_1*y_2 + y_1*x$




$y_2 = /y_2*X + y_1*/y_2 + y_1*x$




$Z_1 = y_1$




$Z_2 = y_2$

Készítsünk a függvényeink alapján elvi logikai rajzot.

Készítsünk a logikai rajz alapján kapcsolási rajzot, nyáktervet, és építsük meg a hálózatunkat.

Használjuk a megvalósított hálózatunk.

MP:
Szinkron soros összeadó megvalósítása:
Két bemenettel ($X_1$, $X_2$) és 1 kimenettel (Z) rendelkezik, a működése pedig a következő:
Két bináris szám összeadására szolgál. A legkisebb helyi értéktől kezdve összeadja a számjegyeket (minden ütemben 1-1 bitet), összegüket kiadja a kimeneten, és emlékszik, hogy az előző összeadásból maradt-e túlcsordulás.

A hálózat előtt és mögött problémákat kell megoldanunk, azonban ezekkel most nem foglalkozunk:
Meg kell oldanunk, hogy a bemeneteken órajelre változzanak az értékek.
Meg kell oldanunk, hogy a kimenet utáni készülék megjegyezze az értékeket.

A működés leírásából kiderül, hogy nekünk 2 állapot elég lesz, ugyanis arra kell csak emlékeznie a hálózatnak, hogy van-e túlcsorduló bit.

Vegyük fel az állapottáblát:

| y | $X_1X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A (nincs túlcsordulás) | | A, 0 | A, 1 | B, 0 | A, 1 |
| B (van túlcsordulás) | | A, 1 | B, 0 | B, 1 | B, 0 |

Állapotösszevonásra ebben az esetben sincs szükség (elég egyszerűek a feladatok ahhoz, hogy egyből minimális állapotot használjunk).

Az állapotkódolás elég egyszerű, 2 állapotot 1 bittel kódolhatunk. Legyen A=0 és B=1.

A kódolt állapottábla:

| y | $X_1X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| 0 | | 0, 0 | 1, 0 | 0, 1 | 1, 0 |
| 1 | | 1, 0 | 0, 1 | 1, 1 | 0, 1 |

A függvényeinket ismét grafikus minimalizálással keressük:

y

| y \ $X_1X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

Z:

| y \ $X_1X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

$y = X_2*y + X_1*X_2 + X_1*y$

$Z = S^3_{1,3}$ (A szimmetrikus függvényekről már korábban beszéltünk, természetesen felírhatjuk egyesével is az összes értéket.)

Készítsünk a függvényeink alapján elvi logikai rajzot.
Készítsünk a logikai rajz alapján kapcsolási rajzot, nyáktervet, és építsük meg a hálózatunkat.
Használjuk a megvalósított hálózatunk.

# Hazárdok sorrendi hálózatokban:

**Gerjedés**:
Szinkron hálózatoknál, ha a felfutó él és a változás éle egybeesik, akkor a flip-flop nem tudja eldönteni milyen állapotba kerüljön, és gerjedés jelensége tapasztalható. Minél gyorsabb a rendszer, annál kisebb az esélye, hogy a felfutó él és a változás egybeessen.

**Kritikus versenyhelyzet**:
A különböző hálózati elemek különböző mértékben késleltetik a jelterjedést, ez bizonyos esetekben problémákat okozhat. Versenyhelyzetről akkor beszélünk, ha a rendszer kimenete és állapota függ attól, hogy egymástól független változások milyen sorrendben, vagy milyen időeltolódással zajlanak le. Kombinációs hálózatoknál is beszélhetünk versenyhelyzetről.

Aszinkron sorrendi hálózatoknál, ahol egynél több szekunderváltozó van, amikor állapotot váltunk, akkor a szekunder változók megváltoznak.
Amennyiben egynél több szekunderváltozó változik az adott állapotváltásnál, akkor az új értéket felvevő változók versenyeznek egymással a jelterjedési sebességek különbsége miatt, hiszen ha az egyik sokkal gyorsabban változik, mint a többi, akkor ez a változás olyan hatással lehet a hálózatra, hogy az rossz állapotba kerül. Ezt a versenyt nevezzük kritikus versenyhelyzetnek.
A kritikus versenyhelyzetet a megfelelő kódolással vagy szinkron sorrendi hálózat használatával kerülhetjük el.

