Széchenyi István Egyetem

dr. Keresztes Péter

# DIGITÁLIS HÁLÓZATOK

TARTALOMJEGYZÉK

**Bevezető** **10**

***1. rész. Kombinációs hálózatok tervezése*** **11**

**1.1. LOGIKAI ÉRTÉKEK ÉS ALAPMŰVELETEK** **11**
**1.1.1. A logikai változók és értékeik** **11**
**1.1.2. A logikai értékek közötti alapműveletek** **11**
**1.1.3. A logikai változók és konstansok közötti azonosságok** **12**

**1.2. A KOMBINÁCIÓS HÁLÓZAT MODELLJE** **13**
**1.2.1. A kombinációs hálózat „fekete.-doboz" modellj** **13**
**1.2.2. Teljesen specifikált és nem teljesen specifikált kombinációs hálózat** **14**

**1.3. LOGIKAI FÜGGVÉNYEK ÉS MEGADÁSI MÓDJAI** **15**
**1.3.1. Logikai függvény megadása igazságtáblázattal** **16**
**1.3.2. Logikai függvény megadása algebrai kifejezéssel. Kanonikus alak** **16**
**1.3.3. Logikai függvény megadása elvi logikai vázlattal** **17**
**1.3.4. A kétváltozós logikai függvények** **18**
**1.3.4.1. A '0' és '1' generátorok** **18**
**1.3.4.2. Az ÉS és a NÉS függvény, illetve kapu** **19**
**1.3.4.3. A VAGY és NVAGY függvény, illetve kapu** **19**
**1.3.4.4. Az ANTIVALENCIA ill EKVIVALENCIA függvény, illetve a XOR és az XNOR kapuk** **19**

**1.4. LOGIKAI FÜGGVÉNYEK MINTERMES ÉS MAXTERMES ALAKJAI** **19**

**1.5. TELJESEN HATÁROZOTT LOGIKAI FÜGGVÉNYEK EGYSZERŰSÍTÉSE** **21**
**1.5.1. Egyszerűsítés algebrai módszerrel** **21**
**1.5.2. Quine módszere** **22**
**1.5.3. Logikai függvények Karnaugh táblás (K-táblás) egyszerűsítése** **24**
**1.5.3.1. Három változós K-tábla** **24**
**1.5.3.2. 4-változós K-tábla** **26**
**1.5.3.3. Összevont termek kiolvasása a K-táblából** **26**

**1.5.3.4. Teljesen határozott logikai függvények egyszerűsítése a K-táblán 28**
**1.5.3.5. Nem teljesen határozott logikai függvények** egyszerűsítése a K-táblán **29**

**1.6. EGYKIMENETŰ KOMBINÁCIÓS HÁLÓZATOK TERVEZÉSE 30**
**1.6.1. Teljesen specifikált, egykimenetű hálózatok tervezése 30**
**1.6.2. Tervezési példa: 31**
**1.6.3. Nem teljesen specifikált, egykimenetű hálózatok tervezése 32**
**1.6.4.Tervezési példa nem teljesen specifikált esetre (1) 32**
**1.6.5.Tervezési példa nem teljesen specifikált esetre (2) 33**

**1.7. TÖBBKIMENETŰ KOMBINÁCIÓS HÁLÓZATOK TERVEZÉSE 35**
**1.7.1. Egy bevezető példa 35**
**1.7.2. Prímimplikáns készlet többkim. kombinációs hálózatok egyszerűsítéséhez 39**

**1.8. HAZÁRDOK 39**
**1.8.1. A statikus hazárd keletkezése 39**
**1.8.2. A statikus hazárd jelenségek pontos meghatározása 40**
**1.8.3. A statikus hazárdok kiküszöbölése 40**
**1.8.4. Dinamikus hazárd 41**
**1.8.5. Funkcionális hazárd 41**
**TERVEZÉSI FELADATOK 1. 41**

***2. Rész. sorrendi hálózatok tervezése* 42**

**2.1 ELEMI SORRENDI HÁLÓZATOK, TÁROLÓK. 42**
**2.1.1. S-R tároló működése és igazság-táblái 42**
**2.1.2. Az S-R tároló állapot-átmeneti táblája 44**
**2.1.3. K-tábla az S-R tároló megvalósítására 44**
**2.1.4. Az S-R tároló realizációi 45**
**2.1.5. Kísérlet J-K tároló megvalósítására 46**
**2.1.6. A kísérleti J-K tároló állapot-átmeneti táblája 47**
**2.1.7. D-G tároló 47**
**2.1.8. D-G tároló egy ekvivalens alakja 50**
**2.1.9. A D-G tároló, mint memória-elem 50**
**2.1.10. Többszörös bemeneti váltás hatásai a D-G tárolóra 50**

**2.1.11. A D-G tároló „átlátszósága” 51**

**2.2. MESTER-SZOLGA TÁROLÓK (FLIP-FLOPOK) 51**
**2.2.1. A D-MESTER-SZOLGA tároló (flip-flop) 51**
**2.2.2. A D-MS tároló kétfázisú órajellel 53**
**2.2.3. Az élvezérelt D-MS tároló 54**
**2.2.4. A J-K –MS tároló 54**
**2.2.5. Flip-flopok segéd-bemenetei és szimbólumaik 56**
**TERVEZÉSI FELADATOK 2. 57**

**2.3. SORRENDI HÁLÓZATOK MODELLJEI, ALAPTÍPUSAI 58**
**2.3.1. A kombinációs hálózatok egy magasabb szintű modellje 59**
**2.3.2. A sorrendi (szekvenciális) hálózatok általános modelljei 60**
**2.3.3. Mealy-típusú sorrendi hálózat. 62**
**2.3.4. Moore-típusú sorrendi hálózat 63**
**2.3.5. Aszinkron sorrendi hálózat 64**
**2.3.5.1. Közvetlen visszacsatolású aszinkron sorrendi hálózat 64**
**2.3.5.2. S-R tárolókkal visszacsatolt aszinkron sorrendi hálózat 65**
**2.3.6. Szinkron sorrendi hálózat 66**
**2.3.6.1. D-MS flip-flopokkal visszacsatolt szinkron hálózat 66**
**2.3.6.2. J-K-MS flip-flopokkal visszacsatolt szinkron sorrendi hálózat 67**

**2.4. SZINKRON HÁLÓZAT TERVEZÉSI FOLYAMAT MINTA-PÉLDÁKON 69**
**2.4.1. Az első szinkron hálózattervezési minta-feladat 69**
**2.4.1.1. A minta-feladat megfogalmazása 69**
**2.4.1.2. Egy MEALY modell felvázolása állapot-átmeneti gráffal és/vagy előzetes állapot-átmeneti táblával 69**
**2.4.1.2.1. Állapot-átmeneti gráf 69**
**2.4.1.2.2. Előzetes, szimbolikus állapottábla 69**
**2.4.1.2.3. A feladat állapotgráfja és előzetes, szimbolikus állapottáblája 70**
**2.4.1.3. Bemeneti egyszerűsítési lehetőségek kihasználása 71**
**2.4.1.4. A feltétlenül szükséges számú állapot megállapítása 71**
**2.4.1.5. Állapot-összevonás az adott feladatban 71**
**2.4.1.6. Az összevont állapottábla 72**
**2.4.1.7. A kódolt állapottábla 72**
**2.4.1.8. A vezérlési tábla 72**
**2.4.1.9. A feladat összevont szimbolikus és kódolt állapottáblája valamint J-K flip -flopokkal történő realizációjának**

vezérlési táblája 74
2.4.1.10. Az $f_y$ és $f_z$ hálózatok tervezése K-táblák segítségével 74
2.4.1.11. A feladat megoldására szolgáló hálózat K táblái és lefedésük 74
2.4.1.12. Realizáció 75
2.4.1.13. A feladat megoldásának realizációja 76
2.4.1.14. A feladat megoldása Moore-típusú hálózattal 76

2.5. ASZINKRON HÁLÓZAT TERVEZÉSI FOLYAMAT MINTA-PÉLDÁKON 79
2.5.1. Az első aszinkron hálózat tervezési mintafeladat 79
2.5.1.1. Időzítési diagram és előzetes szimbolikus állapottábla 79
2.5.1.2. A feladat időzítési diagramja és előzetes szimbolikus állapottáblája 80
2.5.1.3. Állapot-összevonás 81
2.5.1.4. Állapot-összevonás a feladat állapottábláján 81
2.5.1.5. Állapot-kódolás, a kódolt állapottábla felvétele 81
2.5.1.6. A feladat állapotainak kódolása és kódolt állapottáblája 81
2.5.1.7. Analízis a kritikus versenyhelyzetek felderítésére 82
2.5.1.8. Kritikus versenyhelyzetek a feladat kódolt állapottáblájának vizsgálatával 83
2.5.1.9. Kritikus versenyhelyzetek kiküszöbölése állapot-átkódolással 84
2.5.1.10. A feladat állapot-kódjának megváltoztatása a kritikus versenyhelyzetek kiküszöbölésére 84
2.5.1.11. A realizáció K-táblái és lefedésük 84
2.5.1.12. Az első aszinkron feladat realizációja 85
2.5.1.13. Aszinkron hálózatok beállítása kezdeti állapotba 85
2.5.1.14. Az első aszinkron minta-feladat realizációjának kiegészítése kezdeti állapotba kényszerítő, R-logikával 85
2.5.2. A második aszinkron hálózat tervezési mintafeladat 87
2.5.2.1. Előzetes, szimbolikus állapottábla 87
2.5.2.2. Összevont, szimbolikus állapottábla 88
2.5.2.3. A második aszinkron minta-feladat kódolt állapottáblája 89
2.5.2.4. A második aszinkron mintafeladat $f_y$ és $f_z$ függvényeinek lefedése 90
2.5.2.5. A speciális, sorrendi ÉS kapu realizációja R-logika nélkül 90
2.5.2.6. A speciális, sorrendi ÉS kapu realizációja R-logikával 91
2.5.2.7. A második aszinkron feladat realizációja S-R tárolókkal 91
2.5.2.8. Lényeges hazárdok aszinkron hálózatokban 93

**2.6. SORRENDI HÁLÓZATOK TERVEZÉSÉNEK FOLYAMATA 94**
**2.6.1. Szinkron szekvenciális hálózatok tervezésének fő lépései 94**
**2.6.2. Aszinkron szekvenciális hálózatok tervezésének fő lépései 94**

**2.7. SORRENDI HÁLÓZATOK KEZDETI ÁLLAPOTÁNAK BEÁLLÍTÁSA 95**
**2.7.1. Szinkron sorrendi hálózatok kezdeti állapotának beállítása 95**
**2.7.1.1. Beállítás a PRESET (Pr) és a CLEAR (Cl) bemenetek kihasználásával 95**
**2.7.1.2. Beállítás az $f_y$ hálózat kiegészítésével 96**
**2.7.1.2.1. Kiegészítő hálózat D tárolók esetén 96**
**2.7.1.2.2. Kiegészítő hálózat J-K tárolók esetén 97**
**2.7.2. Aszinkron hálózatok kezdeti állapotának beállítása 98**
**2.7.2.1. Közvetlenül visszacsatolt kombinációs hálózattal megvalósított aszinkron hálózat kezdeti állapotának beállítása 98**
**2.7.2.2. S-R tárolókkal visszacsatolt aszinkron hálózatok kezdeti állapotának beállítása 99**
**TERVEZÉSI FEADATOK 3. 100**

**2.8. ÁLLAPOT-ÖSSZEVONÁSI MÓDSZEREK 102**
**2.8.1. Állapot-összevonás teljesen specifikált szimbolikus előzetes állapottáblán 102**
**2.8.1.1. Az összevonhatóság feltétele 102**
**2.8.1.2. A nem-megkülönböztethetőség, mint bináris reláció 103**
**2.8.1.3. Lépcsős tábla az állapotok páronkénti vizsgálatára 104**
**2.8.1.4. Mintapélda megoldása lépcsős táblán 105**
**2.8.1.5. A mintapélda összevont állapottáblájának szerkesztése 108**
**2.8.2. Állapot-összevonás nem teljesen specifikált szimb. előzetes állapottáblán 108**
**2.8.2.1. Állapot-kompatibilitás nem teljesen specifikált állapottáblán 108**
**2.8.2.2. A kompatibilitási osztályok *zárt* halmaza 109**
**2.8.2.3. Kevesebb, vagy kisebb elemszámú zárt kompatibilitású osztály keresése 110**
**2.8.2.4. Az összevont állapottábla szerkesztése 111**
**2.8.2.5. Példa NTSH állapottáblázaton történő állapot-összevonásra 111**
**2.8.3. Összefoglalás az állapot-összevonási módszerekről 113**

**2.9. ÁLLAPOT-KÓDOLÁSI MÓDSZEREK 113**

**2.9.1. Szinkron hálózatok állapot-kódolási módszerei 113**
**2.9.1.1. A szomszédos állapot-kódok választásának elvei 113**
**2.9.1.2. A szekunder változók csoportosítása: Önfüggő csoport keresése 116**
**2.9.1.2.1. A szekunder változók csoportosítása alapján definiált ekvivalencia reláció és a komponensekhez rendelt $\Pi_i$ partíció 117**
**2.9.1.2.2. A komponensekhez tartozó környezeti partíció 118**
**2.9.1.2.3. A komponensekhez rendelt partíció-párok 118**
**2.9.1.2.4. Helyettesítési tulajdonságú partíció 119**
**2.9.1.2.5. Egy HT-partíciós állapotkódolási példa 120**
**2.9.1.3. Szinkron hálózatok 1-es súlyú állapotkódokkal 124**
**2.9.2. Aszinkron sorrendi hálózatok állapotkódolása 125**
**2.9.2.1. Instabil állapotok beillesztése a kritikus versenyhelyzetek kiküszöbölésére 125**
**2.9.2.2. Tracey és Unger módszere a kritikus versenyhelyzetek kiküszöbölésére 127**
**TERVEZÉSI FELADATOK 4. 129**

***3. Rész. Összetett digitális-hálózati egységek* 132**

**3.1. MULTIPLEXEREK, DEMULTIPLEXEREK**
**3.1.1. Egykimenetű, 4 bemenetű MPX 133**
**3.1.2. Bővítés a bemeneti adatok számának növelése céljából 133**
**3.1.3. Bővítés sínek közötti választás céljából (BUS-MPX) 134**
**3.1.4. A multiplexerek felépítése 135**
**3.1.5. A multiplexer, mint vezérelhető logikai hálózat 135**
**3.1.6. Demultiplexerek 136**
**3.1.7. Dekóderek 137**
**3.1.8. Multiplexerek és demultiplexerek CMOS átvivő kapukkal 137**
**3.2. REGISZTEREK 138**
**3.2.1. Szintvezérelt statikus regiszter modell 138**
**3.2.2. Szintvezérelt regiszter modell ponált és negált beírójellel 139**
**3.2.3. Kvázistatikus regiszter modell 139**
**3.2.4. Élvezérelt regiszter modell 140**

**3.3. SOROS ELÉRÉSŰ TÁROLÓK 140**
**3.3.1. 1-bites, párhuzamosan is betölthető soros memóriák 142**
**3.3.2. Gyűrűs számláló 142**
**3.3.3. Johnson számláló 143**

**3.3.4. Véletlenszám-generátor 144**
**3.3.5. Szószervezésű soros memóriák 144**
**3.3.6. FIFO memóriák 145**
**3.3.7. LIFO memóriák 146**

**3.4. SZÁMLÁLÓK 147**
**3.4.1. A MESTER-SZOLGA J-K FLIP-FLOP, mint a számlálók alapeleme. 147**
**3.4.2. Szinkron számlálók modellje 148**
**3.4.3. Adott modulusú számláló átalakítása más modulusú számlálóvá 149**
**3.4.4. Számláló nullától különböző kezdő-értékének beállítása 150**
**3.4.5. Szinkron számlálók kaszkádosítása 151**
**3.4.6. Példa : Modulo-16 szinkron számlálók kaszkádosítása modulo-256 számlálóvá 152**
**3.4.7. Példa : Modulo-8 és modulo-16-os szinkron számlálók kaszkádosítása modulo- 128 számlálóvá 153**
**3.4.8. Aszinkron számlálók 154**
**3.4.9. Aszinkron számlálók kaszkádosítása 155**

**3.5. FUNKCIÓS EGYSÉGEK 155**
**3.5.1. Komparátorok 157**
**3.5.2. Összeadók 159**
**3.5.2.1 A teljes összeadó (1-bites összeadó) 159**
**3.5.2.2. Soros átvitel-képzésű bit-vektor összeadó 161**
**3.5.2.3. Párhuzamos átvitelképzésű bit-vektor összeadó 161**
**3.5.3. Kivonás 162**
**3.5.4. Szorzók 165**

**3.6. VEZÉRLŐ EGYSÉGEK 166**
**3.6.1. Az adatstruktúrára (DATA-PATH) és vezérlőegységre (TIMING AND CONTROL) való felbontás (dekompzíció) 168**
**3.6.2. Számláló-típusú vezérlők 168**
**3.6.3. Példa számláló típusú vezérlő-egység tervezésére 170**
**3.6.3.1 A vezérlési folyamat ütemezése, azaz egy vezérlési szekvencia leírás elkészítése 171**
**3.6.3.2 A multiplexerek megtervezése a vezérlési szekvencia-leírásból 172**
**3.6.3.3 A vezérlőjel-generátorok tervezése 172**
**3.6.4. FSM típusú vezérlők 174**
**3.6.5. Önálló makro-cella tervezése FSM típusú vezérlővel 175**
**3.6.6. Vezérlés mikroprogramozással 177**

**3.6.6.1 A mikroprogram utasításszó felépítése 178**
**3.6.6.2 A mikroprogramozott vezérlő egység időbeli működése 179**

**3.7. BEVEZETÉS A MIKROPROCESSZOROS RENDSZEREK TERVEZÉSÉBE 180**
**3.7.1. A Neumann-architektúra 180**
**3.7.2. A CPU címzési módjai 181**
**3.7.3. Utasítások, adatok 182**
**3.7.4. Szekvenciális program 182**
**3.7. 5. Egy jellegzetes egyszerű mikroprocesszor architektúra 183**
**3.7.6. A processzor időbeli működése (IDŐZÍTÉS, TIMING) 186**
**3.7.7. Az utasítás-készlet 186**
**3.7.8. Néhány kiragadott utasítás és végrehajtása 186**
**3.7.8.1 A 'MOVr,M' ( Move from Memory) utasítás végrehajtása 187**
**3.7.8.2. Az 'ADD M' ( Add Memory) utasítás végrehajtása 187**
**3.7.8.3. A 'LDA' ( Load Accumulator ) utasítás végrehajtása 188**
**3.7.8.4. A 'CALL' ( Call, azaz alprogram hívás) utasítás végrehajtása 189**
**3.7.8.5. A 'RETURN' (Return, azaz visszatérés az alprogramból) utasítás végrehajtása 191**
**3.7.9. A processzor működésének egyéb sajátosságai 192**
**3.7.9.1. READY-WAIT 192**
**3.7.9.2. A státusz- információ 192**
**3.7.9.3. A legfontosabb jelzőbitek 192**

**3.7.9.4. SP értékének beállítása speciális utasítással 193**
**3.7.9.5. Megszakítás-kezelés 193**
**3.7.10. A mikroprocesszoros rendszer 194**
**3.7.11. A mikroprocesszoros rendszerek ASSEMBLY szintű programozása 195**
**3.7.12 A még nem ismert utasítások definíciói 197**
**TERVEZÉSI FELADATOK 5. 200**

**Ajánlott irodalom 204**

## *BEVEZETŐ*

A jegyzet a Széchenyi István Egyetem *villamosmérnöki*, *mérnök-informatikus* és *mechatronikus* egyetemi alapszakok hallgatói számára, a címmel azonos elnevezésű tárgy tanulmányozásának segítésére készült. Tartalmazza a digitális hálózatok legfontosabb építőelemeinek leírását és az azokkal való tervezés legfontosabb módszereit és eljárásait. A jegyzet felsőfokú előtanulmányok nélkül is eredményesen tanulmányozható. Ugyanakkor megalapozza a fenti szakokon az erre épülő tárgyakat.

Hangsúlyozni kell, hogy a jegyzet célja nem az logikai rendszerek leírási módszereinek elmélyült megalapozása, hanem a mérnöki alkotások szempontjából elengedhetetlen tervezési eljárások megismertetése és elsajátíttatása.

A fejezetekhez közvetlenül kapcsolódnak *kérdések és egyszerű feladatok*. A nagyobb egységeket *tervezési feladatok* zárják le. Ugyanakkor a megértést és a tervezési módszerek elsajátítását - részletesen bemutatott megoldással - sok feladat segíti.

# 1.5. LOGIKAI ÉRTÉKEK ÉS ALAPMŰVELETEK

## 1.1.1. A logikai változók és értékeik

Több olyan megfigyelhető objektum van, amelynek mindössze két értéke létezik. Ilyenek például a *logikai állítások,* amelyek *hamisak vagy igazak*, az *események*, amelyek *nem következnek be*, vagy *bekövetkeznek*, a *kapcsolók*, amelyek *nyitottak vagy zártak*. A számítástechnikában az ilyen típusú változókat a nagy angol matematikusról Boole-féle változóknak (*boolean*) nevezzük, lehetséges értékeik a ***'false'*** és a ***'true'***. A modern digitális áramkörök technikájában a jelekhez rendelt vezetékeknek mindössze kétféle feszültségszintje lehet, egy a zérus szinthez igen közeli alacsony (***low***) és egy néhány volt nagyságú magas (***high***) szint. Az egyszerű ábrázolás kedvéért az egyik feszültségszinthez a '***0***', a másikhoz az '***1***' számot rendeljük. Leggyakrabban a '***0***' –t az alacsony, az '***1***'-t a magas szinthez rendelik. A digitális technikában ezeket a változókat, illetve jel-vezetékeket logikai változóknak nevezzük. A '***0***' és az '***1***' pedig logikai értékek.

### 1.1.2. A logikai értékek közötti alapműveletek

A logikai értékek között három logikai alapműveletet definiáltak.
Ezek : A *logikai szorzás vagy konjunkció*, (***ÉS***) egy szorzás műveleti jellel ( • ), a *logikai összeadás vagy diszjunkció*, (***VAGY***) az összeadás műveleti jelével, (+), amelyek két operandusú, úgynevezett *bináris* műveletek, és a *logikai tagadás vagy negáció* (felül-vonás) mely csak egy-operandusú, úgynevezett *unáris* művelet. A definíciókat a következő táblázatokkal adjuk meg:

| '•' | ' + ' |
|---|---|
| 0 • 0 = 0 | 0 + 0 = 0 |
| 0 • 1 = 0 | 0 + 1 = 1 |
| 1 • 0 = 0 | 1 + 0 = 1 |
| 1 • 1 = 1 | 1 + 1 = 1 |

**Negáció:**

$\overline{0} = 1, \qquad \overline{1} = 0$

### 1.1.3. A logikai változók és konstansok közötti azonosságok

Az azonosságok olyan igazságok, amelyek a változók minden lehetséges értékére érvényesek. Bizonyításuk igen egyszerű, ha a változókat minden lehetséges értékkel helyettesítjük, és ellenőrizzük, teljesülnek-e a műveleti táblák előírásai. Lássuk a legfontosabb azonosságokat Az első csoportban változók és konstansok (értékek), majd változók és változók között.

$$A \cdot 0 = 0 \qquad A + 0 = A$$
$$A \cdot 1 = A \qquad A + 1 = 1$$
$$A \cdot \overline{A} = 0 \qquad A + \overline{A} = 1$$

*ASSZOCIATIVITÁS:*

$$A + (B + C) = (A + B) + C$$
$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

*DISZTRIBUTIVITÁS:*

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$
$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

*A DE-MORGAN FÉLE AZONOSSÁGOK*

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$
$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

Adjunk értelmezést például az események körében az

$$A \cdot 1 = A$$

azonosságnak! Eszerint ***A*** egy két-kimenetelű esemény, ***1*** a biztosan bekövetkező esemény, a kettő szorzata pedig egy olyan összetett esemény, amelynek mind az ***A***, mind az ***1***-vel jellemzett esemény bekövetkezése a szükséges és elégséges feltétel. Az azonosság szerint az összetett esemény bekövetkezése az ***A*** esemény bekövetkezésével azonos értékű.

Különös jelentőséggel bír a két *De-Morgan* azonosság. Ezek közül a másodikat értelmezzük a *kapcsolók* körében. Legyen ***A*** és ***B*** két kapcsoló. A kettő *ÉS* kapcsolata egy *sorosan kapcsolt* kapcsolópárt reprezentál, amely csak akkor vezethet, ha mind az ***A***, mind a ***B*** kapcsoló vezető állapotban van. A szorzat tagadása arra az állapotra utal, amikor az összetett kapcsoló nem

vezet. Az azonosság jobb oldala megadja, hogy ez azzal egyenlő értékű, hogy vagy az ***A*** van *nem vezető állásban*, vagy a ***B*** van *nem vezető állásban*, vagy mindkettő *nem vezető állásban* van.

*EGYSZERŰ FELADATOK*
*1. Értelmezzük a logikai állítások körében az*

$$A + \overline{A} = 1$$

*azonosságot!*

*2. Értelmezzük a kapcsolók körében az első De-Morgan azonosságot!*

## 1.2. A KOMBINÁCIÓS HÁLÓZAT MODELLJE

### 1.2.1. A kombinációs hálózat „fekete.-doboz" modellje

A kombinációs hálózat fekete-doboz modelljét az *1.1 ábrán* mutatjuk be. A kombinációs hálózatnak bemenetei és kimenetei vannak, valamennyi egy-egy logikai változó, illetve logikai jel, és ennek megfelelően mindegyik csak a ***0***-t, vagy az ***1***-t veheti fel értékként. Ez a kombinációs hálózat fekete-doboz modelljének egyik lényeges tulajdonsága.
*A bemeneteket az* ***x1, x2,. . . .xi, . . .xn****, , a kimeneteket* ***y1, y2, . . .yj, . . .ym*** *szimbólumokkal jelöltük.*
A *kombinációs hálózat* a bemeneti jelek felett *értelmezett bemeneti érték-variációkhoz a kimeneti jelek érték-variációit rendeli.* (Minden bemenetihez legfeljebb csak egy kimenetit)
Például ***n = 3*** esetben a bemeneti variációk :

***(0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1)***


*1.1. ábra. A kombinációs hálózat „fekete-doboz" modellje.*

ha ***m = 2***, akkor a kimeneti variációk halmaza :

***(0 0, 0 1, 1 0, 1 1)***

Jellemezzük az ilyen bemenetekkel és kimenetekkel rendelkező kombinációs hálózatot a következő táblázattal:

| ***Bemeneti variációk*** | | | ***Kimeneti variációk*** | |
|---|---|---|---|---|
| ***X1*** | ***X2*** | ***X3*** | ***Y1*** | ***Y2*** |
| ***0*** | ***0*** | ***0*** | ***0*** | ***1*** |
| ***0*** | ***0*** | ***1*** | ***1*** | ***0*** |
| ***0*** | ***1*** | ***0*** | ***0*** | ***1*** |
| ***0*** | ***1*** | ***1*** | ***1*** | ***1*** |
| ***1*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***1*** | ***0*** | ***1*** | ***0*** | ***1*** |
| ***1*** | ***1*** | ***0*** | ***1*** | ***1*** |
| ***1*** | ***1*** | ***1*** | ***1*** | ***0*** |

Megjegyezzük, hogy a *bemeneti értékvariáció* helyett gyakran, általánosan elfogadott, de pongyola módon *bemeneti kombinációt*, illetve a *kimeneti érték-variáció* helyett *kimeneti-kombinációt* mondanak. Pedig a szóban forgó fogalom a kombinatorikában *variáció*. Mivel ez a pongyola terminológia szakmai körökben is elterjedt, következőkben mi is használni fogjuk
Az előzőkből következik a kombinációs hálózat fekete-doboz modelljének másik lényeges tulajdonsága, nevezetesen, hogy *adott bemeneti kombinációra a hálózat mindig ugyanazt a kimeneti kombinációt* szolgáltatja.

### 1.2.2. Teljesen specifikált és nem teljesen specifikált kombinációs hálózat

*Teljesen specifikált* a kombinációs hálózat, ha *minden bemeneti variációhoz olyan kimeneti variáció tartozik, amelyben minden kimenet értéke specifikálva van.*

A kombinációs hálózat *nem teljesen specifikált, ha van legalább egy olyan bemeneti variáció, amelyhez rendelt kimeneti variációkban legalább egy változó értéke közömbös.*

Keressünk választ a következő kérdésre:
Hány ***n*** bemenetű és ***m*** kimenetű *teljesen specifikált* kombinációs hálózat létezik?
A választ kombinatorikai módszerrel adhatjuk meg;
Mivel ***n*** bemeneti jelhez $\mathbf{(2)^n}$ érték-variáció tartozik, ***m*** kimeneti jelhez pedig $\mathbf{(2)^m}$ érték-variáció tartozik, *annyi teljesen specifikált **n** bemenetű és **m** kimenetű különböző hálózat van, ahányszor a* $\mathbf{(2)^m}$ számú kimeneti érték-variációból ismétléssel ki tudunk választani egy $\mathbf{(2)^n}$ hosszúságú sorozatot. Ez pedig :

$$(2^m)^{2^n}$$

*EGYSZERŰ FELADAT*
*1. Hány teljesen specifikált két-bemenetű, egy-kimenetű kombinációs hálózat van?*

## 1.3. LOGIKAI FÜGGVÉNYEK ÉS MEGADÁSI MÓDJAIK

A kombinációs hálózat (KH) *minden egyes kimenetére megadhatjuk, hogy a bemeneti jelek mely variációira lesz az adott kimenet '**1**', és mely bemeneti variációkra lesz a kimenet '**0**'*. Ha van olyan bemeneti variáció, *amelyre nincs előírásunk, (*sem '***1***' sem '***0***', hanem 'mindegy', azaz „don't care"), akkor *a hálózat nem teljesen specifikált.* A KH minden egyes kimenetéhez egy ***n***-változós *logikai függvény tartozik.*

### 1.3.1. Logikai függvény megadása igazságtáblázattal

*Egy logikai függvényt igazság-táblázattal úgy adunk meg, hogy minden bemeneti variációt felsorolunk, és megadjuk a hozzájuk rendelt függvényértéket, azaz az ('**1**', '**0**', **'-'**) hármas valamelyikét.* Az ilyen táblázatot a függvény igazság-táblázatának nevezzük.
Az igazságtáblával való megadásra álljon itt egy példa:

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

### 1.3.2. Logikai függvény megadása algebrai kifejezéssel. Kanonikus alak.

Az algebrai alakot az igazságtáblázatból olvashatjuk ki. Azokhoz a bemeneti variációkhoz, amelyekhez ***1***-s kimeneti érték tartozik, egy *logikai szorzatot* rendelünk. A szorzat tényezői a változók *ponált*, vagy *negált* változatai. *Ponált alak maga a változó neve*, a negált változat fogalmát pedig már ismerjük. A szorzat változója

- ponált, ha a bemeneti variációban '***1***' szerepel az oszlopában,
- negált, ha a bemeneti variációban '***0***' szerepel az oszlopában.

Az így felírt bemeneti variációkat logikailag összeadjuk.

Adjuk meg az előző pontban definiált 3 változós függvény algebrai alakját!

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

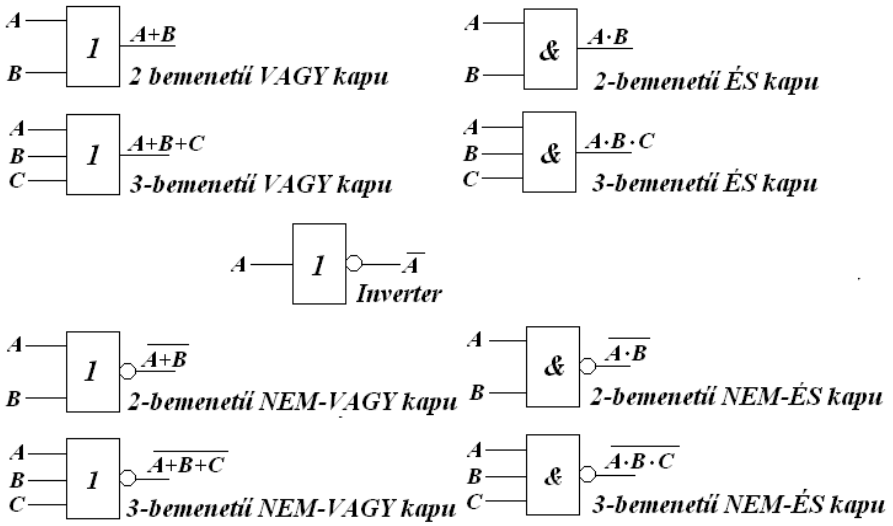
Megjegyezzük, hogy a logikai-algebrában, akár csak a klasszikusban, a szorzás jelét felesleges leírni a szorzandók közé.

Azokat a logikai szorzatokat (termeket), amelyekben a függvény valamennyi változója szerepel, *mintermeknek* nevezzük. A logikai függvény megadásának ezt a módját, azaz azon mintermek összegét, amelyekhez a függvény ***1***-et rendel, *mintermes kanonikus normál alaknak* nevezzük.

### 1.3.3. Logikai függvény megadása elvi logikai vázlattal

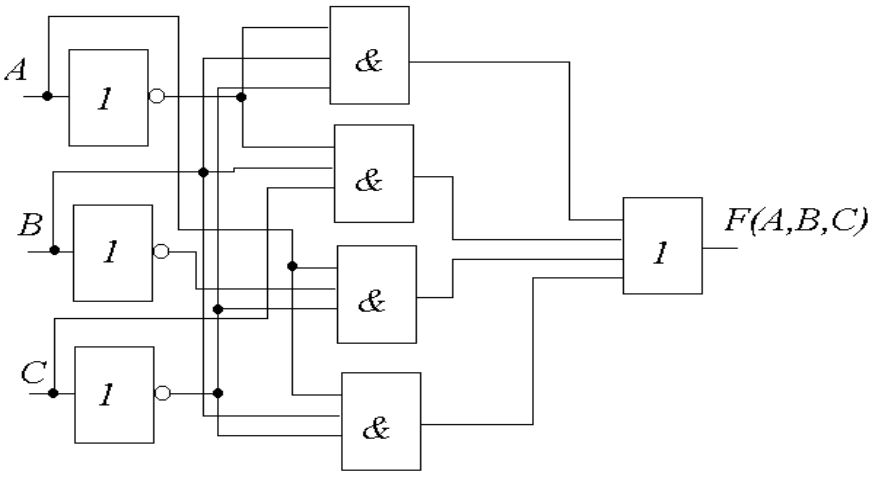
Igen gyakori megadási módszer, hogy a kanonikus alakot grafikus formában adjuk meg. A logikai műveleteket ilyenkor logikai szimbólumok reprezentálják. A negálás műveletét *INVERTEREK,* a szorzattermeket *ÉS*-szimbólumok, illetve *ÉS*-kapuk, az összegzést *VAGY* szimbólumok, illetve *VAGY*-kapuk reprezentálják.

A későbbiekben gyakran fogjuk használni azokat a kapu-szimbólumokat, amelyek a *VAGY* és az *ÉS* kapuktól abban különböznek, hogy a kimenetük azok logikai negáltjai. Tehát a kétváltozós *NEM-VAGY* kapu kimenete akkor ***0,*** ha a bemenetek valamelyike, vagy mindkettő ***1***, a kétváltozós *NEM-ÉS* kapu kimenete pedig akkor ***0***, ha mindkét bemenet értéke ***1***. Könnyen belátható, hogy egy két-bemenetű *NEM-ÉS* egyik bemenetéről *INVERTER*-ként működik, ha a másik bemenetére állandó ***1***-est kapcsolunk, a *NEM-VAGY* pedig akkor, ha a másik bemenetére állandó ***0***-t kapcsolunk.
Az *1.2 ábrán* megmutatjuk az elvi logikai vázlatok kapukészletének egy részét. Rajzolással nagyobb bemenetszámú kapu-szimbólumokat könnyen alkothatunk, de tudnunk kell, hogy nem mindegyik általunk rajzolt kapuszimbólumhoz tartozik gyártott logikai kapu.




*1.2. ábra. A logikai szimbólumok*

Az *1.3 ábrán* mutatjuk meg példánk inverterekkel, *ÉS,* valamint *VAGY* kapu-szimbólumokkal megadott elvi logikai vázlatát.




*1.3. ábra. A példa szerint logikai hálózat megadása grafikus szimbólumokkal*

### 1.3.4. A kétváltozós logikai függvények

A következő táblázatban megadjuk az összes kétváltozós logikai függvény definícióját.

| bemenet | | függvényértékek | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **x1** | **x2** | **f0** | **f1** | **f2** | **f3** | **f4** | **f5** | **f6** | **f7** | **f8** | **f9** | **f10** | **f11** | **f12** | **f13** | **f14** | **f15** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Láthatjuk, hogy *16* kétváltozós logikai függvény van. Ezek közül áttekintjük a nevezetesebbeket, illetve azokat, amelyeket gyakran használunk logikai hálózatok építése során, mint kapu-áramköröket.

#### 1.3.4.1. A ’0’ és ’1’ generátorok

Az ***f0*** értéke állandóan ***0,*** nem érzékeny a bemenetekre. Negáltja az ***f15***, amelynek értéke állandóan ***1 .***

### 1.3.4.2. Az ÉS és a NÉS függvény, illetve kapu

Az ***f1*** és az ***f14*** értékei éppen egymás negáltjai. Az ***f1*** nevezetes, csak akkor szolgáltat ***1***-et, ha mindkét bemeneti változó értéke ***1***, az ***f14*** pedig ekkor éppen ***0***-t szolgáltat**.** Az előbbi neve ***ÉS***, az utóbbié ***NEM-ÉS***, röviden ***NÉS***.
*Az algebrai alakok:*

*FELADAT : Bizonyítsuk be algebrai módszerrel, hogy az **f1** függvény negáltja azonos az **f14** függvénnyel!*

### 1.3.4.3. A VAGY és az NVAGY függvény, illetve kapu

Az ***f7*** és az ***f8*** függvények ugyancsak egymás negáltjai. Az ***f7*** értéke ***1***, ha legalább ez egyik bemenet ***1***. Az ***f8*** értéke éppen ilyenkor ***0.*** Az előbbi a ***VAGY***, az utóbbi a ***NEM-VAGY***, (***N-VAGY***) függvény, illetve kapu.

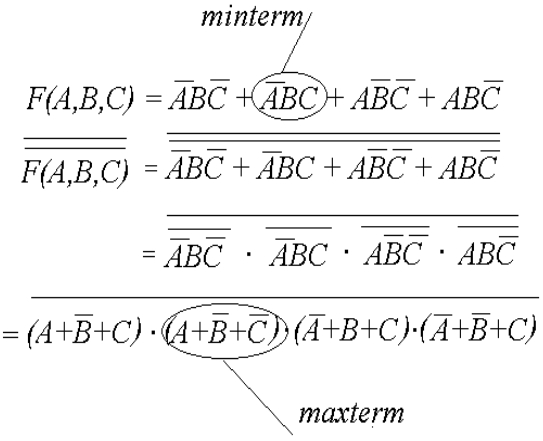
### 1.3.4.4. Az ANTIVALENCIA és az EKVIVALENCIA függvények, illetve kapuk.

Ha a kétváltozós függvény csak abban az esezben szolgáltat 1-t, ha a két bemenet különböző, akkor a függvény neve : ***ANTIVALENCIA***, vagy ***KIZÁRÓ-VAGY***. Ez az ***f6*** függvény. Negáltja az ***f9***, éppen ezekben az esetekben ***0***, és egyezés esetén ***1***. Ez utóbbi neve ***EKVIVALENCIA***, vagy ***KIZÁRÓ-NEM-VAGY***. Az ezeknek megfelelő kapu áramköröket igen gyakran használjuk. A KIZÁRÓ-VAGY művelet jelölésére gyakran használják a ’ $\oplus$ ’ szimbólumot.

*EGYSZERŰ FELADAT : Bizonyítsuk be algebrai módszerrel, hogy az **f6** függvény negáltja azonos az **f9** függvénnyel!.*

## 1.4. LOGIKAI FÜGGVÉNYEK MINTERMES ÉS MAXTERMES ALAKJAI

Az előző pontban megismerkedtünk a teljesen határozott logikai függvény mintermes kanonikus normál alakjával. Létezik egy másik kanonikus alak is, amelyet a mintermes alakból kétszeri tagadással, kétszeres negálással származtathatunk. A már ismert példánkon bemutatjuk, hogy a *De-Morgan* azonosság alkalmazásával hogyan kapjuk a másik kanonikus alakot.




Azt kaptuk, hogy a mintermekkel, azaz logikai *szorzatok összegével* adott eredeti függvény negáltja felírható olyan logikai *összegek szorzataként*, amelyekben ugyancsak minden változó szerepel. Ezeket az összegeket *maxtermeknek* nevezzük. Vegyük nyilvántartásba a kapott maxtermeket a következő módon:

1. Rendeljünk a ponált változókhoz ***1***-t, a negált változókhoz ***0***-t.
2. Rendeljük sorszámként ezekhez a maxtermekhez a kapott bináris számokhoz tartozó decimális számokat. Tegyük ugyanezt a mintermekkel is.

A mintermeket kis ***m*** betűvel, egy felső és egy alsó index-számmal jelöljük. A felső index a változók számát adja meg, az alsó az előbb kiszámított sorszám. Hasonlóan, nagy ***M*** betűvel jelöljük a maxtermeket, ugyanolyan értelmű indexekkel. Például:

$$m_2^3 = \overline{A}\,B\,\overline{C}, \qquad M_5^3 = A + \overline{B} + C$$

Belátható, hogy a fenti sorszámozással és jelölési rendszerben érvényesek a következő transzformációs szabályok:

1. $\overline{m_i^n} = M_{(2^n - 1-i)}$

2. $f(x_1, x_2, \ldots x_n) = m_i^n + m_j^n + \ldots + m_k^n =$

$$= \overline{M_{(2^n - 1-i)} \cdot M_{(2^n-1-j)} \cdot \ \ldots \ \cdot M_{(2^n-1-k)}}$$

## 1.5 TELJESEN HATÁROZOTT LOGIKAI FÜGGVÉNYEK EGYSZERŰSÍTÉSE

A teljesen határozott logikai függvények egyszerűsítési lehetőségeinek vizsgálatára és egyszerűsítésére négy alapvető módszert mutatunk be. Ezek:

1. *Egyszerűsítés algebrai módszerrel*
2. *Quine módszere*
3. *A Karnaugh táblás módszer*
4. *A Quine-McCluskey módszer*

### 1.5.1 Egyszerűsítés algebrai módszerrel

Egyszerűsítsük a bemutatott azonosságokat kihasználva a már megismert háromváltozós logikai függvényünket! A disztributivitást kifejező azonosságok arra mutatnak, hogy a klasszikusan kiemelésnek nevezett átalakítás itt is végrehajtható. A logikai összeg első két szorzat-termjéből és a második két szorzat-termjéből is kiemelhető egy-egy kétváltozós szorzat. Ezután felismerhetjük a zárójelben lévő összegek '***1***' értékét, valamint azt,

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

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

$$= \overline{A}B + A\overline{C}$$

hogy a '***1***'-vel való beszorzást nem kell feltüntetni. Az eredmény kifejezés tehát azonos a kiindulásival, de jóval egyszerűbb annál.

*EGYSZERŰ FELADAT*
*1. Mutassuk meg, hogy igazak a következő állítások:*

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

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

*A bizonyítás elvégezhető a megismert azonosságok felhasználásával, de úgy is eljárhatunk, mint az azonosságok érvényességének belátásakor!*

### 1.5.2 Quine módszere

A Quine módszer alapja a függvény ***1***-es értékéhez rendelt két minterm közös szorzótényezőinek olymódon történő kiemelése, hogy a zárójelben egy logikai változónak és negáltjának az összege maradjon, amely logikai összeg ***1***.

| *mintermek* | *I.* | *II.* | *III.* |
|---|---|---|---|
| *2* | $\overline{A}B\overline{C}\checkmark$ | *(2, 3)* $\overline{A}B$ | |
| *3* | $\overline{A}BC\checkmark$ | *(2, 6)* $B\overline{C}$ | |
| *4* | $A\overline{B}\overline{C}\checkmark$ | *(4, 6)* $A\overline{C}$ | |
| *6* | $AB\overline{C}\checkmark$ | | |

*1.4. ábra. A Quine-módszer oszlopai*

Gondoljunk arra, hogyan hajtottuk végre az algebrai egyszerűsítést az

$$\overline{A}\,B\,\overline{C}$$

és az

$$\overline{A}\,B\,C$$

mintermek közös tényezőjének, az

$$\overline{A}\,B$$

nek a kiemelésével. A zárójelen belül a

$$\overline{C} + C = 1$$

marad. Ezzel a két összevont minterm helyén a közös szorzó-tényező marad, a zárójelen belül maradó változó eltűnik. A *Quine*-módszer lényege ennek az eljárásnak a szisztematikus ismétlése mindaddig, amíg ilyen eltüntethető változó már nem marad. Ha a változók száma ***n***, először az ***n***-tényezős

minterm párokat vonjuk így össze *n-1* tényezős termekké, azután a kiadódó *n-1* tényezős term-párokat *n-2* tényezős termekké, azután a kiadódó *n-2* tényezős term-párokat *n-3* tényezős termekké, és így tovább.
Az *1.4 ábrán* mutatjuk a szisztematikus eljárást ismert példánkra. Azok a mintermek, amelyekhez a függvény ***1***-t rendel, az **I**.–vel jelölt oszlopban láthatók, bináris értékeikkel megcímezve. A **II**. oszlopban az összevonható minterm-párok címei és az összevonások eredményei láthatók. Ha a **II**. oszlopban lennének összevonható kéttényezős szorzatok, akkor a harmadik oszlop nem lenne üres.
Figyeljük meg, a kiemeléssel összevonható termek sajátossága, hogy azok mindig csak egyetlen változóban különböznek egymástól. Azt mondjuk, hogy az ilyen termek un. *Hamming-féle* távolsága egységnyi.
Miután nincs több oszlop, ahová új összevonásokat írhatnánk, az oszlopokban szereplő, további összevonásokba már nem bevonható termeket *prímimplikánsoknak* nevezzük. Példánk prímimplikánsai :

$$\overline{A}\,B,\ \ B\,\overline{C},\ \ A\,\overline{C}$$

A *Quine*-módszer következő lépése a *feltétlenül szükséges prímimplikánsok* kiválasztása. Ezt a prímimplikánsok lefedési táblázatának segítségével végezhetjük el. (*1.5 ábra*) A prímimplikánsok kijelölik a táblázat sorait, az oszlopokat pedig azok a mintermek, amelyekhez a függvény ***1***-t rendel. Ezután '*' karakterrel bejelöljük az egyes prímimplikánsok által lefedett mintermeket.

*Prímimplikáns táblázat*

| | 2 | 3 | 4 | 6 |
|---|---|---|---|---|
| (2, 3) $\overline{A}B$ | ⊛ | ⊛ | | |
| (2, 6) $B\overline{C}$ | * | | | * |
| (4, 6) $A\overline{C}$ | | | ⊛ | ⊛ |

*1.5. ábra. Prímimplikánsok lefedési táblája*

Nyilvánvaló például, hogy az

$$\overline{A}\,B$$

prímimplikáns az

$$\overline{A}\,B\,\overline{C}\ (2)$$

és a

$$\overline{A}\,B\,C\ (3)$$

mintermeket fedi. A lefedési tábla megmutatja, elhagyhatunk-e úgy egy prímimplikánst, hogy a lefedendő mintermek mindegyike fedve marad.
Ha találunk ilyeneket, azokat elhagyhatjuk. Példánkban egyetlen redundáns, tehát elhagyható prímimplikáns a

$$B\,\overline{C}$$

A lényeges, elhagyhatatlan prímimplikánsok tehát :

$$\overline{A}\,B,\ \ A\,\overline{C}$$

A végeredmény a lényeges prímimplikánsok logikai összege, tehát:

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

### 1.5.3 Logikai függvények Karnaugh táblás (K-táblás) egyszerűsítése

A *Karnaugh*-táblás minimalizálás lényegében a *Quine*-eljárás geometriai reprezentációja. Az *n*-változós függvény lehetséges mintermjeinek megfeleltetünk egy-egy négyzetet, és úgy helyezzük el őket, hogy az egymástól egységnyi távolságra lévő, azaz csak egyetlen bit-ben különböző mintermeket reprezentáló négyzetek szomszédosak legyenek. Ennek az a nagy előnye, hogy igen könnyen észrevesszük az összevonható mintermeket illetve termeket, hiszen azok egymás mellett helyezkednek el.

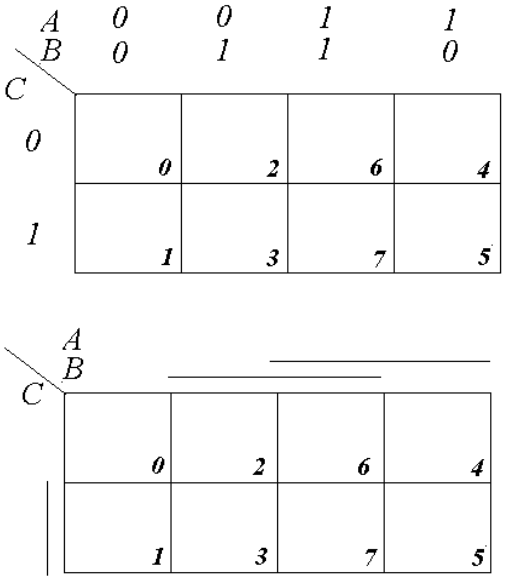
#### 1.5.3.1 Három változós K-tábla

Három változó esetén *8* mintermünk van. Ezeket egy *4 x 2* –es téglalapon helyezzük el, ügyes peremezéssel A peremezés azt jelenti, hogy a változókat két csoportra osztjuk, egy kéttagú és egy egytagú csoportra. Vízszintesen elrendezzük az első csoport lehetséges kombinációit, ***(0 0, 0 1, 1 1, 1 0)***, függőlegesen pedig a második egyváltozós csoport két lehetséges értékét

*(0, 1)* Figyeljünk fel arra, hogy a mintermek geometriai szomszédossága csak úgy biztosítható, ha a csoportok kombinációi is egységnyi távolságra vannak egymástól. Ezért a különleges és szokatlan sorrend, azaz *(0 1)* után az *(1 1)*!
Nézzük az *1.6 ábrát*. Ezzel a peremezéssel a négyzetek megcímezhetők a változókhoz rendelt bináris vektorokkal, ugyanakkor a négyzetek (cellák) jelölhetők is a bináris vektornak megfelelő decimális számjeggyel. Például a felső négyzetsor harmadik négyzete az *( 1 1 0 )* bináris vektorral címezhető, és a ***6***-os decimális értékkel jelölhető. A minterm algebrai alakját az ismert módon olvassuk ki :

$$A\,B\,\overline{C}$$

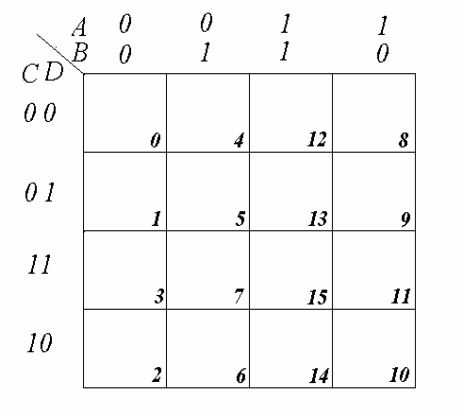
Figyeljünk fel arra is, hogy nem minden logikai szomszédosság jelenik meg geometriai szomszédosságként. Beláthatjuk, hogy a (***0 0 0)*** szomszédos az (***1 0 0)***-val, a (***0 0 1)*** pedig az (***1 0 1)***-vel, bár nincsenek egymás mellett. Ezt szem előtt kell tartanunk a használatnál, de beláthatjuk, hogy egy henger palástjára csavarva a táblát, a geometriai és logikai szomszédosság együttállása tökéletes lenne.