Kritikus versenyhelyzet kódolt állapottáblában:

| $y_1, y_2$ | $X_1, X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | 00 | 00, 0 | 00, 0 | 00, 0 | 11, 0 |
| B | 01 | 00, 0 | --, - | 10, - | 01, 0 |
| C | 10 | 11, 1 | 10, 1 | 10, 1 | 10, 1 |
| D | 11 | 00, 1 | 01, - | --, - | 10, 1 |

**Lényeges hazárd:**
Lényeges hazárd csak aszinkron sorrendi hálózatokban jelentkezhet.
A két szekunder változó nem ugyan olyan gyorsan változik, és előfordulhat, hogy az egyik már érzékelte a bementi változást, a másik viszont még a nem, de a másik szekunder változását már igen, így azonban olyan állapotba kerülhetünk, ami nem megfelelő működést eredményez, amikor a második változó is megérzi a bemeneti változást.
Csak akkor fordulhat elő, ha az állapottábla **Unger konfigurációt** tartalmaz.
Ezt a hazárdot csak szándékos késleltetéssel tudjuk elkerülni!

Az Unger konfiguráció formálisan a következő módon kereshető meg:
Egy stabil állapotból indulunk (a példában 00 állapot és 11 bemenet), és megvizsgálunk egy bemeneti változást (a példában 11 → 10). Arra az állapotra kerülünk, ahova normális működés esetén kerülnünk kell. Változtassuk meg a bemenetet a kiindulási bemenetre (10 → 11), kövessük a hálózatot egy stabil állapotig, majd ismét vizsgáljuk az eredeti bemenetváltozást (11 → 10). Amennyiben nem ugyan oda kerültünk, ahova normális működés esetén kerülnünk kellett volna, a hálózat lényeges hazárdot tartalmaz.

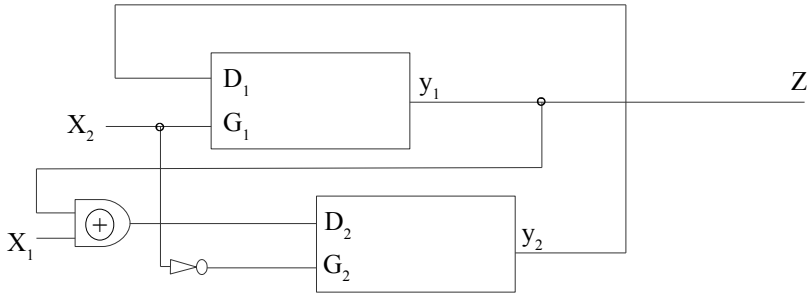
| $y_1, y_2$ | $X_1, X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| A | 00 | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| B | 01 | 00, 0 | --, - | 10, - | 01, 0 |
| C | 10 | 11, 1 | 10, 1 | 10, 1 | 10, 1 |
| D | 11 | 11, 1 | 00, - | --, - | 10, 1 |

**Rendszerhazár / órajel elcsúszásnak (clock skew):**
Olyan helyzetben, ahol két flip-flop sorba van kötve, azaz az egyik bemenete egy másik kimenetétől függ, és mindkét flip-flop ugyan azt az órajelet kapja, a jelterjedésből származó késleltetés hibás működést eredményezhet, mivel lehetséges, hogy az egyik korábban kapja meg az órajelet, és így még rossz értékből vesz mintát.
A megoldást a data-lock out flip-flopok használata jelentheti.

## Sorrendi hálózatok analízise:

MP: Analizáljuk a következő hálózatot:

1. lépés: az elvi logikai rajz visszafejtése:




$D_1 = y_2$

$D_2 = X_1 \oplus y_1$

$G_1 = X_2$

$G_2 = /X_2$

$Z = y_1$

Olvassuk le az egyes bemenetekhez és kimenetekhez tartozó vezérlési függvényeket.
Ne féljünk, csak kövessük a vonalakat.