*1.6. ábra. 3-változós K-tábla kétféle formájú peremezéssel*

### 1.5.3.2 4-változós K-tábla

A *4-változós* K-tábla összeállítása és peremezése elvi újdonságot nem jelent, de a széleken elhelyezkedő mintermek közötti szomszédosságokat érdemes megvizsgálni (*1.7 ábra*).

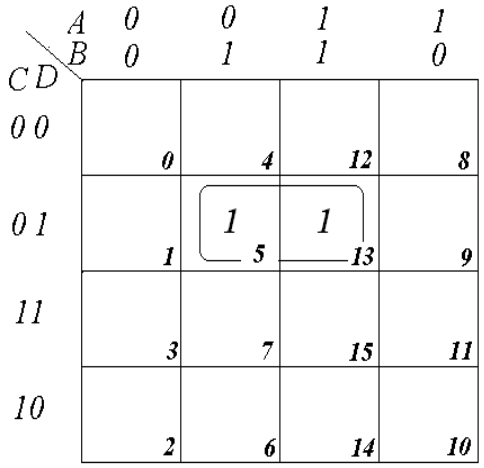



*1.7. ábra. 4-változós K-tábla. Geometriai szomszédosság nélküli szomszédok (0, 8) , (2, 10), (0, 2), (8, 10)*

### 1.5.3.3 Összevont termek kiolvasása a K-táblából

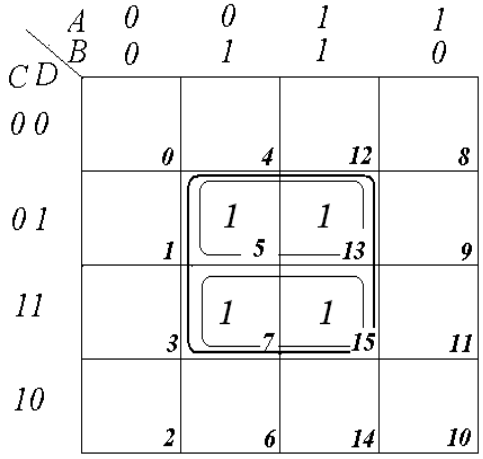
Ez idáig csak az világos, hogyan lehet észrevenni a K-táblán két minterm összevonásának a lehetőségét. Tegyük fel például, hogy a fenti *4*-változós K-táblán ábrázolt mintermek közül az *5*-ös és a *13*-as mintermek mindegyikéhez ***1***-t rendel egy teljesen határozott logikai függvény (*1.8 ábra*). Nemcsak azt vesszük észre a K-táblán, hogy a két minterm összevonható, hanem az összevonás eredményét is azonnal kiolvashatjuk. Az összevont term itt egy kétnégyzetes téglalap, amelynek a címéből az ***A*** változó értéke már hiányzik, hiszen az ***A*** az a változó, amely a „vagy” művelettel kiesik. Így a cím algebrai alakja :

$$B\,\overline{C}\,D$$




*1.8. ábra. Összevont term kiolvasása a K-táblából*

Tegyük most fel, hogy az eddigi mintermeken kívül szerepel a függvényben a *7*-s és a *15*-ös is (*1.9 ábra*). Ez utóbbiakat egymással összevonva felismerjük, hogy az így kialakult két mintermes két term ugyancsak szomszédos, és a ***C*** változó is kiejthető. Ezzel egy geometriailag még nagyobb, négy mintermes term adódott az összevonással, amely már csak két változós. Kiolvasva : ***B D***




*1.9. ábra. Két szomszédos kettes term összevonásával keletkezett 4-es kiolvasása*

### 1.5.3.4. Teljesen határozott logikai függvények egyszerűsítése a K-táblán

A K-táblás minimalizálás lépései következők :

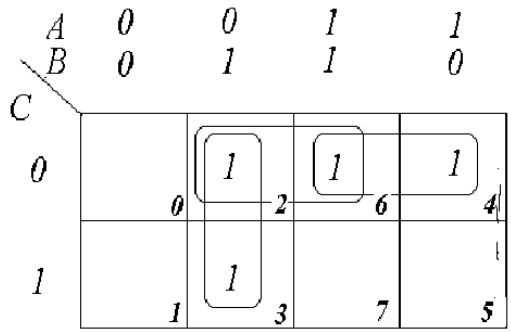
1. *A szükséges méretű K-tábla felvétele*
2. *A fv. '**1**' értékeihez tartozó mintermek bejelölése*
3. *Az összes prímimplikáns-terület kijelölése összevonással*
4. *Az egyszerűsített fv. felírása a prímimplikánsok közül való válogatással*

Egyszerűsítsük K-táblán a már ismert, teljesen határozott függvényünket:

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

Az 1. a 2. és a 3. lépést illusztráljuk az *1.10. ábrán*. Az ***1***-es mintermek bejelölése után megkezdődhet a párosával történő összevonás. Mivel tagjaik szomszédosak, összevonhatók a következő párok: ***(0 1 0, 0 1 1), (0 1 0, 1 1 0), (1 1 0, 1 0 0)***. Ezután azt vizsgáljuk, hogy a három összevont kettes csoportok között vannak-e szomszédos, nagyobbakká összevonhatók. Szembetűnő, hogy nincsenek ilyenek, tehát a három kétváltozós term mindegyike *prímimplikáns*.

A 4. lépés ugyancsak a K-táblán végezhető el a legegyszerűbben. Képzeljük el, hogy a prímimplikánsok négyszögei lemezek, amelyek felemelhetők a tábláról. Ha ezeket sorra felemeljük, és azt találjuk hogy az felemelt term által fedett valamennyi minterm a többi term által fedve marad, akkor a felemelt term felesleges, eltávolítható. Ezzel szemben, ha a felemelés következtében valamelyik minterm fedetlen marad, a prímimplikáns elhagyhatatlan.




*Prímimplikánsok :*

$$\overline{A}B,\ B\overline{C},\ A\overline{C}$$

*1.10. ábra. Példánk prímimplikánsainak megkeresése 3-változós K-táblán*

Ezzel a technikával befejezhetjük feladatunk egyszerűsítését : A

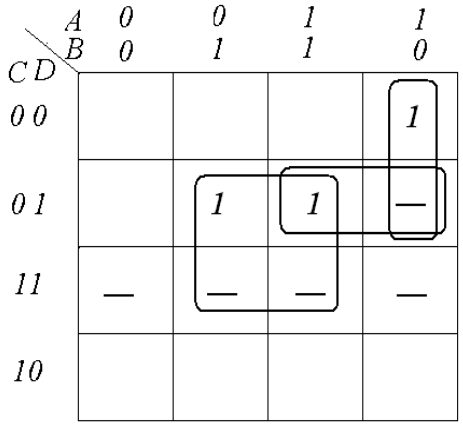
$$B\,\overline{C}$$

term felesleges, elhagyható.

### 1.5.3.5. Nem teljesen határozott logikai függvények egyszerűsítése a K-táblán

Ha a logikai függvény nem teljesen határozott, akkor legalább egy olyan bemeneti kombináció, azaz minterm van, amelyhez rendelt függvény érték számunkra közömbös. Ilyenkor különböző szimbólumokkal jelöljük be a K-táblába az ***1***-es és közömbös mintermeket. Ez utóbbiakat célszerű a már ismert kis vízszintes vonalkával jelölni. A közömbös mintermekkel szabadon bánhatunk. Ha előnyös az egyszerűsítés szempontjából, akkor összevonjuk őket az ***1***-es mintermekkel, ha nem, akkor ***0***-s mintermeknek tekintjük őket. Lássunk egy példát közömbös mintermekkel rendelkező 4-változós logikai függvény egyszerűsítésére (*1.11 ábra*).

A prímimplikánsok között egy négy-mintermes, tehát két-változós, és két két-




*1.11. ábra. Prímimplikánsok közömbös bejegyzések közötti válogatással*

mintermes, azaz három-változós term szerepel. Kiolvasva :

$$B\,D\,,\ A\,\overline{C}\,D,\ A\,\overline{B}\,\overline{C}$$

Ezek közül a második elhagyása nem változtat a teljes lefedésen.

## 1.6 EGYKIMENETŰ KOMBINÁCIÓS HÁLÓZATOK TERVEZÉSE

### 1.6.1 Teljesen specifikált, egykimenetű hálózatok tervezése

Az egy-kimenetű, teljesen specifikált kombinációs hálózatot egyetlen, teljesen határozott logikai függvénnyel specifikáljuk. Ez történhet igazságtáblázattal, az ***1***-es mintermek számjegyes felsorolásával, vagy algebrai alak megadásával. A tervezés lépései a következők:

*1. Egyszerűsítés Karnaugh táblával*
*2. Döntés a logikai építőelemek választékáról*
*3. Realizáció*

Igazságtábla vagy számjegyes minterm felsorolás esetén az ***1***-es mintermek táblázatba-vitele után megindulhat a prímimplikánsok megkeresése, és a szükséges prímimplikánsok kiválasztása. Algebrai alak esetén a megadott szorzattermek K-táblán történő ábrázolása után megpróbálunk egy egyszerűbb lefedést találni.

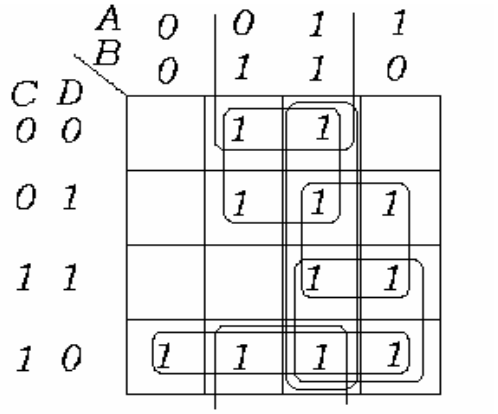
### 1.6.2. Tervezési példa

Tervezzük meg ***NÉS*** kapukkal a következő specifikációval megadott négybemenetű, teljesen specifikált kombinációs hálózatot.

$$F : ( 2, 4, 5, 6, 9, 10, 11, 12, 13, 14, 15)$$

***Megoldás.***

*1. A K-tábla felvétele, összevonások, a prímimplikánsok megkeresése*




*1.12. ábra. Az F : ( 2, 4, 5, 6, 9, 10, 11, 12, 13, 14, 15) prímimplikánsai*

Kiolvasva a prímimplikánsokat:

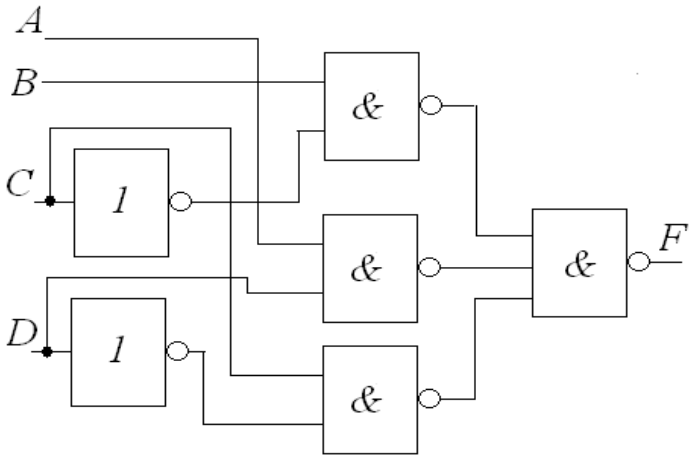
$$B\,\overline{D},\ B\,\overline{C},\ A\,D,\ A\,C,\ C\,\overline{D},\ A\,B$$

Meghatározva a minimális irredundáns lefedést :

$$B\,\overline{C},\ A\,D,\ C\,\overline{D}$$

Átalakítva a kapott függvényt a *De-Morgan* azonosság felhasználásával az *1.13. ábra* szerinti realizációt kapjuk.

$$F = \overline{\overline{B\overline{C} + AD + C\overline{D}}} = \overline{\overline{B\overline{C}} \cdot \overline{AD} \cdot \overline{C\overline{D}}}$$




*1.13. ábra. Realizáció NÉS kapukkal*

### 1.6.3. Nem teljesen specifikált, egykimenetű hálózatok tervezése

Az egy-kimenetű, nem teljesen specifikált kombinációs hálózatot egyetlen, *nem teljesen határozott* logikai függvénnyel specifikáljuk. Ez történhet igazságtáblázattal, az ***1***-es és a *közömbös* mintermek felsorolásával. A tervezés lépései itt is a következők:

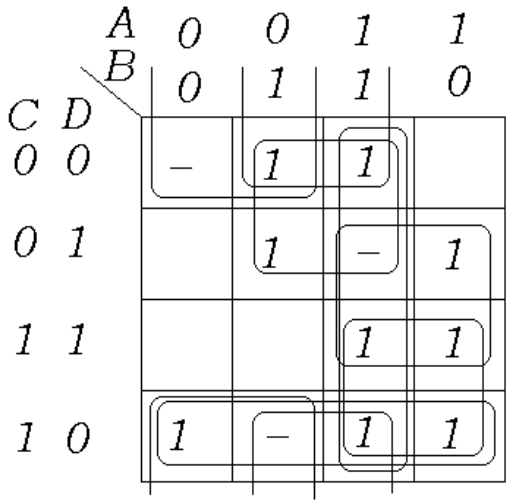
1. *Egyszerűsítés Karnaugh táblával*
2. *Döntés a logikai építőelemek választékáról*
3. *Realizáció*

### 1.6.4. Tervezési példa nem teljesen specifikált esetre (1)

Felsoroljuk az ***1***-es és *közömbös* mintermeket:

***$F_1$ : ( 2, 4, 5, 9, 10, 11, 12, 14, 15)***
***$F_{dc}$ : (0, 6, 13)***

Feltünteve ezeket a K-táblán, az *1.14. ábra* szerinti elrendezést kapjuk




*1.14. ábra. Az $F_1$ : ( 2, 4, 5, 9, 10, 11, 12, 14, 15), $F_{dc}$ : (0, 6, 13) függvény prímimplikánsai a K-táblán*

*Prímimplikánsok :*

$$\overline{A}\,\overline{D},\ B\,\overline{D},\ B\,\overline{C},\ A\,D,\ A\,C,\ C\,\overline{D},\ A\,B$$

*Minimális irredundáns lefedés:*

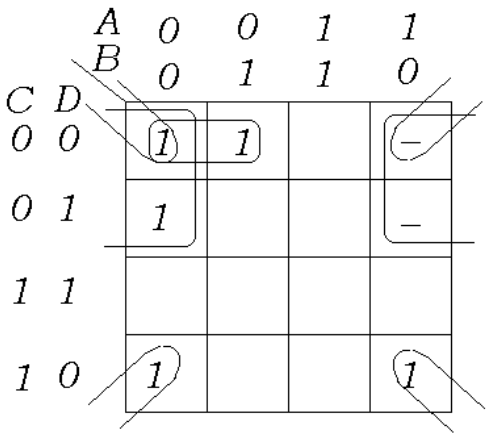
$$B\,\overline{C},\ A\,D,\ C\,\overline{D}$$

### 1.6.5. Tervezési példa nem teljesen specifikált esetre (2)

Ezt a feladatot igazság-táblával adjuk meg:

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

A prímimplikánsokat az *1.15. ábrán* mutatjuk meg. Figyeljünk fel a sarkok szomszédosságából adódó 4 mintermből álló termre..




*1.15. ábra. A 2-es tervezési példa K-táblája*

A tábla alapján kapott irredundáns lefedés:

$$\overline{C}\,\overline{B},\ \ \overline{A}\,\overline{C}\,\overline{D},\ \ \overline{B}\,\overline{D}$$

Láthatjuk, hogy itt nincs felesleges prímimplikáns.

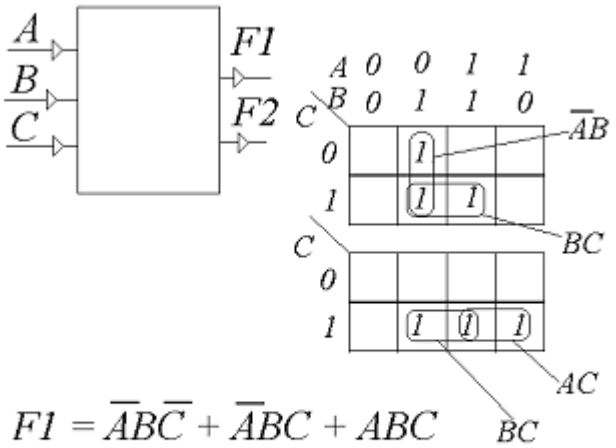
*EGYSZERŰ FELADAT : Realizáljuk NÉS kapukkal az előbbi prímimplikánsokkal lefedett függvényt!*

## 1.7. TÖBBKIMENETŰ KOMBINÁCIÓS HÁLÓZATOK TERVEZÉSE

A több-kimenetű, például *m*-kimenetű kombinációs hálózatot *m* számú logikai függvénnyel adunk meg. Lehetséges volna, ha ezeket egymástól teljesen függetlenül egyszerűsítenénk és realizálnánk. Ennél azonban sokszor van egyszerűbb alakra vezető megoldás is. Ennek illusztrálására tekintsük a következő bevezető tervezési feladatot.

### 1.7.1. Egy bevezető példa

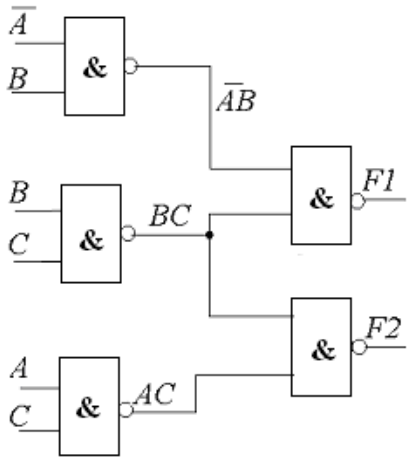
Képzeljük el, hogy egy három-bemenetű, két-kimenetű hálózat függvényeinek lefedésekor a következő K-táblákra jutottunk:




$$F1 = \overline{A}B\overline{C} + \overline{A}BC + ABC$$

$$F2 = \overline{A}BC + A\overline{B}C + ABC$$

*1.16. ábra. Kétkimenetű hálózat függvényeinek lefedése*



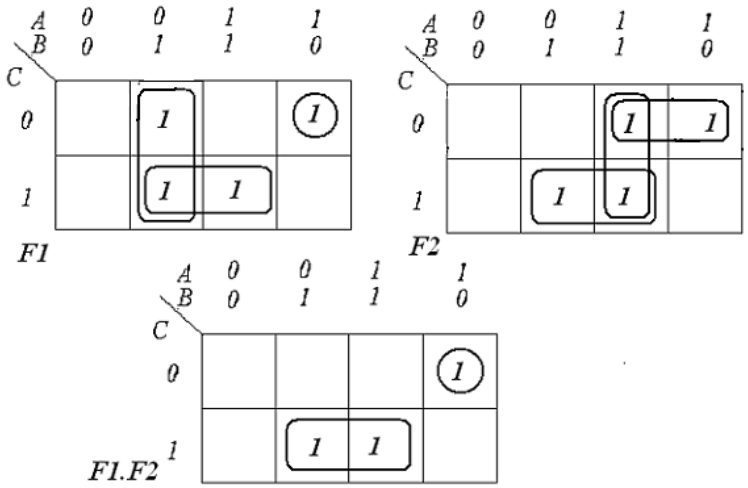

*1.17. ábra. Realizáció a közös prímimplikáns egyszeri megvalósításával*

A két függvény prímimplikánsai között találunk egy közös, mindkét lefedésben prímimplikáns szerepet játszó termet, a ***BC***-t. Ezt nyilvánvalóan csak egyszer, azaz egy *ÉS* vagy egy *NÉS* kapuval realizáljuk, hiszen mindkét második szinten levő *VAGY*, illetve *NÉS* kapuba bevezethető az ezt reprezentáló logikai jel. A közös prímimplikánsokat tehát célszerű megkeresni. Arra is gondolnunk kell azonban, hogy nemcsak a közös prímimplikánsok egyszeri megvalósítása egyszerűsítheti a realizációt, hanem a közös implikánsok is. Ezek közül a legnagyobbakat érdemes megkeresni.
Fontos igazság, hogy a legnagyobb közös implikánsokat a *függvények szorzatánk* lefedésével találjuk meg. A szorzatfüggvény prímimplikánsai között ott vannak a kért függvény legnagyobb közös impplikánsai, amelyek között természetesen a közös prímimplikánsok is ott vannak. *A szorzatfüggvény lefedése nagyon egyszerű, a K-táblába bejegyezzük azokat a mindkét függvényben 1-es értékkel szereplő mintermeket*. Így minden egyes függvény lefedéséhez nemcsak a saját prímimplikánsokat, hanem a legnagyobb közös implikánsokat is számításba vesszük.

### 1.7.2. Prímimplikáns készlet többkimenetű kombinációs hálózatok egyszerűsítéséhez

Bevezető példánkban találtunk egy közös, mindkét függvényben szereplő prímimplikánst, és ezt csak egyszer kellett realizálnunk. Ebből következik, hogy a két kimenetet nem célszerű egymástól függetlenül egyszerűsíteni. *Az előző pontban leírtak alapján a két függvény legnagyobb közös implikánsait megkapjuk, ha előállítjuk a szorzat függvény prímimplikánsait. Ezek között a közös prímimplikánsok is megjelennek.*

Állításunkat nem bizonyítjuk, hanem az *1.18. ábrán* illusztráljuk azt




*1.18. ábra. A legnagyobb közös implikánsok megkeresése a szorzat-függvény prímimplikánsainak kijelölésével*

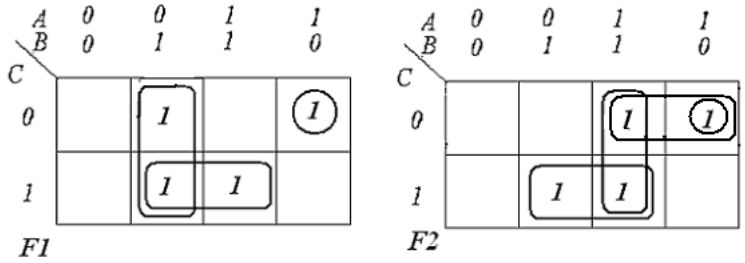
A prímimplikáns készleteket tehát a következő term-halmazokból állítjuk össze:

$$F1 : \overline{A}\,B,\ B\,C,\ A\,\overline{B}\,\overline{C}$$

$$F2 : B\,C,\ A\,B,\ A\,\overline{C}$$

$$F1.F2 : BC,\ A\,\overline{B}\,\overline{C}$$

Látjuk, hogy a ***BC*** közös prímimplikáns, az $A\,\overline{B}\,\overline{C}$ pedig két nem közös prímimplikáns közös része, azaz egy legnagyobb közös implikáns. A kétkimenetű hálózat függvényeinek lehetséges lefedő-termjeit tehát az *1.19. ábrán* látható csoportokból állítjuk össze.




*1.19. ábra. Az F1 és F2 függvények prímimplikánsai, kiegészítve a legnagyobb közös implikánssal*

Az *1.19. ábra* alapján elvégzett lefedés-vizsgálat és a felesleges implikánsok eltávolítása során előnyben részesítjük a legnagyobb közös implikánsokat. Így az ***F2*** teljes lefedéséhez és felépítéséhez az

$$A\,\overline{C}$$

helyett a közös,

$$A\,\overline{B}\,\overline{C}$$

termet használjuk. Természetes, hogy a ***BC***-t csak egyszer kell realizálni.

A fentiek több kimenetre való általánosítása alapján megfogalmazható a többkimenetű hálózat egyszerűsítésének célszerű folyamata

1. *Megkeressük valamennyi kimenethez rendelt függvény prímimplikánsait*
2. *Megkeressük valamennyi lehetséges függvény-szorzat prímimplikánsait.*
3. *Minden egyes kimeneti függvény mintermjeit megpróbáljuk lefedni a következő készletből :*
   - *a saját, más kimenetekhez nem tartozó prímimplikánsokkal,*
   - *azokkal a maximális közös implikánsokkal, amelyek az adott függvénynek implikánsai.*

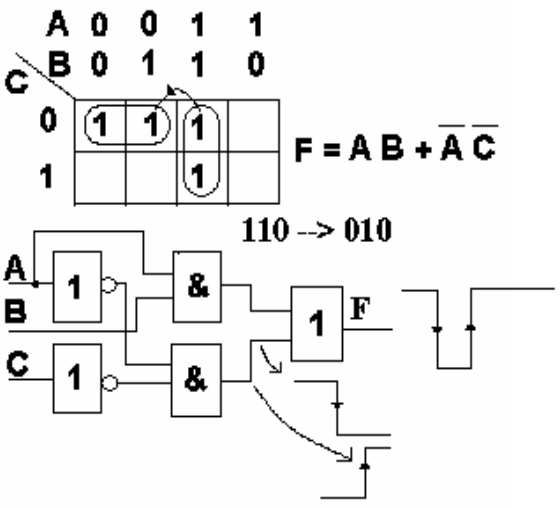
*EGYSZERŰ FELADAT : realizáljuk az 1.19. ábra szerinti két-kimenetű hálózatot NÉS kapukkal!*

## 1.8. HAZÁRDOK

Az eddigiek során a kombinációs hálózatokat statikusan szemléltük, nem foglalkoztunk a tranziensekkel, azaz az átmeneti jelenségekkel. Általában megköveteljük, hogy *bemenet változásának hatására a kimenet a specifikációnak megfelelően reagáljon*, azaz, ha a bemeneti vektor úgy változik, hogy a kimenet szintje megváltozik, akkor ez a változás egy bizonyos időbeli késleltetéssel, de *egyszeri eseményként menjen végbe*. Az ettől való eltérések a *hazárdok.*

### 1.8.1. A statikus hazárd keletkezése

Tekintsük meg az *1.20 ábrán* látható hálózat tranziensét, azaz átkapcsolását az (***1 1 0)*** bemenetre beállt állapotból a (***0 1 0)*** bemenetre beálló állapotba, azaz azt a tranzienst, amelyet a hálózat az ***A*** bemenet ***1***-ből ***0***-ba való átmenetre mutat. A tranziens analízis során feltételezzük, a minden egyes kapufokozatnak valamekkora késleltetési ideje van. Beláthatjuk, hogy az ***A*** lefutásakor a *VAGY* kapu egyik bemenetén a magas szint biztosan előbb fut ***0***-ra, mint a másik ***1***-re, hiszen ez utóbbi változás két kapun, egy inverteren és egy *ÉS* kapun halad át, míg az előbbi késleltetése csak egy *ÉS* kapunyi. Van tehát egy átmeneti idő-intervallum, amikor A *VAGY* kapu egyik bemenete már nem, a másik bemenete még nem ***1***-es szintű. Annak ellenére tehát, hogy a kimenetnek a logikai specifikáció szerint ***1***-ben kéne maradnia, átmenetileg lefut ***0***-ra.




*1.20. ábra. Hazárd keletkezése*

### 1.8.2. A statikus hazárd jelenségek pontos meghatározása

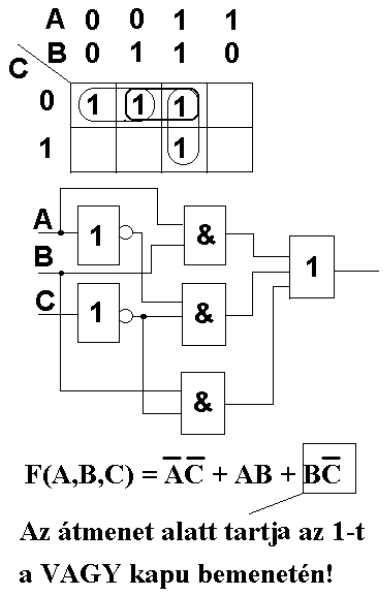
*Ha egyetlen bemeneti változó logikai értékének megváltozásakor a kimenet a specifikáció szerint nem változna, a realizált hálózat kimenetén mégis átmeneti változás zajlik le, statikus hazárdról beszélünk.*

***1-es típusú statikus hazárd:*** ha a specifikált hálózat kimenete a bemeneti változás ellenére magasan marad, de a realizált hálózat egy '***0***' impulzust mutat.

***0-s típusú statikus hazárd :*** ha a specifikált hálózat kimenete a bemeneti változás ellenére alacsonyan marad, de a realizált hálózat egy '***1***' impulzust mutat.

### 1.8.3. A statikus hazárdok kiküszöbölése *(1.21. ábra).*

A statikus hazárdok kiküszöbölése igen egyszerű, redundáns implikánsok bevezetésével. Az ***1***-es típusú statikus hazárd veszély mindig abból adódik, hogy különböző prímimplikánsokhoz tartozó ***1***-es mintermek kerülnek egymás mellé. Egy újabb áthidaló implikáns bevezetésével elérjük, hogy az átmenet alatt a kimenet logikai szintje állandó marad.




*1.21. ábra. A statikus hazárd kiküszöbölése*

### 1.8.2 Dinamikus hazárd

Dinamikus hazárdról beszélünk, ha a egy bemeneti-változó értékváltására a kimenetnek logikai értéket kell váltania, de ez egy átmeneti visszatérés kíséretében zajlik le. Belátható, hogy a dinamikus hazárd többszintű hálózatokban lép fel akkor, ha a hálózat valamely része nincs statikus-hazárdoktól mentesítve. A statikus hazárdokat kell kiküszöbölni, így a dinamikus hazárd eltűnik.

### 1.8.3 Funkcionális hazárd

Több bemeneti változó együttes változása többszörös szintváltáshoz vezet. Csak a késleltetések manipulálásával küszöbölhetők ki, de az a legjobb, ha a tranziensek továbbterjedését megakadályozzuk szinkronizációval.

*TERVEZÉSI FELADATOK 1.*

**1**. Tervezzen statikus hazárdoktól mentes olyan négy-bemenetű hálózatot, amely a binárisan kódolt számok közül ***1***-es szintű kimenettel jelzi a *páratlan prímszámokat*. A hálózatot ***NAND*** kapukból kell összeállítani.

**2.** Tervezzen olyan négy-bemenetű hálózatot, amely ***1***-es szintű kimenettel jelzi, ha a bemeneten több ***1*** van, mint ***0***. A hálózatot ***NAND*** kapukból kell összeállítani.

**3**. Egy hálózat bemenetére *4-bites bináris számok* érkeznek, ***0****-tól* ***9***-ig. A hálózat egyetlen kimenetén ***1***-et ad, ha a bemenetre érkező bináris szám decimális értéke ***1****, vagy páratlan és osztható* ***3****-al.* Tervezze meg a hálózatot kizárólag két-bemenetű ***NAND*** kapukkal!

*2. Rész. sorrendi hálózatok tervezése*

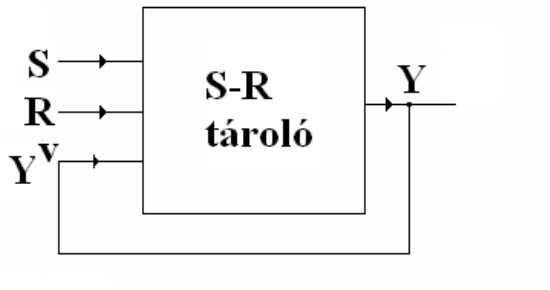
## 2.1. ELEMI SORRENDI HÁLÓZATOK, TÁROLÓK

Ebben a fejezetben a kombinációs hálózatokkal kapcsolatban megismert módszerekkel olyan egyszerű logikai elemeket ismerünk meg, amelyeket a sorrendi (szekvenciális) hálózatok építőelemeiként fogunk felhasználni. Ezeket az áramköröket összefoglaló néven *tárolóknak* nevezzük. A szekvenciális hálózatok általános tulajdonságait, tervezésük általános módszereit a tárolók megismerése után tanulmányozzuk.

### 2.1.1. S-R tároló működése és igazság-táblái

Az aszinkron **S-R** tárolónak két bemenete (**S, R**) és egy kimenete (**Y**) van. A tároló viselkedését a következőképpen adjuk meg:
*Ha a tároló mindkét bemenetére **0**-t kapcsolunk, a kimenet nem változik. Ha csak az **R** bemenetet emeljük fel, akkor ennek hatására a kimenet, aktuális szintjétől függetlenül **0**-ra áll. Ha csak az **S** bemenetet emeljük fel, akkor a kimenet, aktuális szintjétől függetlenül **1**-re áll.*
*Azt, hogy mindkét bemenetet egyidejűleg felemeljük, megtiltjuk.*

Felhívjuk a figyelmet arra, hogy ebben a specifikációban szokatlan dolgot fogalmaztunk meg. Nevezetesen, ha a tároló kimenete **1,** és (**0 0**)-t kapcsolunk a bemenetre, akkor a kimenet **1** szintű marad, míg ha ugyanezt a kimenet **0** állapotában tesszük, akkor a kimenet **0** marad. Ugyanarra a bemeneti kombinációra adott válasz tehát a kimenet a saját korábbi értékétől is függ, nemcsak a bemenetektől. Ez tehát nem kombinációs hálózat!
A *2.1 ábrán* látjuk, hogy az *áramkör kimenete visszakerül a bemenetek közé*.




*2.1. ábra. Az S-R tároló logikai sémája*

A specifikáció alapján megalkotható a tároló *egyszerű igazságtáblája* (*2.2.a. ábra*). Az egyszerű igazságtábla csak a bemeneti kombinációkhoz tartozó

kimeneteket, azaz állapotokat tartalmazza, de ezúttal úgy, hogy a kimenetek értékei között a *nullákon* és az *egyeseken* kívül az aktuális állapot, vagy negáltja is megjelenhet. Az egyszerű igazságtábla alapján könnyen generálható az úgynevezett *összetett igazságtábla (2.2.b. ábra)*, amelyben a kimenet aktuális értékei a bemeneti kombinációk közé kerülnek, és az igazságtábla kimeneti változóját a kimenet *következő értékeinek* meghatározására használjuk. Ezt úgy is felfoghatjuk, hogy egy olyan speciális kombinációs hálózatot tervezünk, amelyen ugyanaz a változó kimenetként és bemenetként is szerepel - azaz a kimenet a bemenetre vissza van vezetve - de az igazságtáblába írt értékek különbözhetnek, hiszen időbeli eltolódás van közöttük. Formálisan meg is különböztetjük a bemenetként szereplő változót a kimenetként szereplő változótól. A bemeneten az **Y** szimbólumot ellátjuk egy felső „***v***” (visszacsatolt) index-vel. Ennek megfelelően az $\mathbf{Y}^{v}$ szintjeit tekintjük *aktuális kimeneti értékeknek*, és a **Y** szintjeit *következő kimeneti értékeknek.*
Az összetett igazságtábla kitöltésekor fontos, hogy az **S=1, R=1** bemeneti kombináció tiltott, ezért ott élhetünk a közömbös kimenet előírással.

*egyszerű igazságtábla*

| S | R | Y |
|---|---|---|
| 0 | 0 | $Y^V$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | - |

*2.2.a. ábra. Az S-R tároló egyszerű igazságtáblája*

*összetett igazságtábla*

| S | R | $Y^V$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | - |
| 1 | 1 | 1 | - |

*2.2.b. ábra. Az R-S tároló összetett igazságtáblája*

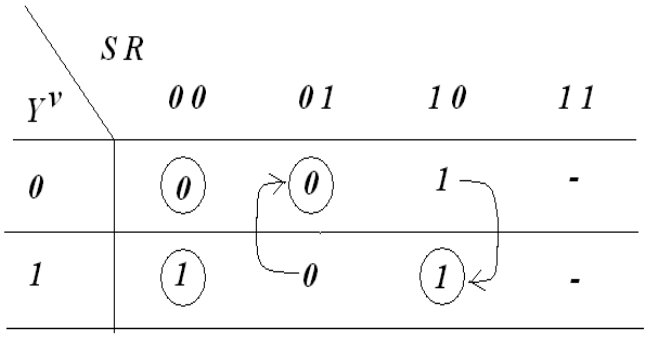
### 2.1.2. Az S-R tároló állapot-átmeneti táblája

A viselkedés leírása alapján az *állapot-átmeneti tábla, más néven állapottábla* (*2.3. ábra*) felírása következik. A balszélső oszlopban felsoroljuk az aktuális kimeneti állapotokat, a többi oszlopot pedig a bemeneti kombinációkkal jelöljük. Az összetett igazság-tábla értékeit egyszerűen bemásoljuk ebbe az új struktúrájú táblázatba.

Nagyon fontos annak megjelölése, hogy a bejegyzett következő állapot csak *átmenetileg jelentkező (tranziens)*, vagy a rákapcsolt bemeneti kombináció fenntartása mellett *nem változik (stabil)*.

Vizsgáljuk meg, hogyan állapítható meg az állapottáblából a stabilitás vagy az instabilitás ténye. Ha a bemenetekre az **S = 0, R = 0** bemeneti kombinációt kapcsoljuk, és az aktuális kimeneti érték **0**, azaz $\mathbf{Y^v = 0}$, akkor a következő állapot is 0, és ez a bemenetre visszakerülve nem változtat a új következő kimeneti értéken. Azaz az **S = 0, R = 0**-nál az **Y = 0** stabil kimeneti állapot. Ezzel szemben **S = 1, R = 0,** $\mathbf{Y^v = 0}$ instabil, hiszen az erre következő kimeneti állapot az **Y = 1**. Ez viszont stabilizálódik, hiszen visszakerülve a bemenetre, nem vált ki újabb változást.

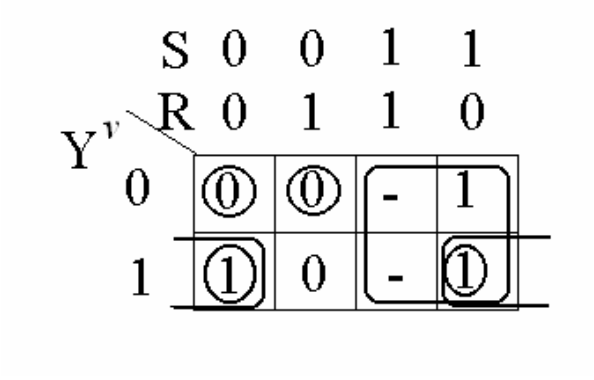
*Ennek alapján az állapottábla azon kimeneti állapotai stabilak, amelyeknél a bejegyzett következő kimeneti állapot megegyezik az aktuális kimeneti állapottal.*




*2.3. ábra. Az S-R tároló állapot-átmeneti táblája*

### 2.1.3. K-tábla az S-R tároló megvalósítására

Akár az igazságtábla, akár az állapottábla könnyen átrajzolható minimalizálásra alkalmas **K**-táblává is (*2.4. ábra*). A stabil állapot-értékeket itt is jelöltük, bár a minimalizálás, illetve megvalósítás szempontjából erre már nincs szükség. A **K**-táblán elvégzett lefedésből adódó **ÉS-VAGY** és **NÉS-NÉS** realizáció algebrai alakjait is feltüntettük az ábrán.

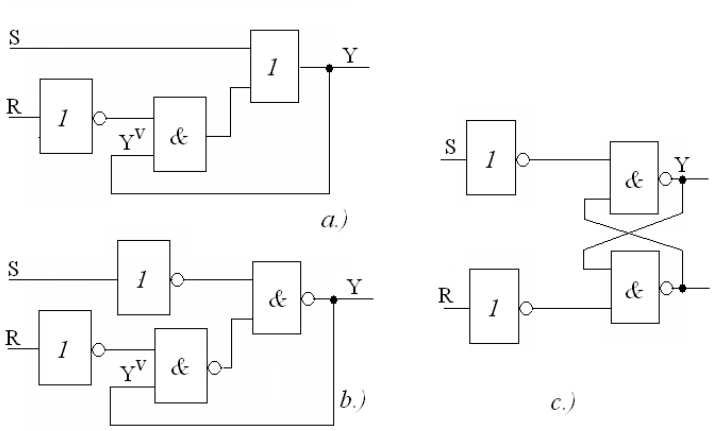



$$Y = S + \overline{R}\,Y^{v} = \overline{\overline{S + \overline{R}\,Y^{v}}} = \overline{\overline{S}\,(\overline{\overline{R}\,Y^{v}})}$$

*2.4. ábra. Az S-R tároló K-táblája és annak lefedése*

### 2.1.4. Az S-R tároló realizációi *(2.5. ábra)* .

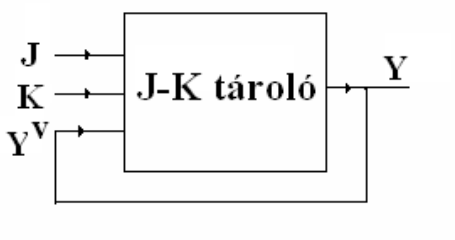
A realizációkat bemutató ábrák közül a legismertebb forma a *c.* részábra szerinti. Lássuk be, hogy az **S-R** tároló minden stabil állapotában az **Y** kimeneten megjelenő érték negáltja jelenik meg a másik **NÉS** kapu kimenetén.
Ezért ezt a kimenetet gyakran jelölik az **Y** negáltjával.




*2.5. ábra. Az S-R tároló ÉS-VAGY realizációja (a), NÉS-NÉS realizációja (b), és az utóbbinak egy ismert alakú logikai sémája (c).*

### 2.1.5. Kísérlet J-K tároló megvalósítására

Módosítani szeretnénk az **S-R** tároló működését úgy, hogy a két bemenet egyidejű felemelését megengedjük, és erre azt szeretnénk, ha a tároló állapota ellenkezőjére változna. Természetesen a bemenetek jelölése megváltozik, **J**-re és **K**-ra. Az új sémát a *2.6. ábra* mutatja.




*2.6. ábra. A J-K tároló logikai sémája*

Az *2.7. ábrán* látható egyszerű és összetett igazságtábla mutatják az elvárt működést.

*egyszerű igazságtábla*

| J | K | Y |
|---|---|---|
| 0 | 0 | $Y^v$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\overline{Y^v}$ |

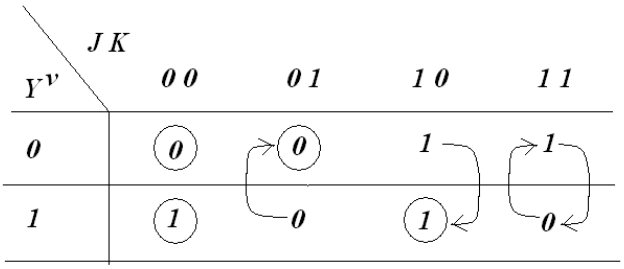
*összetett igazságtábla*

| J | K | $Y^v$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

*2.7. ábra. A kísérleti J-K tároló igazságtáblái*

### 2.1.6. A kísérleti J-K tároló állapot-átmeneti táblája

Szerkesszük meg most az J-K kimenetére érvényes állapot-átmeneti táblát a stabil kimeneti állapotok bejelölésével (*2.8. ábra*):

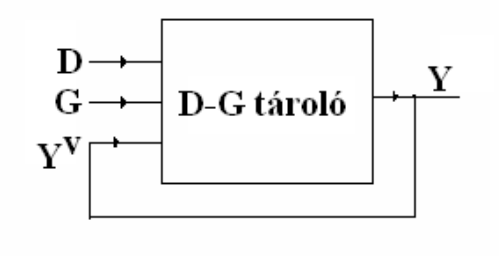



*2.8. ábra. A kísérleti J-K tároló állapottáblája*

Láthatjuk, hogy az aszinkron **J-K** tárolónak a **J = 1 K = 1** esetben egyik aktuális állapot-értéknél sincs stabil állapota. Ez azt jelenti, hogy ez a tároló használhatatlan. A kísérlet negatív eredménnyel zárult, és kimondhatjuk, hogy ilyen **J-K** tároló megépítésének nincs értelme.

### 2.1.7. D-G tároló *(2.9. ábra)*

A **D-G** tároló működését úgy definiáljuk, hogy a tároló a **G** felemelésekor írja a kimenetre a **D** aktuális értékét, majd a **G** lefutásakor ez az érték maradjon meg a kimeneten.




*2.9. ábra. A D-G tároló logikai sémája*

Az igazságtáblák, az állapot-tábla a stabil állapotok bejelölésével, valamint a **K**-táblák a vázolt működés alapján könnyen felírhatók. ( *2.10 .a.- d . ábrák*)

*egyszerű igazságtábla*

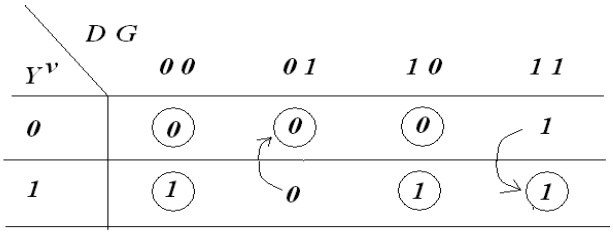
| D | G | Y |
|---|---|---|
| 0 | 0 | $Y^V$ |
| 0 | 1 | 0 |
| 1 | 0 | $Y^V$ |
| 1 | 1 | 1 |

*2.10.a. ábra. A D-G tároló egyszerű igazságtáblája*

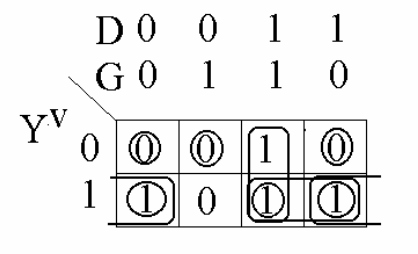
*összetett igazságtábla*

| D | G | $Y^V$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

*2.10.b. ábra. D-G tároló egyszerű igazságtáblája*




*2.10.c. ábra. A D-G tároló állapottáblája*




*2.10.d. ábra. A D-G tároló K-táblája*

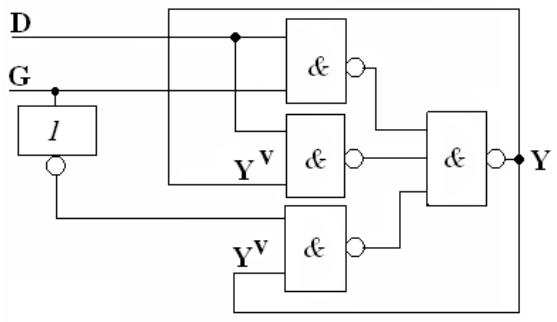
A **K**-táblán ezúttal a függvény valamennyi prímimplikánsát feltüntettük, mert az **1**-ek elhelyezkedése emlékeztet bennünket a statikus hazárdokra jellemző helyzetre. Az **S-R** tároló **K**-tábláján ilyen helyzetet nem láttunk, ezért a statikus hazárd problémája ott fel sem merült. Vizsgáljuk meg, hogy egy nem hazárdmentes lefedés, az

$$Y = D\,G + \overline{G}\,Y^{v}$$

megoldás zavart okoz-e a tároló működésében.
Tegyük fel, hogy a ***D, G - Y***$^{v}$ jelek rendjében (**11- 1**)-ben vagyunk, és **G** leemelésével az (**10-1**) helyzetbe akarunk átmenni. Ha **G** előbb lefut, minthogy a negált **G** felfutna, beállhat az (**10-0**) állapot, és ennek hatására **0**-ban stabilizálódnánk !
A helyes működés érekében kell tehát a statikus hazárdtól való mentesítés. Ezzel a realizáció ***ÉS-VAGY*** alakja:

$$Y = D\,G + D\,Y^{v} + \overline{G}\,Y^{v}$$

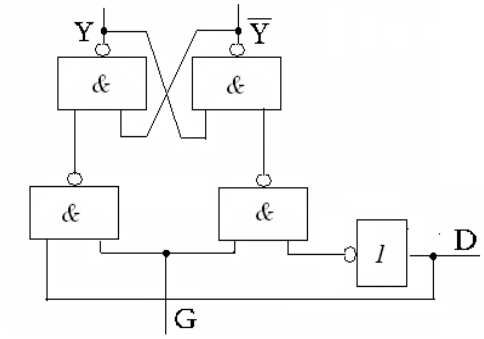
A ***NÉS-NÉS*** realizációt a *2.11. ábrán* láthatjuk.




*2.11. ábra. A D-G tároló statikus-hazárdmentesített NÉS-NÉS realizációja*

### 2.1.8. D-G tároló egy ekvivalens alakja

A *2.12. ábrán* látható megoldás könnyen kialakítható az ismert ***S-R*** tároló sémájából. Belátható, hogy ez a megoldás teljesen ekvivalens a statikus hazárdmentes lefedéssel előállított változattal. Bizonyítsuk be az ekvivalenciát algebrai módszerrel!



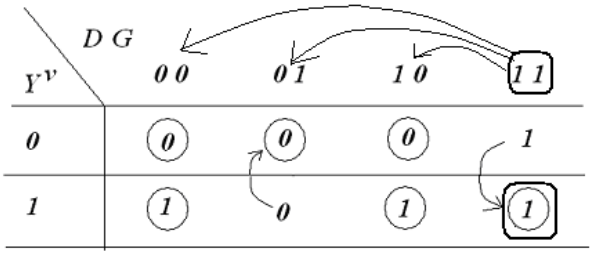

2*.12. ábra. A D-G tároló egy ismert alakja*

### 2.1.9. A D-G tároló, mint memória-elem

A **D-G** tárolót **1**-bites memória-elemnek tekinthetjük. A beírandó adatot a **D** bemeneten előkészítjük, majd a **G** beíró bemenetet magas szintre emeljük. A tranziens lejátszódása után a beírt szint az **Y** kimeneten stabilizálódik. Ezután A **D** értékét még nem változtatva visszaejtjük a **G** bemenet szintjét. Az **Y** kimeneten a beírt érték ott marad. A **G** lefutása után **D** hiába változik, a kimenetre ez már hatástalan.