2. lépés: vegyük fel a vezérlési táblát:

A fentebb meghatározott vezérlési függvények alapján vegyük fel a vezérlési táblát.
Kitölthetjük a Karnaugh táblákat, és azokat bemásolhatjuk helyes sorrendben a táblázatba, a hálózatok megvalósításánál tanult módszer visszafelé alkalmazásával, vagy kitölthetünk minden cellát egyesével is.
A kimenetet is beírhatjuk, de nem feltétlenül szükséges.

| $y_1, y_2$ \ $X_1, X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| $y_1, y_2$ | $D_1, G_1, D_2, G_2$ | $D_1, G_1, D_2, G_2$ | $D_1, G_1, D_2, G_2$ | $D_1, G_1, D_2, G_2$ |
| 00 | 0001 | 0100 | 0110 | 0011 |
| 01 | 1001 | 1100 | 1110 | 1011 |
| 11 | 1011 | 1110 | 1100 | 1001 |
| 10 | 0011 | 0110 | 0100 | 0001 |

3. lépés: vegyük fel a kódolt állapottáblát:

A vezérlési táblát fejtsük vissza kódolt állapottáblává. Az új $y_i$ szekunder változót az i-edik DG flip-flop állítja elő. Vizsgáljuk meg, hogy az adott cellában az i-edik flip-flop milyen állapotban van, és a vezérlési táblából olvassuk ki, hogy itt mivel vezéreljük.

MP: Sötétzölddel jelöljük az első cella első bitjének keletkezését. Az első flip-flop 0 állapotban van, 00-val vezéreljük, a DG flip-flop ebben az esetben 0 kimenetet ad.

| $y_1, y_2$ \ $X_1, X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| $y_1, y_2$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ |
| 00 | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | 11, 1 | 11, 1 | 11, 1 | 10, 1 |
| 10 | 11, 1 | 00, 1 | 00, 1 | 10, 1 |

4. lépés: hazárdok vizsgálata:

Kritikus versenyhelyzetek:

| $y_1, y_2$ | $X_1, X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| $y_1, y_2$ | | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ |
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | | 11, 1 | 11, 1 | 11, 1 | 10, 1 |
| 10 | | 11, 1 | 00, 1 | 00, 1 | 10, 1 |

Keressünk kritikus versenyhelyzetet.
A fentebbi kódolt állapottáblában nem találunk ilyet.

Módosítsuk a kódolt állapottáblát, és nézzünk példát kritikus versenyhelyzetre:

| $y_1, y_2$ | $X_1, X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| $y_1, y_2$ | | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ |
| 00 | | 00, 0 | 00, 0 | 00, 0 | 11, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | | 11, 1 | 11, 1 | 11, 1 | 00, 1 |
| 10 | | 11, 1 | 00, 1 | 00, 1 | 10, 1 |

Mindkét eset kritikus versenyhelyzet.

Lényeges hazárd keresése:

| $y_1, y_2$ | $X_1, X_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| $y_1, y_2$ | | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ | $Y_1, Y_2, Z$ |
| 00 | | 00, 0 | 00, 0 | 00, 0 | 01, 0 |
| 01 | | 00, 0 | 11, 0 | 11, 0 | 01, 0 |
| 11 | | 11, 1 | 11, 1 | 11, 1 | 10, 1 |
| 10 | | 11, 1 | 00, 1 | 00, 1 | 10, 1 |

Megvizsgáljuk a stabil állapotokat, és Unger-konfigurációt keresünk.
Lényeges hazárdot találhatunk. A zölddel jelölt állapotba kéne jutnunk normális működés esetén, de a pirossal jelölt állapotba juthatunk hibás működés esetén, amennyiben a sárgával jelölt állapotból indulunk.

Rendszerhazárd vizsgálata:

CSAK szinkron sorrendi hálózatban kereshető.
CSAK akkor fordulhat elő, ha szinkron elemek (flip-flopok, teljes szinkron hálózatok) sorba vannak kötve, és az egyik elem bemenete függ, egy másik elem kimenetétől.

Jelen példában nem lehet rendszerhazárd, mivel nincs órajel, tehát nem szinkron hálózatról van szó.