### 2.1.10. Többszörös bemeneti váltás hatásai a D-G tárolóra

Vizsgáljuk a *2.13. ábra* alapján realizált ***D-G*** működését olyan bemeneti kombináció váltásoknál, amikor egyszerre mindkét bemenetet változtatjuk.




*2.13. ábra. A D-G viselkedése többszörös bemeneti szint-váltásoknál*

Vizsgáljuk például a **D G – Y**$^{v}$ jelek rendjében az ***1 1-1*** *helyzetből előidézett* ***0 0*** bemeneti átmenet hatását. Azt várjuk, hogy a hálózat megtartja az ***1***-est, hiszen az állapottábla szerint a ***0 0-1*** nél stabil ***1***-es bejegyzés szerepel.
Sajnos ezt két okból sem garantálhatjuk. Ezek a következők:

- Két bemenetet tökéletesen egy időben nem tudunk változtatni
- A két bemeneti változás hatása különböző késleltetésű utakon érvényesül

A valóságban tehát nem lehet kizárni, hogy vagy az ***10***, vagy a ***01*** bemeneti kombináció átmenetileg beáll. Így ha a ***01*** jelentkezik, azaz ***D*** lefutásának hatása előbb érvényesül, a kimeneten beállhat a ***0*** tranziens, amely végül, miután ***G*** is lefut, az ***Y = 0*** kimenetet stabilizálja.
Legjobb, ha megtiltjuk, a többszörös bemeneti váltásokat, azaz *egyszerre csak egyetlen egy bemeneti jel értéke változhat meg*.

*Feladat : Elemezzük az* ***1 0 – 1*** *helyzetből a* ***D = 0, G = 1*** *be való váltás lehetséges kimeneteleit a realizált tárolón!*

### 2.1.11. A D-G tároló „átlátszósága”

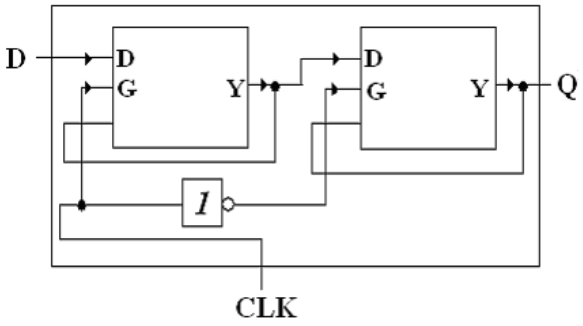
A **D-G** tároló sajátossága, hogy a **G=1** helyzetben a **D**-re adott változások kijutnak a kimenetre. A **G=1** helyzetben tehát a tároló a **D**-bemenet felől „*átlátszó” (transzparens)*.
Felmerült az igény olyan tároló előállítására, amely a beírási folyamat alatt sem átlátszó. Az ilyen tárolókkal ismerkedünk meg a következő pontokban.

## 2.2.MESTER-SZOLGA TÁROLÓK (FLIP-FLOPOK)

### 2.2.1. A D-MESTER-SZOLGA tároló (flip-flop)

Kapcsoljunk össze két **D-G** tárolót a *2.14. ábra* szerinti módon, és az így létrehozott egység bemenetét jelöljük **D**-vel, kimenetét **Q**-val. Azt a bemenetet, amelyről az első tároló **G** bemenetét vezéreljük, és amelynek negált változója a másik tároló beírását vezérli, speciális funkcióval ruházzuk fel: **ÓRA (CLOCK, CLK)** lesz a neve, illetve funkciója. A működést analizálva beláthatjuk, hogy az **ÓRA** magas értékénél a **D** bemenet szintje beíródik az első **D-G** tárolóba, de eközben a második tároló kimenete változatlan, hiszen a **G** bemenetére **0** jut. Az **ÓRA** lefutásakor az első fokozat átlátszatlanná válik, a kimenetének értéke azonban átíródik a második tároló kimenetére. A beírás egy óra-ciklussal megtörtént, de úgy, hogy a teljes tároló ezalatt átlátszatlan maradt, hiszen egyik fokozata mindkét órajel-helyzetben átlátszatlan volt. Az ilyen két ütemben beírható tárolókat nevezzük

**MESTER-SZOLGA (MASTER-SLAVE)** tárolóknak, mivel az első fokozatba beírt értéket a második szolgai módon bemásolja.




*2.14. ábra. A D-MS tároló megvalósítása D-G tárolókból*

A **D-MS** tároló működését tehát az órajel két fázisra bontja:

1. A **D** bemenet mintavételezése és a mintavételezett érték tárolása, miközben a **Q** kimenet változatlan, őrzi az utolsóként beállt értéket.

2. A **Q** kimenetre a mintavételezett érték rákapcsolása és tárolása, miközben a **D** bemenet változásai már hatástalanok maradnak.

Az analízis eredményeképpen felvehetjük a **D-MS** tároló egyszerű igazságtábláját. Az ilyen órajel-vezérelt tárolóknál a következő kimeneti állapot jelölésére az (**n+1**) felső indexet szokás alkalmazni, mivel az aktuális kimeneti állapotot az n. órajel-ciklus eredményének tekintjük, és így **n** felső indexet alkalmazunk a jelölésre. Azaz az aktuális kimenet jele $\mathbf{Q^n}$, a következőé $\mathbf{Q^{n+1}}$.
Feltűnő, hogy a **D-MS** tároló következő kimeneti állapota nem függ az aktuális kimeneti állapottól, csakis a **D** bemenettől. Az összetett igazság-tábla tehát azonos az egyszerűvel.(*2.15. ábra*)

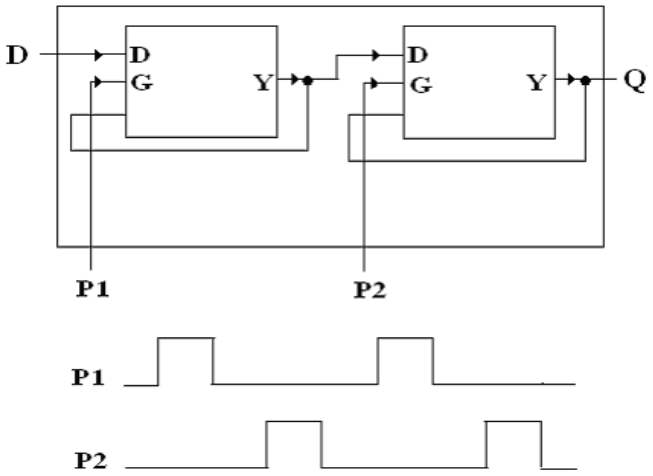
*egyszerű igazságtábla*

| D | $Q^{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

*2.15. ábra. A D-MS tároló igazságtáblája*

### 2.2.2. A D-MS tároló kétfázisú órajellel

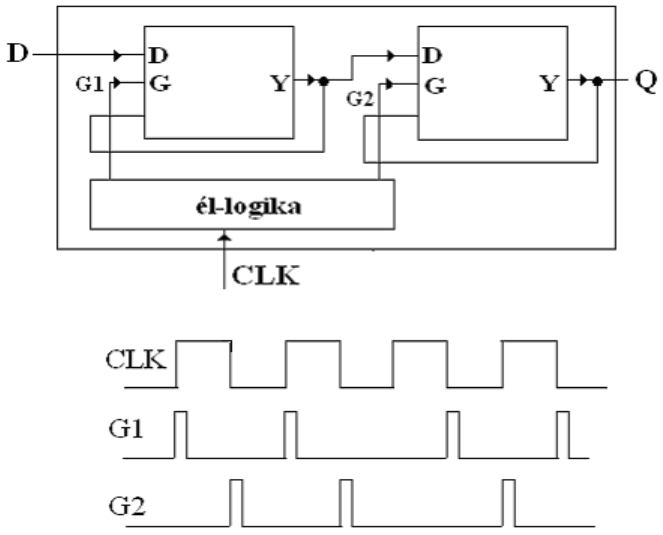
Ugyanez a tároló másféle, az átlátszatlanságot biztonságosabban garantáló órajel elrendezéssel is megvalósítható. A *2.16. ábrán* látható kétfázisú, nem átlapolt órajelekkel a két komponens-tároló működése időben egymástól biztonságosan elválasztható.




*2.16. ábra. Kétfázisú D-MS*

### 2.2.3. Az élvezérelt D-MS tároló *(2.17. ábra)*

Hasonlóan biztonságos időbeli elválasztásra törekedtek az élvezérelt **MESTER-SZOLGA** tárolók kialakítása során. Ennek azonban az az előnye, hogy csak egyfázisú órajelet igényel. A megoldás az, hogy az *órajel felfutó élére* az egyik, a *lefutó élre a* másik tároló működik. A két tároló **G** beíró-jelének kialakítása az áramkörön belül speciális él-logikát igényel.




*2.17. ábra. Élvezéret D-MS*

### 2.2.4. A J-K-MS tároló

A **D-MS** tárolóból kiindulva valósítsuk meg a **J-K-MS** tárolót. A kimenet visszacsatolása a bemenetre nem okozhat instabilitást, hiszen a **D-MS** tároló nem átlátszó, így minden egyes állapotváltás az órajel ütemében megy végbe. A **J-K-MS** egyszerű és összetett igazságtáblái a *2.18.a.* és *b.* ábrákon láthatók

*egyszerű igazságtábla*

| J | K | $Q^{n+1}$ |
|---|---|---|
| 0 | 0 | $Q^n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\overline{Q^n}$ |

*2.18.a. ábra. A J-K-MS egyszerű igazságtáblája*

*összetett igazságtábla*

| J | K | $Q^n$ | $Q^{n+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

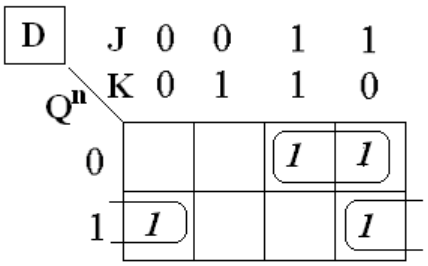
*2.18. b. ábra. A J-K-MS összetett igazságtáblája*

A **D-MS** tárolót a következő módon használjuk fel: tudjuk, hogy az órajel felfutása előtt a **D** bemenetre kell kapcsolnunk azt a logikai szintet amit a lefutáskor, mint a következő kimeneti állapotot, a kimeneten látni kívánunk. Ez azt jelenti, hogy egy olyan kombinációs hálózatot kell tervezni a **J, K** bemenetek és a **D** közé, amelynek igazság-tábláját a **J-K** tároló összetett táblájából könnyűszerrel megkapunk, ha a $\mathbf{Q^{n+1}}$-t **D**-re cseréljük (*2.19. ábra*).

| J | K | $Q^n$ | D |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

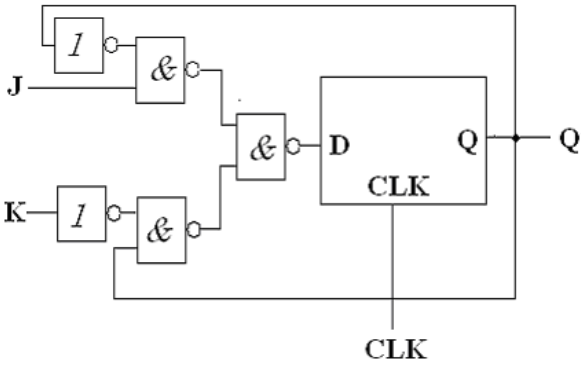
*2.19. ábra. A D-bemenetre érvényes igazságtábla*

Ezután a *2.20. ábrán* látható **K**-tábla segítségével realizálható a **D**-t meghajtó hálózat (*2.21 ábra*).




$$D = J\overline{Q^n} + \overline{K}Q^n$$

*2.20. ábra. A J-K MS K táblája és lefedése*



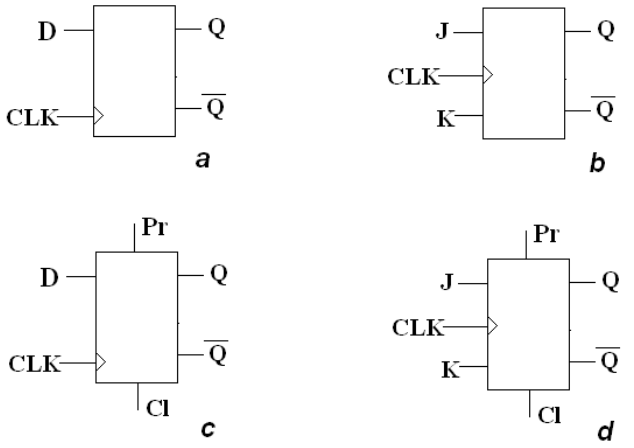

*2.21. ábra. A J-K-MS megvalósítása D-MS felhasználásával*

Megjegyezzük, hogy a K-tábla statikus hazárdot mutat. Mégsem kell hazárdmentesítés, mivel előírhatjuk, hogy az órajelet csak akkor lehet felemelni, ha a ***D***-t meghajtó hálózat tranziensei befejeződtek, és ***D*** már nyugalmi állapotba került.

### 2.2.5. Flip-flopok segéd-bemenetei és szimbólumaik

A *2.22. ábrán* bemutatjuk azokat a szimbólumokat, amelyekkel az élvezérelt **MS t**árolókat jelölni szokták. Tudnunk kell, hogy mindkét fajta flip-flopot

gyakran kiegészítik két bemenettel. A **PRESET (Pr)** bemenet a tárolót a funkcionális bemenetektől függetlenül **1**-be, a **CLEAR (Cl)** a funkcionális bemenetektől függetlenül **0**-ba állítja. Ezekkel a bemenetekkel igen egyszerűen állíthatjuk be a szinkron sorrendi hálózatok kezdeti állapotát.




*2.22. ábra. Preset és Clear bemenetekkel nem, és azokkal is rendelkező, élvezérelt flip-flopok szimbólumai*

*TERVEZÉSI FELADATOK 2.*

**1**. Szinkron, él-vezérlésű **Master-Slave** ***D*** flip-flopunkból olyan új szinkron flip-flopot szeretnénk építeni, amelynek két bemenete (***A, B***) van. Ha a bemenetek ellentétesek, akkor a tároló kimenete váltson értéket, míg ha egyformák, akkor maradjon meg az aktuális állapot a kimeneten.
Tevezzük meg a hálózatot ***NAND*** kapukkal!

**2**. Szinkron, él-vezérlésű **Master-Slave** ***D*** flip-flopunkból olyan új szinkron flip-flopot szeretnénk építeni, amelynek két bemenete (***A, B***) van. Ha a bemenetek egyformák, akkor a tároló kimenete váltson értéket, míg ha ***A = 1*** és ***B = 0***, akkor a tároló ***1***-be, míg ha ***A = 0*** és ***B = 1, 0***-ba álljon.
Tevezzük meg a hálózatot ***NAND*** kapukkal!

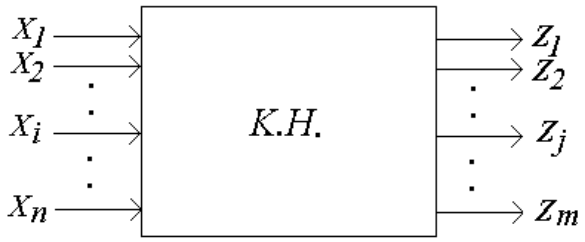
***3.*** Szinkron, él-vezérlésű **Master-Slave *J-K*** flip-flopunkból *vezérelhető funkciójú* flip-flopot építünk, mégpedig olyat, amely egy kombinált, ***DT*** bemenettel rendelkezik, és amely a ***V = 1*** vezérlő bemenet-szintről a normál ***D,*** a ***V = 0*** szintről pedig ***T*** flip-flopként működik. ( A ***T*** *f*lip-flop megváltoztatja a kimenet értékét, ha ***T = 1***, meghagyja, ha ***T = 0***. Tervezzük meg!

***4.*** Szinkron, él-vezérlésű **Master-Slave *D*-**flip-flopunkból kiválasztható bemenetekkel bíró flip-flopot építünk, mégpedig olyat, amely két , ***D1*** és ***D2*** bemenettel rendelkezik, és amely a ***V = 0*** vezérlő bemenet-szintről a ***D1*** a ***V = 1*** szintről pedig ***a D2*** bemenetről működő ***D -*** flip-flopként működik.

***5.*** Szinkron, él-vezérlésű **Master-Slave *J-K*** flip-flopunkból *vezérelhető* szinkron flip-flopot építünk, mégpedig olyat, amelynek egy ***D*** bemenete, és egy ***E*** engedélyező bemenete van. Ha ***E = 0***, a tároló kimenete nem változik, ha ***E = 1***, akkor a kimenet ***D*** értékét veszi fel.

## 2.3. SORRENDI HÁLÓZATOK MODELLJEI, ALAPTÍPUSAI

**2.3.1. A kombinációs hálózatok egy magasabb szintű modellje** *(2.23. ábra)*
A szekvenciális hálózatok tárgyalása előtt utalunk a kombinációs hálózatok már megismert „fekete-doboz" modelljére. A hálózatot jellemezhetjük az egyes kimenetekhez rendelt logikai függvényekkel, amelyeket a dobozon belül kapukkal realizálunk a megismert módon. A modell az időbeli, tranziens viselkedést nem írja le, de tudjuk, hogy a bemeneti változások hatása időkésleltetésekkel jelentkezik a kimeneteken.




*2.23. ábra. A kombinációs hálózat fekete-doboz modellje*

A modellt a következő logikai egyenletrendszerrel írhatjuk le:

$$
\begin{aligned}
Z_1 &= f_{z_1}(X_1, X_2, \ldots X_i \ldots X_n) \\
Z_2 &= f_{z_2}(X_1, X_2, \ldots X_i \ldots X_n) \\
&\ldots\ldots\ldots\ldots\ldots\ldots \\
Z_j &= f_{z_j}(X_1, X_2, \ldots X_i \ldots X_n) \\
&\ldots\ldots\ldots\ldots\ldots\ldots \\
Z_m &= f_{z_m}(X_1, X_2, \ldots X_i \ldots X_n)
\end{aligned}
$$

A fenti modell egy tömörebb megfogalmazása szerint az egyes kimenetekhez rendelt logikai függvények összességéből egy új függvényt konstruálva, és azt $f_z$-vel jelölve, továbbá bevezetve a bemeneti kombinációk halmazát (**X**) és a kimeneti kombinációk halmazát (**Z**), a kombinációs hálózat egy olyan leképezésként ragadható meg, amely a bemeneti kombinációk halmazát leképezi a kimeneti kombinációk halmazába.

$$f_z : \mathbf{X} \Longrightarrow \mathbf{Z}$$

$\mathbf{X}$ : *a bemeneti kombinációk halmaza*

$\mathbf{Z}$ : *a kimeneti kombinációk halmaza*

A másik, itt bemutatott jelölés a halmazok elemeire értelmezi az $f_z$ szerepét. Eszerint a bemeneti kombinációk halmazának elemeihez a kombinációs hálózat függvénye hozzárendel egy kimeneti kombinációt. Tudjuk, hogy a $t$ időpontban megváltoztatott bemeneti kombináció hatása valamilyen késleltetéssel jelenik meg a kimeneten, $t + \Delta t$ időpontban.

$$f_z : x_i \longrightarrow z_j$$

$x_i$ : *egy bemeneti kombináció*

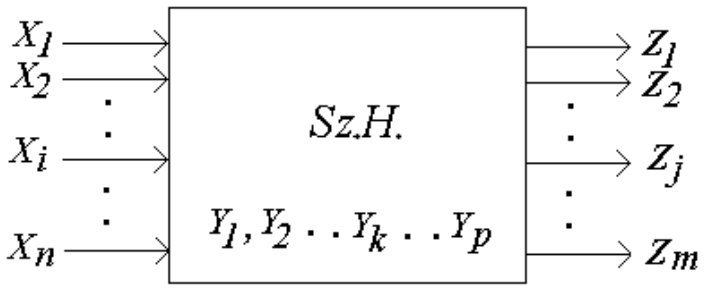
$z_j$ : *egy kimeneti kombináció*

*A kombinációs hálózat viselkedésének legfontosabb sajátossága, hogy egy meghatározott bemeneti kombináció ismételt rákapcsolásaira a tranziens idő eltelte után mindig ugyanazt a kimeneti kombinációt szolgáltatja, függetlenül attól, hogy az adott bemeneti kombináció két rákapcsolása között milyen más bemeneti kombinációkat kapcsoltunk a hálózatra.*

Ez az a tulajdonság, amely a kombinációs hálózatokat megkülönbözteti a sorrendiektől, illetve azokat ez utóbbiak speciális részhalmazává teszi.

### 2.3.2. A sorrendi (szekvenciális) hálózatok általános modelljei

A sorrendi hálózatok „fekete doboz" modellje formájában nem, csak viselkedésében különbözik a kombinációs modelltől. Ez azt jelenti, hogy a sorrendi hálózat ugyanarra a bemeneti kombinációra rendre más és más kimenő-kombinációt szolgáltathat a kimenetein. Másképpen megfogalmazva: a kimeneti kombináció nem csak a pillanatnyi bemeneti kombinációtól függ, hanem a korábbi bemeneti kombinációktól, sőt azok *sorrendjétől* is függ.
Ez csak úgy lehetséges, a kimeneti kombinációk nem csak a bemenetektől függenek, hanem a dobozon belül „elrejtett" *szekunder változóktól* ($\boldsymbol{Y_1 \ldots Y_p}$) is. A modellt leíró logikai függvények tehát összetettebbek. Két logikai függvénycsoportot kell megadnunk. Az első csoport a kimeneteket adja meg a bemenetek és a szekunder változók függvényében, a másik az állapotváltozók új értékeit határozzák meg a bemenetek és az éppen fennálló (aktuális) szekunder változó értékek alapján. A *2.24. ábra* mutatja a modellt.




*2.24. ábra. A szekvenciális hálózat modellje*

Ha a szekunder változók új értékeit felső-csillaggal különböztetjük meg az aktuálisaktól, a modell a következő logikai egyenletrendszerrel írható le:

$$
\begin{aligned}
Y_1^* &= f_{y_1}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p) \\
Y_2^* &= f_{y_2}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p) \\
&\cdots\cdots\cdots\cdots\cdots\cdots \\
Y_k^* &= f_{y_k}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p) \\
&\cdots\cdots\cdots\cdots\cdots\cdots \\
Y_p^* &= f_{y_p}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p)
\end{aligned}
$$

$$
\begin{aligned}
Z_1 &= f_{z_1}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p) \\
Z_2 &= f_{z_2}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p) \\
&\cdots\cdots\cdots\cdots\cdots\cdots \\
Z_j &= f_{z_j}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p) \\
&\cdots\cdots\cdots\cdots\cdots\cdots \\
Z_m &= f_{z_m}(X_1 \ldots X_i \ldots X_n, Y_1 \ldots Y_k \ldots Y_p)
\end{aligned}
$$

A függvények általánosításával formailag ez a modell is egyszerűsíthető. Két függvényt kell definiálni. Az első, amit *kimeneti függvénynek* nevezünk a bemeneti kombinációk halmazának és a szekunder változók kombinációiból álló halmaznak a szorzatát képezi le a kimeneti kombinációk halmazába. A második ugyanezt a szorzat halmazt a szekunder kombinációk halmazába képezi le. A szekunder változók itt kihasznált kombinációit a hálózat *belső állapotainak*, röviden *állapotainak* nevezzük. Az **Y** halmaz neve így *állapothalmaz*.

$$
\begin{aligned}
f_y &: \mathbf{X} \times \mathbf{Y} \Longrightarrow \mathbf{Y} \\
f_z &: \mathbf{X} \times \mathbf{Y} \Longrightarrow \mathbf{Z}
\end{aligned}
$$

**X** : *a bemeneti kombinációk halmaza*

**Z** : *a kimeneti kombinációk halmaza*

**Y**: *a szekunder-változó kombinációk halmaza*

Ha a függvényeket a halmazok elemeire értelmezzük, akkor a viselkedésnek egy finomabb leírását kapjuk. A kimeneti függvény megvalósítása egy olyan kombinációs hálózat, amelynek bemeneteire a hálózat bemenetei és az állapotváltozók csatlakoznak, tehát a bemeneti-kombináció és állapot-kombináció párok alkotnak egy bemeneti kombinációt az $f_z$ hálózaton. Ha ezeket egy $t$ időpontbeli értékkel jellemezzük, akkor a hálózat kimenetein valamilyen tranziens után előállnak a hozzárendelt kimeneti kombinációk. Ugyanakkor ugyanez a páros a másik, $f_y$ hálózat bemeneteire érkezve egy másik tranziens után egy új, $t+\Delta t$ időpontbeli állapotot hoz létre, amely visszakerül az $f_y$-val jellemzett hálózat bemeneteire.

$$f_y : (x_i^t, y_k^t) \longrightarrow y_l^{t+\triangle t}$$
$$f_z : (x_i^t, y_k^t) \longrightarrow z_j^t$$

$x_i^t$ : *bemeneti kombináció a t időpontban azaz, a bemeneti kombináció értéke a t időpontban* $x_i$

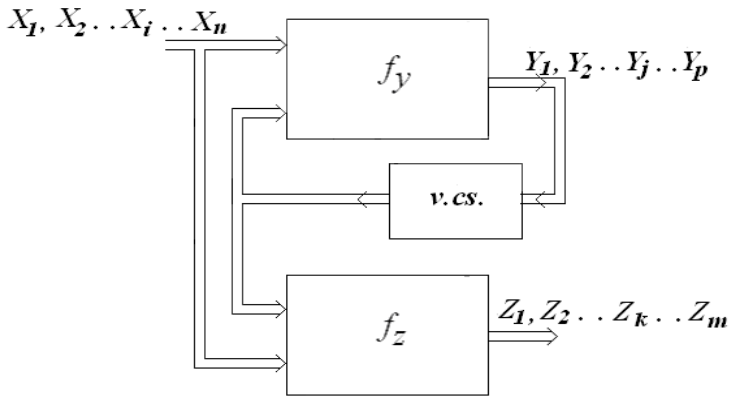
$y_k^t$ : *szekunder változó kombináció a t időpontban*

$z_j^t$ : *kimeneti kombináció a t időpontban*

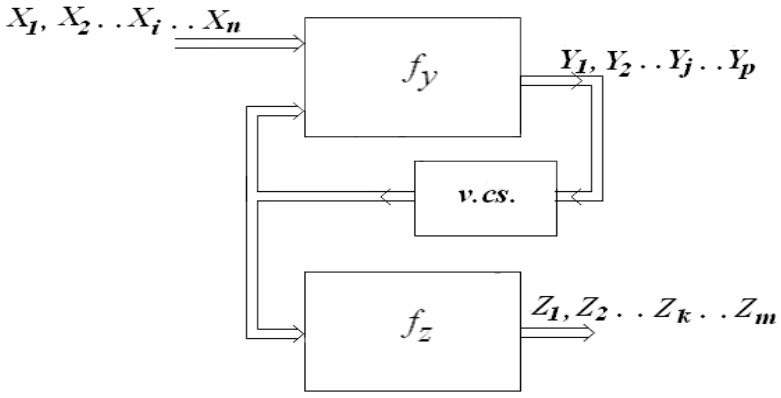
$y_l^{t+\triangle t}$ : *szekunder változó kombináció a* $(t+\triangle t)$ *időpontban*

### 2.3.3. Mealy-típusú sorrendi hálózat.

A sorrendi hálózat belső struktúráját mutatja a *2.25.a. ábra*. A struktúra az előző pontban bemutatott függvény-kapcsolatokat is tükrözi. Azt a sorrendi hálózatot, amelynek $f_z$ kimeneti hálózatára – az egyenletekkel is bemutatott modellnek megfelelően - mind a bemenetek, mind az állapot-változók rácsatlakoznak, *Mealy-típusú* hálózatnak nevezzük.




*2.25. a. ábra. A Mealy típusú sorrendi hálózat struktúrája*




*2.25. b. ábra. A Moore-típusú sorrendi hálózat struktúrája*

### 2.3.4. Moore-típusú sorrendi hálózat *(2.25. b. ábra)*

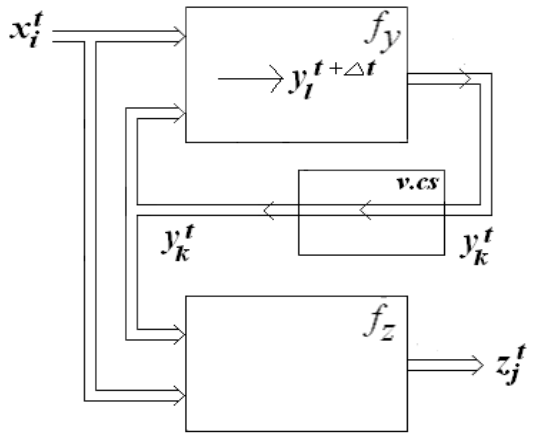
Azt a speciális sorrendi hálózatot, amelynek kimeneti kombinációira csak a belső állapotok hatnak, *Moore-típusú* sorrendi hálózatnak nevezzük. Azt is mondhatjuk, hogy a *Moore*-féle sorrendi hálózatban a kimeneti kombinációk a belső állapotok átkódolt formáit szolgáltatja a kimeneteken.

### 2.3.5. Aszinkron sorrendi hálózat

Az aszinkron sorrendi hálózat $f_y$ hálózatának visszacsatolása órajel nélküli, direkt visszacsatolás. Ezt a direkt visszacsatolást vagy közvetlenül, huzalozással hozzuk létre, vagy **S-R** tárolókat helyezünk el a visszacsatoló körben. Mindkét módszer közös sajátossága, hogy a visszacsatolás a hálózat saját késleltetési idejének megfelelően érvényesül. Egy stabil $y_j$ állapot adott $x_i$ bemeneti kombinációra csak akkor áll be, ha az $f_y$ leképezésre igaz, hogy

$$f_y(x_i, y_j) = y_j$$

#### 2.3.5.1. Közvetlen visszacsatolású aszinkron sorrendi hálózat



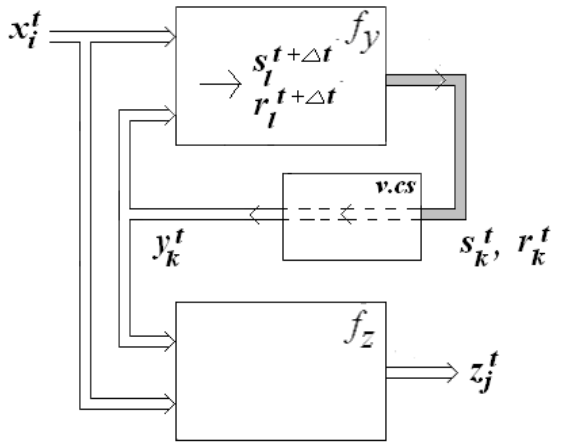

*2.26. ábra. Közvetlen visszacsatolású aszinkron sorrendi hálózat*

A *2.26. ábra* a közvetlenül, vezetékekkel visszacsatolt aszinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor egy új bemeneti kombinációt már rákapcsoltunk a hálózatra, és az $f_y$ kombinációs hálózat belsejében már megindult a *következő állapot kombináció* generálása. Az $f_y$ hálózat kimenetein azonban a változás még nem jelent meg, az egyenlőre még az *aktuális állapot kombinációt* őrzi. Az $f_z$ kimenetein ugyancsak átmeneti állapot van, hiszen a bemeneti kombináció már megváltozott, az aktuális állapot-kombináció ugyanakkor még uralkodik a bemenetein.

Tegyük fel, hogy a következő állapot kombináció az új bemeneti kombinációval olyan párost alkot az $f_y$ bemenetein, amelyhez az $f_y$ hálózat az ugyanezt a következő állapot-kombinációt rendeli. Ez azt jelenti, hogy az új állapot-kombináció stabilizálódik. Így végül a kimeneti kombináció is nyugalomba jut, a stabil új aktuális állapothoz és az eseménysort kiváltó bemeneti kombinációhoz rendelt $f_z$ érték jelenik meg a kimeneteken. Ha ezt követően megváltoztatjuk a bemeneti kombinációt, új tranziens indul meg.

### 2.3.5.2. S-R tárolókkal visszacsatolt aszinkron sorrendi hálózat

A *2.27. ábra* az **S-R** tárolókkal visszacsatolt aszinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor egy új bemeneti kombinációt már rákapcsoltunk a hálózatra, és az $f_y$ kombinációs hálózat belsejében már megindult a *következő állapotot generáló* ***S-R*** *kombináció* kialakulása. Az $f_y$ hálózat kimenetein azonban a változás még nem jelent meg, az egyenlőre még az *aktuális állapotot generáló* ***S-R*** *kombinációt* őrzi. Az $f_z$




*2.27. ábra. Az S-R tárolókkal visszacsatolt aszinkron sorrendi hálózat*

kimenetein ugyancsak átmeneti állapot van, hiszen a bemeneti kombináció már megváltozott, az aktuális állapot-kombináció ugyanakkor még uralkodik a bemenetein. Tegyük fel, hogy a kialakuló következő állapot az új bemeneti kombinációval olyan párost alkot az $f_y$ bemenetein, amelyhez az $f_y$ hálózat az

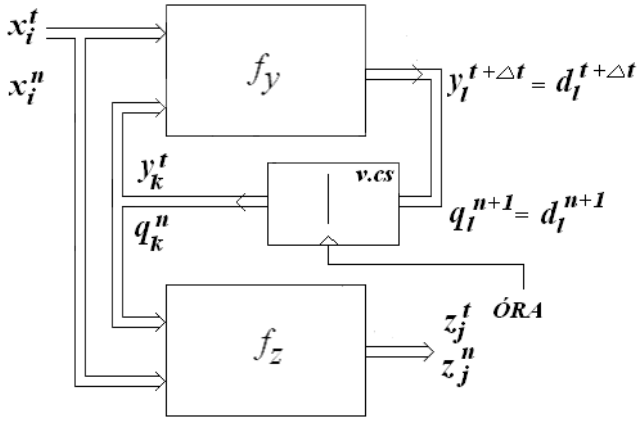
ugyanezt a következő állapot-kombinációt generáló **S-R** kombinációt rendeli. Ez azt jelenti, hogy az új állapot kombináció stabilizálódik. Így végül a kimeneti kombináció is nyugalomba jut, a stabil új aktuális állapothoz és az eseménysort kiváltó bemeneti kombinációhoz rendelt $f_z$ érték jelenik meg a kimeneteken. Ha ezt követően megváltoztatjuk a bemeneti kombinációt, új tranziens indul meg.

### 2.3.6. Szinkron sorrendi hálózat

A szinkron sorrendi hálózat $f_y$ hálózatának visszacsatolása órajellel, **MS** tárolókon keresztül érvényesül. A gyakorlatban vagy **D-MS**, vagy **J-K-MS** tárolós visszacsatolást használnak. Mindkét módszer közös sajátossága, hogy az órajel minden állapot kombináció fenállásának idő-intervallumát egyértelműen kijelöli, függetlenül attól, hogy az $f_y$ visszaadja-e a rákapcsolt aktuális állapot-kódot, vagy nem. Így feleslegessé válik az instabil és a stabil állapotok megkülönböztetése.
A szinkron hálózatok az egymást követő állapotokat és kimeneti kombinációkat időben diszkrét sorozatokká alakítja. Ennek megfelelően sajátos jelöléseket alkalmazhatunk, a $t$ időpont-beli kombinációkat inkább az $n$ természetes számmal, a $t+\Delta t$ időpont-beli kombinációkat inkább az $n+1$ természetes számmal, mint sorszámokkal jelöljük. A szinkron tárolók kimeneti kombinációinak jelölésére inkább a már megismert $q$ szimbólumot használjuk. Megjegyezzük, hogy az ábrákon mindkét jelölés-rendszer látható, de a későbbiekben szinkron hálózatok esetén csak a most bevezetett jelölés-rendszert alkalmazzuk.

#### 2.3.6.1. D-MS flip-flopokkal visszacsatolt szinkron sorrendi hálózat

A *2.28. ábra* egy **D-MS** flip-flopokkal visszacsatolt szinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor az ***(n-1)***. órajel-ciklus már lejátszódott, és az ***n.*** felfutó élre várunk. Ennek megfelelően a hálózatra már rákapcsoltuk az ***n***. ütemnek megfelelő bemeneti kombinációt, és megjelent az ennek megfelelő kimeneti kombináció is.

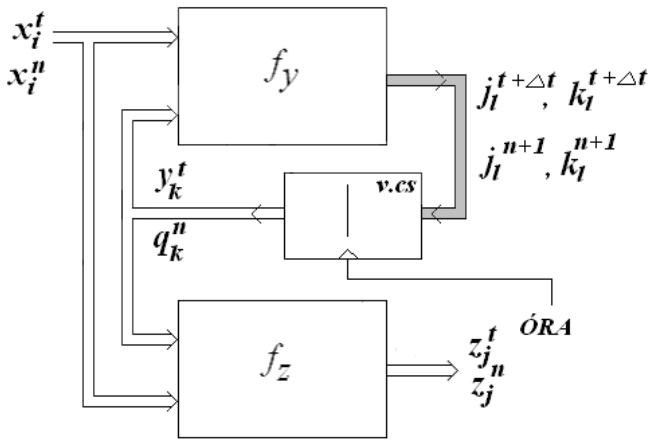



*2.28. ábra. D-flip-flopokkal visszacsatolt szinkron sorrandi hálózat*

Az $f_y$ kombinációs hálózat a kimenetein előállítja a *tárolók bemeneteinek kombinációját.* Ezek azonos kombinációk a megkívánt *következő állapot kombinációkkal*, hiszen a **D** típusú tárolók kimeneteiken megismétlik a bemeneteikre kerülő értékeket. Míg egy adott $d_l^{n+1}$ kombináció a visszacsatoló flip-flopok bemenetein várakozik, az $f_y$ hálózatra csatlakozó kimeneteik változatlanok, és az *aktuális állapot-kombinációt*, a $q_k^n$-t őrzik. Az $f_z$ kombinációs hálózat a bemenetére jutó aktuális állapot-kombináció és a bemeneti kombináció eredményeképpen szolgáltatja az *aktuális kimeneti kombinációt*. Ekkor érkezik az órajel felfutó éle. A következő állapotkód a **MESTER** tárolókba kerül, miközben a **SZOLGA** kimenetek változatlanok.
A következő változás akkor következik be, amikor az órajel lefutó éle megérkezik. Ennek hatására a **D-MS** tárolók kimenetein megjelenik az eddig *következő állapot-kombinációnak* nevezett kombináció, és aktuális állapottá válik. Ha ezt követően megváltoztatjuk a bemeneti kombinációt, akkor az $f_z$ kimenetein egy új aktuális kimeneti kombináció, és az $f_y$ kimenetein egy újabb következő állapot kódja jelenik meg.

### 2.3.6.2. J-K-MS flip-flopokkal visszacsatolt szinkron sorrendi hálózat

A *2.29. ábra* **J-K-MS** flip-flopokkal visszacsatolt szinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor az ***(n-1)***. órajel-ciklus már lejátszódott, és az ***n***. felfutó élre várunk. Ennek megfelelően a hálózatra már rákapcsoltuk az n. ütemnek megfelelő bemeneti kombinációt, és megjelent az ennek megfelelő kimeneti kombináció is.




*2.29. ábra. J-K flip-flopokkal visszacsatolt szinkron sorrendi hálózat*

Az $f_y$ kombinációs hálózat a kimenetein előállítja a *tárolók bemeneteinek kombinációját.* Ezek nem azonos kombinációk a megkívánt *következő állapot kombinációkkal*, hiszen a **J-K** tárolók bemeneteire olyan kombinációkat kell adnunk, amelyek a *következő állapot kombinációkat* majd az órajel ciklus lejátszódásakor generálják. Míg egy adott $j_l^{n+1}$ , $k_l^{n+1}$ kombináció a visszacsatoló flip-flopok bemenetein várakozik, az $f_y$ hálózatra csatlakozó kimeneteik változatlanok, és az *aktuális állapot-kombinációt*, a $q_k^n$-t őrzik. Az $f_z$ kombinációs hálózat a bemenetére jutó aktuális állapot-kombináció és a bemeneti kombináció eredményeképpen szolgáltatja az *aktuális kimeneti kombinációt*. Ekkor érkezik az órajel felfutó éle. A következő állapotkód a **MESTER** tárolókba kerül, miközben a **SZOLGA** kimenetek változatlanok.
A következő változás akkor következik be, amikor az órajel lefutó éle megérkezik. Ennek hatására a **J-K MS** tárolók kimenetein megjelenik az eddig *következő állapot-kombinációnak* nevezett kombináció, és aktuális állapottá válik. Ha ezt követően megváltoztatjuk a bemeneti kombinációt, akkor az $f_z$ kimenetein egy új aktuális kimeneti kombináció, és az $f_y$ kimenetein egy újabb következő állapot kódja jelenik meg.

## 2.4. SZINKRON HÁLÓZATOK TERVEZÉSI FOLYAMATA MINTA-PÉLDÁKON

### 2.4.1. Az első szinkron hálózattervezési minta-feladat

#### 2.4.1.1. A minta-feladat megfogalmazása

*Egy hálózatra egy órajel ütemében az **X1, X2** jelek érkeznek.*
*A hálózat az első **X1 = X2** bemeneti kombinációtól kezdve vizsgálja a bemeneteket, és a **Z** kimenetén jelzi, ha a két bemenet kétszer egymás után azonos logikai szintű. Ha ilyen kombináció-sorozat lezajlott, a vizsgálatot újra kezdi. Tervezzük meg a hálózatot **J-K-MS** tárolókkal!*

#### 2.4.1.2. Egy MEALY modell felvázolása állapot-átmeneti gráffal és/vagy előzetes állapot-átmeneti táblával

A verbálisan megadott szinkron hálózattervezési feladat megoldásának első lépése, hogy a működést megpróbáljuk egy állapot-átmeneti gráfon, majd egy állapot-átmeneti táblán is megfogalmazzuk. Meg kell jegyeznünk, hogy a szóbeli specifikáció minden többértelműségét csak ezek a módszerek tisztázhatják. Értelmezzük a feladatot úgy, ahogyan azt az alábbi gráf, illetve táblázat tartalmazza.

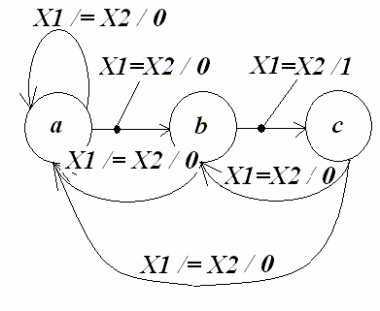
**2.4.1.2.1. Állapot-átmeneti gráf (***2.30. a. ábra***)**

Az állapot-átmeneti gráf, röviden állapotgráf csomópontjai szimbolikus (szimbólumokkal jelölt) állapot-kombinációk, röviden állapotok, élei a bemeneti kombinációk. *Mealy*-típusú állapot-gráfon minden élhez hozzárendeljük a kimeneti kombinációt is. Az állapot-gráf tehát megmutatja, hogy a hálózat adott aktuális állapotból egy adott bemeneti kombináció rákapcsolása után a következő órajel-ciklus során melyik következő állapotba kerül, és az adott aktuális állapothoz adott bemeneti kombináció rákapcsolásakor milyen kimeneti kombináció jelentkezik a kimeneteken. Ez utóbbiakat „/” jellel választjuk el a következő állapot szimbólumától. A *Moore*-típusú állapot-gráf ettől abban különbözik, hogy a gráf csomópontjaihoz, azaz az állapotokhoz rendeljük hozzá a kimeneti kombinációkat.

**2.4.1.2.2. Előzetes, szimbolikus állapottábla** (*2.30. b. ábra*)

Az előzetes szimbolikus állapottáblában ugyanazokat az információkat rögzítjük, mint az állapot-gráfon. Ennek szerkezetét a tárolók, illetve flip-flopok tervezéséből már ismerjük. Ez azonban az állapotokat absztrakt formában tartalmazza, a konkrét állapot-kombinációk megállapítását, azaz az állapot-kódolást később végezzük el. A állapottáblában tehát bemeneti-

kombinációnként minden állapothoz megadjuk a következő állapot szimbólumát, és a „/” jellel elválasztva a megkívánt kimeneti kombinációt.
Mivel a továbbiakban az állapottáblával dolgozunk tovább, az állapot-gráf felvétele kihagyható, ha azonnal fel tudjuk venni az állapottáblát.




*2.30. a. ábra. A specifikáció állapotgráffal*

| akt. áll. | köv.állapot / kimenet | |
|---|---|---|
| | *X1/=X2* | *X1=X2* |
| *a* | *a/0* | *b/0* |
| *b* | *a/0* | *c/1* |
| *c* | *a/0* | *b/0* |

*2.30. b. ábra. A specifikáció szimbolikus állapottáblával*

**2.4.1.2.3. A feladat állapotgráfja és előzetes, szimbolikus állapottáblája**
Feladatunk megoldásához ezúttal Mealy-típusú gráfot választottunk, később megmutatjuk e feladat megoldását Moore-típusú hálózattal is. Úgy képzeljük, hogy hálózatunk bekapcsolás után egy ***a*** jelű kezdeti állapotba kerül. Az ezután jelentkező első órajel-ciklus eredménye attól függ, milyen bemeneti kombináció érkezik. Ha **X1 /= X2** ( **01** vagy **10**) , ez nem kedvező számunkra, hiszen hálózatunknak a második egyforma szint-állást kell detektálnia, ehhez pedig az első egyforma szint-állásnak előbb meg kell érkeznie. Tehát, mindaddig amíg **01** vagy **10** érkezik a bemenetre, maradunk a kezdeti állapotban, és a kimenet, **Z** alacsony szinten marad. Azt viszont, hogy megérkezik az első **00** vagy **11**, a hálózatnak *meg kell jegyeznie*, hogy a második együttállást jelezni tudja. Tehát az ***a*** állapotból a **00** vagy az **11** bemeneti kombinációkra a ***b*** állapotba kerül a hálózat, és a ***b*** szimbólum jelentése, hogy megjött az első, számunkra kedvező bemeneti kombináció. Persze az ***a***-ból ***b***-be tartó átmenet alatt **Z=0** marad, hiszen ez még csak az első kedvező bemeneti kombináció.
Ha hálózatunk a ***b*** állapotban van, és a következő órajel-ciklust **01** vagy **10** bemeneti kombináció fogadja, akkor hálózatunkat reményt vesztve vissza kell irányítani a kezdeti a állapotba, a **Z** nullán tartása mellett. Ezzel szemben, ha a **00** vagy az **10** érkezik, hálózatunk a ***c*** állapotba kerül, és a **Z = 1** értékkel jelzi, hogy megjött a második együttállás.

A ***c*** állapotból való elágazásnál, ha **01** vagy **10** érkezik, a kezdőállapotba kell mennünk, hiszen újra az első együttállásra kell várni. Jöhet azonban **00** vagy **11** is, és akkor máris a ***b*** állapotba kell mennünk. Mindkét esetben persze a **Z = 0.**

Az előzetes, szimbolikus állapottáblába csak azokat az információkat rögzítjük, amelyeket az állapot-gráffal megadtunk.

Megjegyezzük, hogy ennél a feladatnál egyetlen gráf-él, vagy a tábla egyetlen oszlopa lényegében két bemeneti kombinációt képvisel. Természetesen megadható lett volna mindkettő állapotonként négy éllel illetve négy oszloppal is, de célszerű kihasználni az ilyen és ehhez hasonló egyszerűsítési lehetőségeket.

### 2.4.1.3. Bemeneti egyszerűsítési lehetőségek kihasználása

A fentiek alapján felismerjük, hogy a hálózat elágazásait tulajdonképpen egyetlen jel vezérelheti. Bevezetjük tehát az ***e*** *l*ogikai változót, amelynek értéke az **X1** és **X2** együttállása esetén **1**, egyébként **0**. Ez egy ekvivalencia kapu, vagy más néven **KIZÁRÓ-NEM-VAGY (EXNOR)**. Tehát legyen**:**

$$e = X_1 X_2 + \overline{X_1}\,\overline{X_2}$$

### 2.4.1.4. A feltétlenül szükséges számú állapot megállapítása. (Állapotminimalizálás)

Amikor a fent bemutatott módon az állapot-gráfot szerkesztjük, akkor könnyen lehet, hogy akkor is új állapotot veszünk fel, amikor pedig egy már meglévőt is felhasználhatnánk. Legtöbbször éleslátás vagy gyakorlat kérdése, hogy elsőre felismerjük-e a feltétlenül szükséges állapotokat. Ne aggasszon azonban bennünket ez a dolog, hiszen az első, előzetes állapottábla állapotainak számát szisztematikus módszerekkel minimalizálni fogjuk. Ennek általános megfontolásaival egy későbbi fejezetben részletesen foglalkozunk, most egy magától értetődő kritériumot fogalmazunk meg arra, mikor nem kell megkülönböztetni két állapotot az előzetes állapottáblán. Ez a következő:

*Az előzetes állapottábla két állapotát nem kell megkülönböztetni, ezért azok összevonhatók, ha bemeneti kombinációnként egyeznek a hozzájuk rendelt kimeneti kombinációk, és bemenő kombinációnként ugyanarra a következő állapotra vezetnek.*

### 2.4.1.5. Állapot-összevonás az adott feladatban

Az állapottábla tüzetes vizsgálatából kiderül, hogy a fenti feltétel az **a** és a **c** állapotokra teljesül. Mindkettőből kiinduló bemeneti kombinációkhoz

ugyanazok a kimeneti értékek tartoznak, hiszen **e = 0**-nál **a**-ból **Z = 0, e = 1** esetben ugyancsak, és egyformák a **Z** értékek az **e = 1** bemenetnél is. A következő állapotok az **e = 0**-nál mindkettőre **a**, és **e = 1**-re mindkettőre **b**.
Az **a** és **c** állapot megkülönböztetése tehát felesleges, azokat össze lehet vonni. Jelöljük az új, összevont állapotot **ac**-vel.

### 2.4.1.6. Az összevont állapottábla

Új állapottáblát szerkesztünk, most már a feltétlenül szükséges állapotokkal. Ezt összevont szimbolikus állapottáblának nevezzük. Az összevont állapotok kerülnek bejegyzésre mindazokon a helyeken, ahol az előzetes táblában az összevont állapotok valamelyike szerepelt.

### 2.4.1.7. A kódolt állapottábla

Az összevont szimbolikus tábla állapotait bináris kódokkal, azaz állapot kombinációkkal kell ábrázolni. Itt a választott kód hosszúsága egyértelműen meghatározza a visszacsatoló **MS** tárolók számát. Természetesen legtöbbször minimális számú **MS** tároló alkalmazására törekszünk, ezért a következő összefüggés alapján kódoljuk az állapotokat:

$$N_{MS} \geq log_2 N_a$$

Itt $N_{MS}$ az állapot-kód hossza, azaz a szükséges **MS** tárolók száma**,** $N_a$ **pedig** a kódolt összevont állapottáblán az állapotok száma**.**

### 2.4.1.8. A vezérlési tábla

A tervezési folyamatnak ezen a pontján általában döntenünk kell, milyen flip-flopokkal valósítjuk meg a hálózatot. A **D-MS** tárolókkal való megvalósítás kevesebb tervezői munkával jár, hiszen minden flip-flopnak csak egyetlenegy bemenete van, ugyanakkor a kiadódó kombinációs hálózat általában bonyolultabb, mint **J-K MS** tárolók alkalmazásakor. Ez az előny illetve hátrány persze erősen feladat-függő. Ezért javasolható mindkét flip-flop típus kipróbálása, és egy jól megválasztott költség-függvény szerinti összehasonlítás. A vezérlési táblák megalkotásához szükségünk van a flip-flopok *vezérlési tábláira,* nevezetesen arra, hogy adott kimeneti változáshoz milyen szinteket kell kapcsolnunk a bemenetekre. A *2.31. ábra* mutatja a **D-MS** tároló vezérlési tábláját a már ismert összetett igazságtáblával együtt, abból levezetve. A vastag határvonalakkal feltüntetett táblázat baloldala mutatja a flip-flop kimenetének lehetséges változásait, a jobboldali oszlop pedig azt, hogy az adott változás végbemeneteléhez milyen logikai értéket kell a **D** bemenetre kapcsolni. Nyilván ez igen egyszerű, hiszen mindig a megkívánt új értékkel azonos értéket kell a **D**-re kapcsolnunk.

Kicsit bonyolultabb a **J-K MS** tároló vezérlési táblájának megszerkesztése. A *2.32. ábra* mutatja ezt, ugyancsak az összetett igazságtáblából levezetve. Láthatjuk, hogy minden lehetséges változáshoz van egy olyan bemenet a kettő közül, amelynek szintje lehet **0** és lehet **1** is, azaz közömbös. Ezek a *don't care* bejegyzések teszik népszerűvé a **J-K** flip-flopot, hiszen a következő állapot kombinációt generáló kombinációs hálózat egyszerűsítéséhez ezek jótékonyan járulnak hozzá.

| D | $Q^n$ | $Q^{n+1}$ | $Q^n \rightarrow Q^{n+1}$ | | D |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

*2.31. ábra. A D-MS tároló vezérlési táblája*

| J | K | $Q^n$ | $Q^{n+1}$ | $Q^n \rightarrow Q^{n+1}$ | | J | K |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | — |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | — |
| 0 | 1 | 0 | 0 | 1 | 0 | — | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | — | 0 |
| 1 | 0 | 0 | 1 | | | | |
| 1 | 0 | 1 | 1 | | | | |
| 1 | 1 | 0 | 1 | | | | |
| 1 | 1 | 1 | 0 | | | | |

*2.32. ábra. A J-K MS tároló vezérlési táblája*

### 2.4.1.9. A feladat összevont szimbolikus állapottáblája és kódolt állapottáblája, valamint J-K flip-flopokkal történő realizációjának vezérlési táblája

A *2.33. ábra* mutatja a felsorolt táblázatokat, feladatunk megoldásának soron következő lépéseként. A baloldali összevont szimbolikus állapottábla alapján két állapothoz (**ac, b**) kell állapot-kódot választani. Ez egyetlen tárolóval lehetséges. Mindegy, melyik állapothoz rendeljük a **Q = 0,** és melyikhez a **Q = 1** értéket. Ezután illesztjük a kódolt állapottáblához a **J-K** tárolóval való megvalósításhoz szükséges vezérlési táblát. A kódolt állapottáblából kiolvassuk a **Q** előírt megváltozását, majd a tároló vezérlési táblája alapján a **J** és **K** oszlopokba beírjuk az ehhez szükséges értékeket.

**összevont szimb. áll.tábla** | **kódolt áll.tábla** | **vezérlési tábla**

| akt. áll. | köv.áll/kim. | | k.köv.áll./kim | | | $\bar{e}$ | | $e$ | |
|---|---|---|---|---|---|---|---|---|---|
| | $\bar{e}$ | $e$ | $Q$ | $\bar{e}$ | $e$ | $J$ | $K$ | $J$ | $K$ |
| *ac* | *ac/0* | *b/0* | *0* | *0/0* | *1/0* | *0* | *-* | *1* | *-* |
| *b* | *ac/0* | *ac/1* | *1* | *0/0* | *0/1* | *-* | *1* | *-* | *1* |

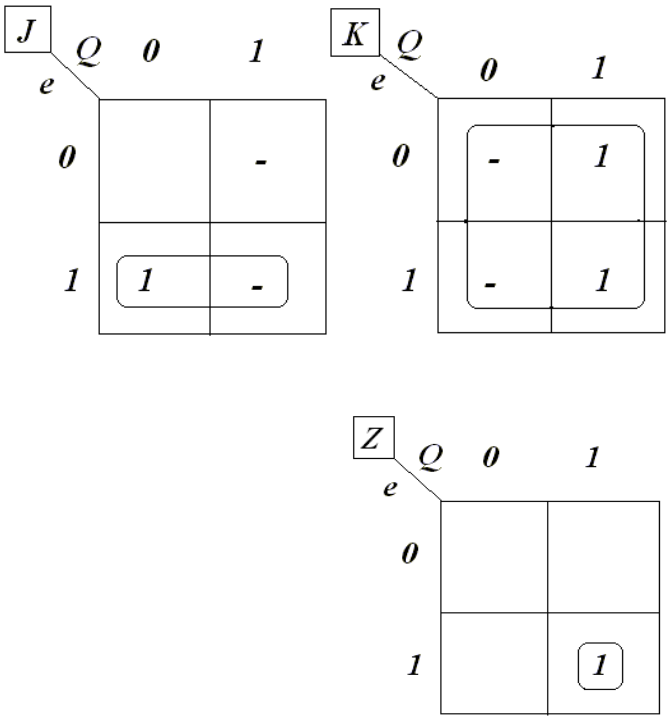
*2.33. ábra. A feladat megoldásának három fontos táblázata*

### 2.4.1.10. Az $f_y$ és $f_z$ hálózatok tervezése K-táblák segítségével

A vezérlési táblák megszerkesztése után hozzáfoghatunk a két kombinációs hálózat megtervezéséhez. Ehhez minden adat kiolvasható a táblázatokból. Nyilvánvaló, hogy az általánosabb Mealy-típusú realizációnál mind az $f_y$, mind az $f_z$ hálózat bemeneteit a teljes hálózat bemenetei és a tárolók kimenetei alkotják, tehát a szükséges **K**-táblákat ennek alapján kell felvennünk.

### 2.4.1.11. A feladat megoldására szolgáló hálózat K táblái és lefedésük (*2.34. ábra*)

A kódolt állapottáblából már látható, hogy esetünkben igen egyszerű, kétváltozós K táblákkal megadhatjuk a két kombinációs hálózat logikai függvényeit.




*2.34. ábra. A feladat K táblái*

A K-táblák kiértékelése igen egyszerű eredményekhez vezet :

$$J = e, \quad K = 1, \quad Z = Qe$$

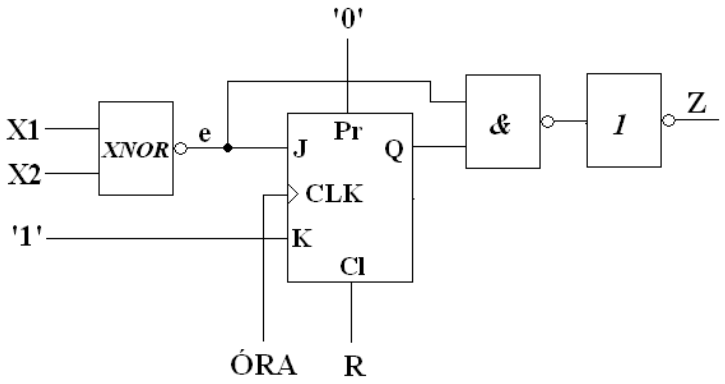
### 2.4.1.12. Realizáció

A realizáció során – amellett, hogy a flip-flop szimbólumokkal és az ismert kapu-szimbólumokkal felrajzoljuk a hálózat struktúráját, gondoskodnunk kell a kezdeti (bekapcsolás utáni) állapot beállításáról is. Ha **PRESET** és **CLEAR** bemenetekkel is rendelkező flip-flopokat választunk, akkor egyszerű dolgunk van: Ha a kezdeti állapotban egy adott tároló kimenete **0**, a **CLEAR** bemenetre adunk egy kezdeti állapotba állító impulzust, és a **PRESET** bemenetet állandó 0-ba állítjuk, ha pedig egy adott tárolót kezdeti állapotban **1**-be kell állítani, akkor a **PRESET** kimenetre adunk impulzust, és a **CLEAR** bemenetre konstans **0**-t.

Mindaddig, amíg a kezdeti beállítás más módszereit meg nem ismerjük, csak **PRESET** és **CLEAR** bemenetekkel is rendelkező flip-flopokat használunk.
A kezdeti állapot beállítását eredményező speciális bemenő-jelet **R (RESET)** szimbólummal jelöljük.

### 2.4.1.13. A feladat megoldásának realizációja

Meglepő lehet számunkra, hogy a *2.35. ábrán* bemutatott realizáció tárolójának kimenete nincs visszacsatolva a következő állapotot generáló hálózat bemenetére. Nincs visszacsatolás, annak ellenére, hogy a szinkron sorrendi hálózat általános modelljében ezt hangsúlyoztuk. Könyveljük el, hogy egy adott speciális funkció megvalósítása vezethet arra, hogy egyik vagy másik szekunder változó értékétől nem függenek egyik vagy másik flip-flop állapot-változásai. Realizált hálózatunk sajátossága, hogy a flip-flop aktuális kimenetétől nem függ annak a következő állapota.

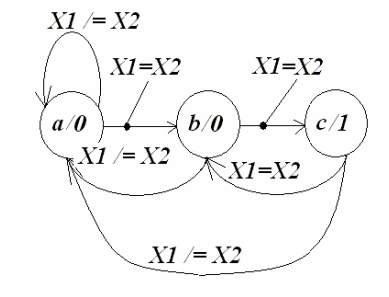



*2.35. ábra. A feladat realizációja*

### 2.4.1.14. A feladat megoldása Moore-típusú hálózattal

Ebben a pontban bemutatjuk a feladat Moore-típusú hálózattal történő realizációját, elsősorban a Mealy-típussal való realizációtól való eltérések hangsúlyozásával. Már az állapot-gráfon az állapotok jelölése mutatja a Moore típusnak azt a jellegzetességét, hogy a kimeneti kombinációk az csak az állapotok függvényei. Ugyancsak látható az eltérés a szimbolikus állapottáblán is. (*2.36. ábra*) Az előzetes szimbolikus állapottábla alapján elvégezzük az állapot-összevonási lehetőségek vizsgálatát. A szabály hasonló : kér állapot összevonható, ha a hozzájuk tartozó kimeneti kombinációk és a következő állapotok bemeneti kombinációnként azonosak. Könnyen megállapíthatjuk, hogy ilyen párok nincsenek, tehát a Moore-típusú hálózatot három állapottal, azaz két szekunder változóval kell megterveznünk.

A *2.37. ábra* együtt mutatja a szimbolikus és a kódolt állapottáblát, illetve a vezérlési táblát. Ennek kiemelendő sajátossága, hogy a két szekunder változó egyik kombinációja, nevezetesen az **1 1** itt kihasználatlan, tehát a belőle származó következő állapotokhoz, és azok vezérléseihez közömbös bejegyzéseket tehetünk. A **K**-táblákon történő függvény-lefedések és a kapu-szintű realizáció már rutin-munka. Az állapot-kombinációk és a kimeneti kombinációk közötti egyértelmű függés szerint a **Z** kimenet csak a **c** állapotban, azaz az **1 0** állapotkombináció fennállásakor lesz magas szinten. Ezt **K**-tábla nélkül is könnyen realizáljuk. (*2.38., 2.39. ábrák*)

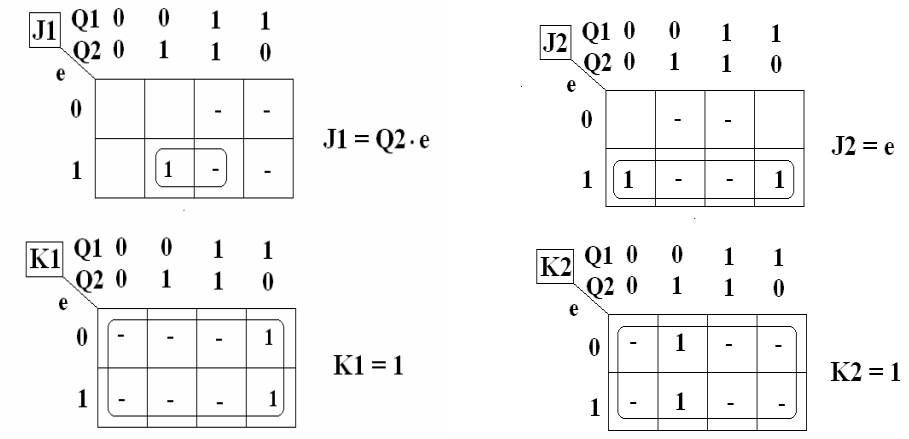



| akt. áll./ /kim. | köv.állapot | |
|---|---|---|
| | X1/=X2 | X1=X2 |
| a/0 | a | b |
| b/0 | a | c |
| c/1 | a | b |

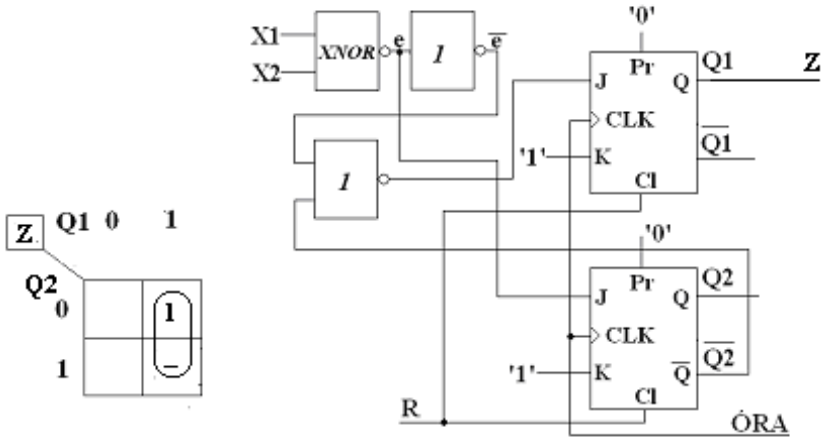
*2.36. ábra. A feladat Moore-típusú realizációjának állapot-gráfja és előzetes szimbolikus állapottáblája*

| szimb. áll.tábla | | | | kódolt áll.tábla | | vezérlési tábla | | | |
|---|---|---|---|---|---|---|---|---|---|
| akt. áll. /kim. | k. akt. áll. | köv.áll | | k.köv.áll. | | $\overline{e}$ | | $e$ | |
| | $Q1Q2$ | $\overline{e}$ | $e$ | $\overline{e}$ $Q1Q2$ | $e$ $Q1Q2$ | $J1\ K1$ | $J2\ K2$ | $J1\ K1$ | $J2\ K2$ |
| a/0 | 0 0 | a | b | 0 0 | 0 1 | 0 – | 0 – | 0 – | 1 – |
| b/0 | 0 1 | a | c | 0 0 | 1 0 | 0 – | – 1 | 1 – | – 1 |
| c/1 | 1 0 | a | b | 0 0 | 0 1 | – 1 | 0 – | – 1 | 1 – |
| | 1 1 | – | – | – – | – – | – – | – – | – – | – – |

*2.37. ábra. Állapottáblák és vezérlési tábla a feladat Moore.féle realizációjához*




*2.38. ábra.. A Moore-féle realizáció K-táblái és a lefedések algebrai alakjai Z kimenet nélkül*




*2.39. ábra. A feladat Moore-féle realizációja a Z kimenet lefedésével.*

## 2.5. ASZINKRON HÁLÓZAT TERVEZÉSI FOLYAMAT MINTA-PÉLDÁKON

### 2.5.1. Az első aszinkron hálózat tervezési mintafeladat

*Közvetlenül visszacsatolt kombinációs hálózattal tervezzünk olyan egy-bemenetű (**X**) és egy-kimenetű (**Z**) hálózatot, amelynek kimenetén a szint mindannyiszor ellenkezőjére vált, ahányszor **X** magas szintről alacsonyra vált. Bekapcsolás után a hálózat az **X=0** bemenetnél **Z = 0** kimenetet szolgáltasson.*

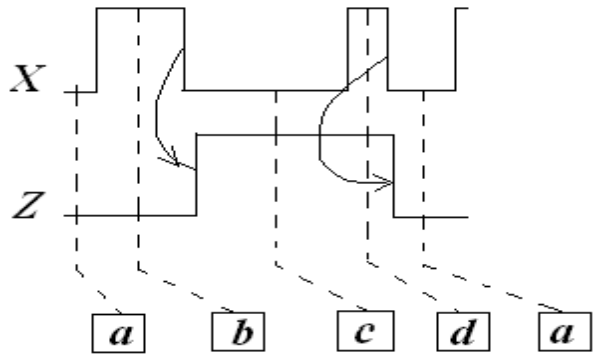
#### 2.5.1.1. Időzítési diagram és előzetes szimbolikus állapottábla

Ahogyan a szinkron hálózattervezés kezdeti lépéseként az állapot-gráf felvételét ajánlottuk, úgy ajánlható aszinkron hálózat tervezésének első lépéséül egy idődiagram felvétele. Az idődiagram felvételekor figyelembe kell venni, hogy az aszinkron hálózat minden új stabil állapotba való elindulása egy bemeneti jel változására indul meg, és működtetési szabály, hogy egyidejűleg csak egyetlen bemeneti jel változhat. Az idődiagram jól mutatja a bemeneti jelváltozások és a kimeneti kombináció-változások közötti ok-okozati összefüggéseket, és segítséget nyújt az előzetes, szimbolikus állapottábla felvételéhez. Az aszinkron hálózat szimbolikus előzetes állapottáblájának felvétele ugyanúgy intuitív módon oldandó meg, mint a szinkron hálózatok esetében, de általában nehezebb feladat annál. A specifikáció alapján itt is meg kell határoznunk egy kezdeti állapotot, amelybe a realizált hálózatnak a bekapcsolás után kerülnie kell, és szükséges, hogy ehhez egy bemeneti kombináció tartozzék.

Ugyancsak fontos tervezői döntés, hogy Mealy-, vagy Moore-típusú hálózatot akarunk-e tervezni, hiszen az előzetes állapottábla felépítése ettől jelentősen függ. Ezután az idődiagram alapján, a bemenetekre előírt jelváltozási szabály szem előtt tartásával előírjuk a következő szimbolikus állapotokat. Úgy képzeljük, hogy egy új állapot kezdetben tranziens állapotként jelentkezik az $f_y$ kimenetén, majd a bemenetre visszajutva stabilizálódik. Amikor arról döntünk, hogy egy adott változásra új állapotot vegyünk-e fel, vagy megteszi egy már felhasznált szimbolikus állapot, gondoljunk arra, hogy új állapot felvétele sohasem vezet logikai hibához, és a feltétlenül szükséges állapotokat később úgyis szisztematikus módszerrel határozzuk meg (állapot-összevonás). Ezzel szemben az, ha egy régi állapotot használunk fel kellő óvatosság és meggondolás nélkül, abból könnyen lehet funkcionális hiba. Ezért az a legfontosabb, hogy ismerjük fel azokat az eseteket, amikor nem szabad régi állapotot felhasználni. Ezek listáját majd később, némi példa megoldási tapasztalat birtokában fogjuk megadni.

### 2.5.1.2. A feladat időzítési diagramja és előzetes szimbolikus állapottáblája *(2.40. és 2.41. ábrák)*

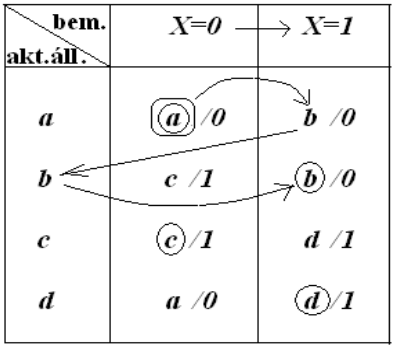
Az idődiagramban az **X =0** bemenethez tartozó kezdeti állapotot **a**-val jelöltük, és ehhez felvettük a **Z = 0** kezdeti kimeneti kombinációt. Az **X** első felfutása hatástalan, de fontos hogy az első felfutás tényét a hálózat regisztrálja egy új, **b** állapotba menetellel. Az ezután bekövetkező lefutás nemcsak újabb állapotváltást (**c**) , de a kimenet felfutását is kiváltja. A **c** állapot egyértelműen jelzi, hogy az **X** első lefutása bekövetkezett. **X** második felfutás ismét új állapot bevezetését igényli, **Z** változása nélkül. Az is érthető, hogy az újabb lefutás a kezdeti állapotot állítja be, mind a belső, mind a kimeneti állapot szempontjából.




*2.40. ábra.. Az első aszinkron tervezési feladat idődiagramja*

Ennek az idődiagramnak alapján szerkeszthető az előzetes szimbolikus állapottábla. Az állapottáblán körbe-foglalással jelöljük a stabil állapotokat. Tudjuk, hogy aszinkron állapottáblán stabil következő állapot az, amelynek szimbóluma azonos az aktuális állapot szimbólumával. A működést az állapottáblán is követhetjük. A kezdeti **a** aktuális állapotban az **X=0** bemenet az **a** állapotot stabilizálja. Az állapot mellett „/” jellel elválasztva látjuk a kezdeti kimeneti kombinációt.

| bem. / akt.áll. | X=0 | X=1 |
|---|---|---|
| a | (a)/0 | b /0 |
| b | c /1 | (b)/0 |
| c | (c)/1 | d /1 |
| d | a /0 | (d)/1 |




*2.41. ábra. Az első aszinkron feladat állapottáblája, és stabil átmenetek közötti átmenet szemléltetésével*

Szemléletes, ha a bekarikázott **a** állapotra „helyezzük a ceruzánkat", figyelemmel a sort és oszlopot kijelölő aktuális állapotra, illetve a bemeneti kombinációra. Ha az **X** felfut, akkor a ceruzánkat vízszintesen elmozdítjuk, és a **b**, tranziens (nem stabil) következő állapotkódot találjuk, változatlan **Z** értékkel. Ha ez visszajut a bemenetre, ezt azzal követhetjük, hogy ceruzánkat elmozdítjuk a **b** aktuális állapot sorára. Itt viszont nyilvánvaló, hogy a **b** állapot stabilizálódik . . . és így tovább.

### 2.5.1.3. Állapot-összevonás

Aszinkron hálózatok előzetes szimbolikus állapottáblája alapján végzett állapot-összevonás elvei azonosak a szinkron hálózatoknál megismertekkel. Két állapot összevonható, ha bemeneti kombinációnként azonosak a hozzájuk rendelt kimeneti kombinációk, és a következő állapotok is.

### 2.5.1.4. Állapot-összevonás a feladat állapottábláján

A *2.41. ábrán* látható állapottáblán nem találunk összevonható állapotokat.

### 2.5.1.5. Állapot-kódolás, a kódolt állapottábla felvétele

Következő lépés a kódolt állapottábla felvétele. Ez semmilyen elvi nehézséget nem támaszt. Célszerű a stabilitás tényét a kódolt állapotokon továbbra is jelölni.

### 2.5.1.6. A feladat állapotainak kódolása és kódolt állapottáblája

Négy belső állapotot két szekunder változóval kódolhatunk. Egy lehetséges és kézenfekvő kód-kiosztás lehet a következő :

**a → 0 0**
**b → 0 1**
**c → 1 0**
**d → 1 1**

Az ennek megfelelő kódolt állapottáblát a *2.42. ábrán* láthatjuk.

### 2.5.1.7. Analízis a kritikus versenyhelyzetek felderítésére

A szinkron hálózatok állapotkódolásánál szabad kezünk van abban, hogy a megfelelő hosszúságú szavakból álló kódkészlet szavait hogyan rendeljük hozzá a szimbolikus állapotokhoz, ugyanis minden választás a specifikációnak megfelelő megoldáshoz vezet. Aszinkron hálózatok állapotkódolásakor nem ilyen jó a helyzet. Amennyiben egy tranziens állapot kódja egynél több szekunder változó értékében különbözik a kiinduló stabil állapot kódjától, a reális hálózaton az eltérő jel-késleltetési utak miatt átmenetileg olyan más, tranziens állapotok is jelentkezhetnek az $f_y$ hálózat kimenetén, amelyek stabilizálódhatnak. Ezzel más, a specifikációnak ellentmondó pályára áll az aszinkron hálózat. Az ilyen hiba-lehetőségeket kritikus versenyhelyzeteknek nevezzük. Kiküszöbölésükre számos módszert dolgoztak ki, amelyek a kód megfelelő megválasztását eredményezik. Ezek közül most egy egyszerű, intuitív módszert mutatunk be a feladat kapcsán.

| bem. / akt. áll. | köv.áll/kim | | k. akt. áll. | k.köv.áll/kim | |
|---|---|---|---|---|---|
| | X=0 | X=1 | $Y1^vY2^v$ | X=0 Y1Y2/Z | X=1 Y1Y2/Z |
| a | (a)/0 | b /0 | 0 0 | (0 0)/0 | 0 1 /0 |
| b | c /1 | (b)/0 | 0 1 | 1 0 /1 | (0 1)/0 |
| c | (c)/1 | d /1 | 1 0 | (1 0)/1 | 1 1 /1 |
| d | a /0 | (d)/1 | 1 1 | 0 0 /0 | (1 1)/1 |

*2.42. ábra.. Az első aszinkron feladat kódolt állapottáblája*

### 2.5.1.8. Kritikus versenyhelyzetek a feladat kódolt állapottáblájának vizsgálatával

Tegyük fel, hogy az **X = 1**-hez tartozó **0 1** (**b**) állapotban vagyunk. Ha most **X** bemenetet **0**-ra kapcsoljuk, akkor tranziens állapotként az **1 0 (c)** állapot beállását várjuk, amely a bemenetre visszajutva stabilizálódik, és ezzel megtörténik az elvárt **0 1 → 1 0** átmenet. Ennek szemléltetését látjuk a *2.43. ábrán*. Sajnos ha a hálózatot ezzel az állapotkóddal megvalósítjuk, hibás lehet a valóságos működés. A *2.44. ábrán* szemléltetjük, mi lesz annak a következménye, ha a késleltetési idők különbözősége miatt egy másik tranziens állapot, a **0 0** áll be.

| bem. / akt. áll. | köv.áll/kim | | k. akt. áll. | k.köv.áll/kim | |
|---|---|---|---|---|---|
| | X=0 | X=1 | $Y1^vY2^v$ | X=0 Y1Y2/Z | X=1 Y1Y2/Z |
| a | (a)/0 | b /0 | 0 0 | (0 0)/0 | 0 1/0 |
| b | c /1 | (b)/0 | 0 1 | 1 0/1 | (0 1)/0 |
| c | (c)/1 | d /1 | 1 0 | (1 0)/1 | 1 1/1 |
| d | a /0 | (d)/1 | 1 1 | 0 0/0 | (1 1)/1 |

*2.43. ábra.. Egy ideális állapot-átmenet szemléltetése a kódolt állapottáblán*

| bem. / akt. áll. | köv.áll/kim | | k. akt. áll. | k.köv.áll/kim | |
|---|---|---|---|---|---|
| | X=0 | X=1 | $Y1^vY2^v$ | X=0 Y1Y2/Z | X=1 Y1Y2/Z |
| a | (a)/0 | b /0 | 0 0 | (0 0)/0 | 0 1/0 |
| b | c /1 | (b)/0 | 0 1 | 1 0/1 | (0 1)/0 |
| c | (c)/1 | d /1 | 1 0 | (1 0)/1 | 1 1/1 |
| d | a /0 | (d)/1 | 1 1 | 0 0/0 | (1 1)/1 |

*2.44. ábra.. Egy lehetséges valóságos állapot-átmenet szemléltetése : kritikus a versenyhelyzet **Y1** és **Y2** szekunder változók között a **0 1 → 1 0** átmenetnél*

### 2.5.1.9. Kritikus versenyhelyzetek kiküszöbölése állapot-átkódolással

A kritikus versenyhelyzetek sok esetben egyszerű kód-átrendezéssel, vagy a nem használt kódszavak bevonásával kiküszöbölhetők. A lényeg, hogy a kiindulási és a cél stabil állapotok kódjai között csak egy szekunder változó értékében legyen különbség.

### 2.5.1.10. A feladat állapot-kódjának megváltoztatása a kritikus versenyhelyzetek kiküszöbölésére

A következő kódválasztás a *2.45. ábra* tanúsága szerint megfelel e kritikus hazárdmentesség követelményének. Javasoljuk az olvasónak, analizálja valamennyi stabil állapotátmenet mentességét a kritikus versenyhelyzetektől.

**a → 0 0**
**b → 0 1**
**c → 1 1**
**d → 1 0**

| bem. / akt. áll. | köv.áll/kim | | k. akt. áll. | k.köv.áll/kim | |
|---|---|---|---|---|---|
| | X=0 | X=1 | $Y1^vY2^v$ | X=0 Y1Y2/Z | X=1 Y1Y2/Z |
| a | (a)/0 | b /0 | 0 0 | (0 0)/0 | 0 1 /0 |
| b | c /1 | (b)/0 | 0 1 | 1 1 /1 | (0 1)/0 |
| c | (c)/1 | d /1 | 1 1 | (1 1)/1 | 1 0 /1 |
| d | a /0 | (d)/1 | 1 0 | 0 0 /0 | (1 0)/1 |

*2.45. ábra. Az első aszinkron feladat új kódolt állapottáblája, kritikus versenyhelyzetektől mentes állapotkódokkal***.**

### 2.5.1.11. A realizáció K-táblái és lefedésük

A kritikus versenyhelyzetektől mentes kódolt állapottábla realizációjának két útja van. Az egyik egy közvetlenül visszacsatolt $f_y$, a másik egy **S-R** tárolókkal visszacsatolt $f_y$ hálózat tervezése. Az utóbbi egy vezérlési tábla kidolgozását is igényli.

Vigyáznunk kell arra, hogy valamennyi szekunder változóhoz tartozó függvényt *statikus hazárdoktól mentesen* kell lefedni, Az $f_y$ hálózat

kimenetein jelentkező statikus hazárd ugyancsak a specifikációtól eltérő, hibás működéshez vezethet. Általában követelmény a kimenet hazárdmentessége is.

### 2.5.1.12. Az első aszinkron feladat realizációja

Az első aszinkron feladat **K-**táblái és lefedésük rutin-feladat, de vigyáznunk kell arra, hogy mindhárom függvény realizációját *statikus hazárdoktól mentesen* kell megoldani. Az $f_y$ hálózat kimenetein jelentkező statikus hazárd ugyancsak a specifikációtól eltérő, hibás működéshez vezethet (*2.46. ábra*).
Értelmezzük a **Z**-re kapott érdekes eredményt. **Z** nem függ a bemenettől, csakis egyetlen szekunder-változótól. Mealy típusú hálózat tervezésébe fogtunk, mégis, annak speciális eseteként, Moore típusú hálózatot kaptunk. A realizációt a *2.47. a. ábra* mutatja. Az ábra szerint hálózat azonban még nem használható. Ha bekapcsoljuk, azaz ráadjuk a tápfeszültséget, hiába adjuk rá az alapállapotnak megfelelő **X = 1** kombinációt, nemcsak az **a**, a **c** állapot is beállhat. Márpedig bizonyosnak kell abban lennünk, hogy a bekapcsolás után, az **a** állapot áll be, sőt a hálózatot ebbe az alapállapotba bármikor be szeretnénk állítani.

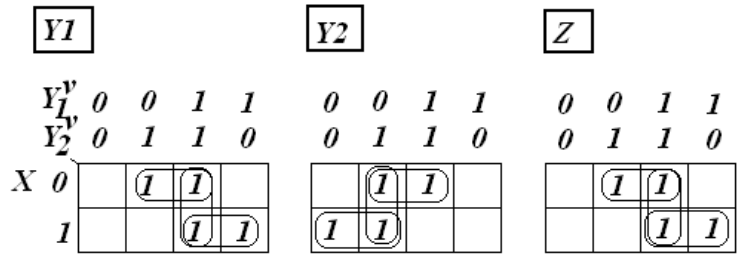
### 2.5.1.13. Aszinkron hálózatok beállítása kezdeti állapotba

Az aszinkron hálózatok kezdeti állapotba kényszerítésének módszereit egy későbbi pontban tárgyaljuk. Az alapelvet azonban már itt megadjuk, hogy az egyik legegyszerűbb módszer alkalmazásával első feladatunk megoldása teljes legyen. Az állapottáblából világosan megállapítható, hogy a tábla szerinti kezdeti állapot beállításának érdekében három feltételt kell teljesíteni. Először is, a kezdeti állapot kódját rá kell kényszerítenünk az $f_y$ hálózatra, a visszacsatolástól függetlenítve ezeket a bemeneteket. Ezt a helyzetet legalább addig kell fenntartani, amíg az $f_y$ kimenetein kialakul az kezdeti állapot kódja, illetve ha **S-R** tárolókkal csináljuk a visszacsatolást, azok kimenetén kialakul ez a kód. Másodszor, rá kell kapcsolnunk azt a bemeneti kombinációt, amely a kezdeti állapothoz tartozik. Harmadszor, megszüntetjük ezt az állapotot, és helyreállítjuk a visszacsatolást. Így a hálózat a kezdeti állapotban stabilizálódik.

### 2.5.1.14. Az első aszinkron minta-feladat realizációjának kiegészítése kezdeti állapotba kényszerítő, R-logikával

Az előző pont szerint eljárva, hasítsuk fel a visszacsatolást, és illesszünk be a visszacsatoló körbe *két két-bemenetű logikát*. Az egyik bemenetük közös, a kezdeti állapotba kényszerítő **R** (**RESET**) jel, a másik bemenetük a visszacsatolandó szekunder változókra kapcsolandó. A kimeneteket kapcsoljuk az $f_y$ hálózat bemeneteire. Mindkét **R**-logika az **R = 1** esetben **0**-t

ad tovább, ez pedig a kezdeti állapot kódja. Ha **R = 0**, a logikák kimenetére a megfelelő szekunder változó kerül, tehát él a visszacsatolás. A *2.47.b.* ábra mutatja az **R**-logikákkal kiegészített realizációt.

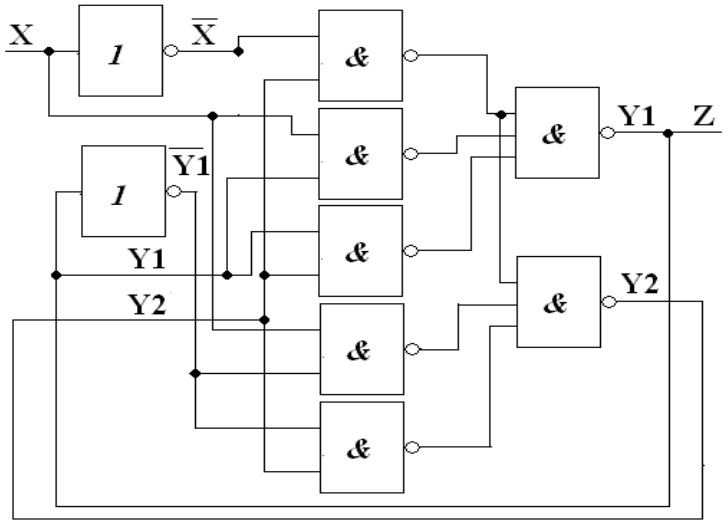



$$Y_1 = Y_2^v \overline{X} + Y_1^v X + Y_1^v Y_2^v$$

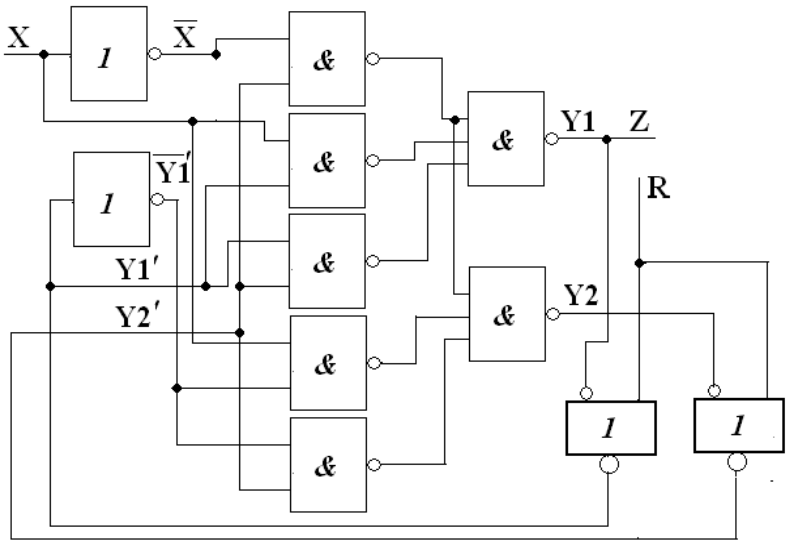
$$Y_2 = \overline{Y_1^v} X + \overline{Y_1^v} Y_2^v + Y_2^v \overline{X}$$

$$Z = Y1$$

*2.46. ábra. Az első aszinkron feladat K táblái*




*2.47.a.. ábra. Az első aszinkron feladat realizációja a kezdeti állapotba való beállítás nélkül.*




*2.47. b. ábra. A realizáció **R (RESET)** kezdeti állapotba állító logikákkal*

### 2.5.2. A második aszinkron hálózat tervezési mintafeladat

*Tervezzünk két-bemenetű (**X1, X2**) „sorrendi **ÉS** áramkört. A **Z** kimenet akkor és csakis akkor **1,** ha az **X1** bemenet előbb áll **1**-re, mint az **X2.** A tervezést végezzük el a következő állapotot előállító hálózat közvetlen visszacsatolásával, és **S-R** tárolókkal történő visszacsatolással is !*

**2.5.2.1. Előzetes, szimbolikus állapottábla** *(2.48. ábra)*
Ennél a feladatnál mellőzhetjük az idődiagramot, az előzetes szimbolikus állapottábla anélkül is megszerkeszthető. A feladat specifikációjából egyértelmű, hogy a **Z** magas értékei csak az **1 1** oszlopban lesznek, de csak azoknál az állapotoknál. Amelyek az **1 0**-ban levő állapotokat követik. Az kezdeti állapotot (**a**) a **0 0** bemeneti kombinációhoz vesszük fel. Ebből az állapotban az **1 1** bemeneti kombinációra való áttérés nem megengedett, hiszen ebben az esetben két bemeneti változó is értéket váltana, ezt pedig megtiltjuk. Így az **1 1**-heztartozó következő állapot és a kimenet értéke közömbös. A **0 1** és az **1 0** azonban megengedettek, mindkettőre új állapotot vettünk fel, és a hozzájuk tartozó kimenetek természetesen **0**-k. A **b** és **c** állapot sorainak kitöltésekor ismét célszerű először a tiltott bemeneti kombinációkhoz tartozó bejegyzésekről gondoskodni. Ha a **b** állapotban **0 0** jelentkezik, az a állapotba mehetünk vissza. Ha **1 1** jön, akkor egy új, a **d**

állapotot vesszük fel, és **Z**-t továbbra is alacsonyan tartjuk. A **c** állapotból **0 0**-ra **a**-ba mehetünk a **Z = 0**-val, **1 1**-re viszont az új e állapot beálltához a **Z = 1** tartozik, hiszen teljesült a speciális **ÉS** feltétel, **X2** az **X1**-t követően emelkedett magasra. Most a két legutóbb felvett állapotról, **d**-ről és **e**-ről kell gondoskodni. Egyikből sem kapcsolhatunk **0 0**-ra, de a **0 1**-re a **b / 0**, **1 0**-ra a **c / 0** jó választás, hiszen az előbbi esetben **X1** lefut, így a speciális-**ÉS** lehetőség távolabbra kerül, a második esetben viszont fennmarad.

| *X1 X2* **bem.** / **akt. áll** | **szimb.köv.áll. / kim.** | | | |
|---|---|---|---|---|
| | *0 0* | *0 1* | *1 0* | *1 1* |
| *a* | (*a*)*/0* | *b/0* | *c/0* | *-/-* |
| *b* | *a/0* | (*b*)*/0* | *-/-* | *d/0* |
| *c* | *a/0* | *-/-* | (*c*)*/0* | *e/1* |
| *d* | *-/-* | *b/0* | *c/0* | (*d*)*/0* |
| *e* | *-/-* | *b/0* | *c/0* | (*e*)*/1* |

*2.48. ábra. A második aszinkron minta-feladat előzetes szimbolikus állapottáblája.*

**2.5.2.2. Összevont, szimbolikus állapottábla** *(2.49. ábra)*

A közömbös bejegyzések miatt finomítjuk a két állapot összevonhatóságáról kimondott kritériumunkat. Két állapot összevonható, ha bemenő-kombinációnként megegyeznek a *specifikált* kimeneti kombinációk, és a *specifikált* következő állapotok. Ennek alapján a következő párok vonhatók össze: **ab , ad , bd , ce.** Az összevont állapotok tehát : (**abd**), (**ce**). Jelöljük az (**abd**) összevont állapotot **s1**-vel, a (**ce**)-t **s2**-vel. Az összevont állapottábla sorainak kitöltésénél az eredeti tábla közömbös bejegyzései okoznak gondot. Könnyen belátjuk azonban, hogy mindig azt az összevont állapotot kell beírnunk, amelyhez tartozó állapot az adott oszlopban, az összevont állapot eredeti állapotainak sorában szerepelt. Például: **s1** sorában az **1 1** oszlopban **s1**-t írunk, mivel az **s1**-hez tartozó állapotok specifikált következő állapota a **d**, ami az **s1**-ben szerepel.

| X1 X2 \ bem. / akt. áll | szimb.köv.áll. / kim. | | | |
|---|---|---|---|---|
| | 0 0 | 0 1 | 1 0 | 1 1 |
| s1 | (s1)/0 | (s1)/0 | s2/0 | (s1)/0 |
| s2 | s1/0 | s1/0 | (s2)/0 | (s2)/1 |

*2.49. ábra. A második aszinkron mintafeladat összevont állapottáblája*

**2.5.2.3. A második aszinkron minta-feladat kódolt állapottáblája**

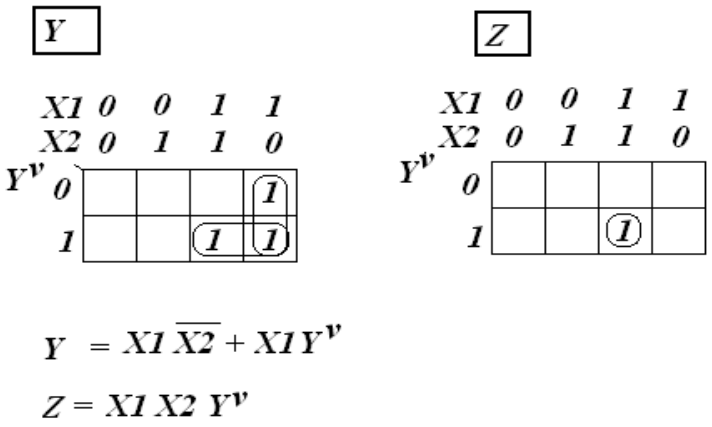
*(2.50.. ábra)*

Mivel két állapot van, egyetlen szekunder változó elég a kódoláshoz. Nyilvánvaló, hogy egy szekunder változó esetén a kritikus versenyhelyzet problémája fel sem merül.

| X1 X2 \ bem. / k.akt.áll. | kódolt.köv.áll. / kim. | | | |
|---|---|---|---|---|
| | 0 0 | 0 1 | 1 0 | 1 1 |
| 0 | (0)/0 | (0)/0 | 1/0 | (0)/0 |
| 1 | 0/0 | 0/0 | (1)/0 | (1)/1 |

*2.50. ábra. A második aszinkron minta-feladat kódolt állapottáblája*

**2.5.2.4. A második aszinkron mintafeladat $f_y$ és $f_z$ függvényeinek lefedése** *(2.51. ábra)*



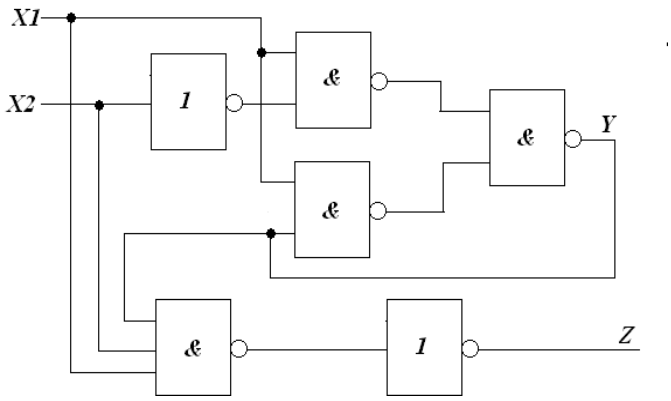

$$Y = X1\,\overline{X2} + X1\,Y^{v}$$

$$Z = X1\,X2\,Y^{v}$$

*2.51. ábra. K-táblák a második aszinkron feladathoz*

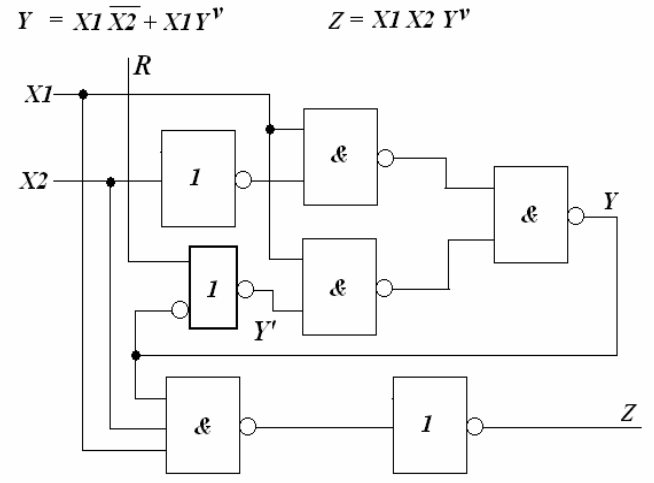
**2.5.2.5. A speciális, sorrendi ÉS kapu realizációja R-logika nélkül** *(2.52..ábra)*

$Y = X1\,\overline{X2} + X1\,Y^{v}$ $\qquad$ $Z = X1\,X2\,Y^{v}$




*2.52. . ábra. A sorrendi ÉS kapu NÉS-NÉS realizációja*

### 2.5.2.6. A speciális, sorrendi ÉS kapu realizációja R-logikával *(2.53. ábra)*




*2.53. ábra. A sorrendi ÉS áramkör R-logikával kiegészítve.*

### 2.5.2.7. A második aszinkron feladat ( a sorrendi ÉS hálózat) realizációja S-R tárolókkal

A *2.54. ábra* mutatja az S-R tároló vezérlési tábláját. Ennek segítségével kapjuk meg a sorrendi ÉS áramkör S-R tárolós realizációjának vezérlési tábláját, ami a *2.55. ábrán* szerepel. A *2.56. ábra* a K-táblákat, a *2.57. ábra* a realizációt mutatja. Ez kezdeti állapot beállítás nélküli realizáció.

| S | R | $Y^v$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | – |
| 1 | 1 | 1 | – |

| $Y^v$ → Y | | S | R |
|---|---|---|---|
| 0 | 0 | 0 | – |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | – | 0 |

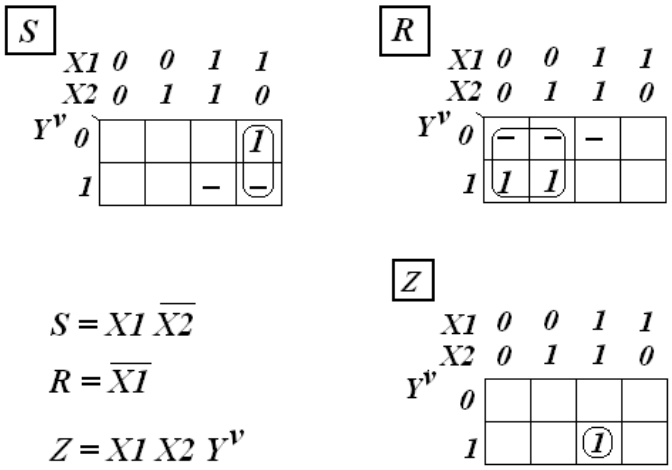
*2.54. ábra. Az S-R tároló vezérlési táblája*

| X1 X2 / bem. k.akt.áll. | kódolt.köv.áll. / kim. | | | |
|---|---|---|---|---|
| | 0 0 | 0 1 | 1 0 | 1 1 |
| 0 | (0)/0 | (0)/0 | 1/0 | (0)/0 |
| 1 | 0/0 | 0/0 | (1)/0 | (1)/1 |

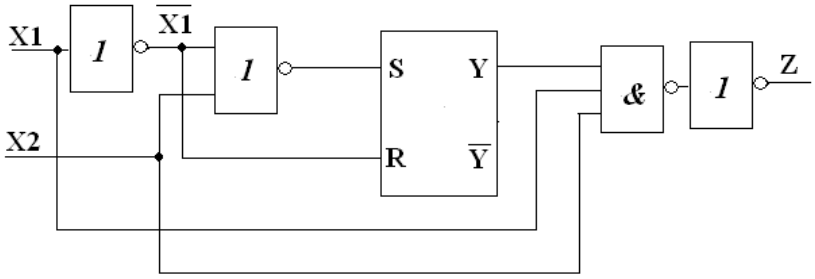
**vezérlési tábla:**

| X1 X2 / bem. k.akt.áll. | 0 0 | | 0 1 | | 1 0 | | 1 1 | |
|---|---|---|---|---|---|---|---|---|
| | S | R | S | R | S | R | S | R |
| 0 | 0 | – | 0 | – | 1 | 0 | 0 | – |
| 1 | 0 | 1 | 0 | 1 | – | 0 | – | 0 |

*2.55. ábra. A sorrendi ÉS kódolt állapottáblája és vezérlési táblája*



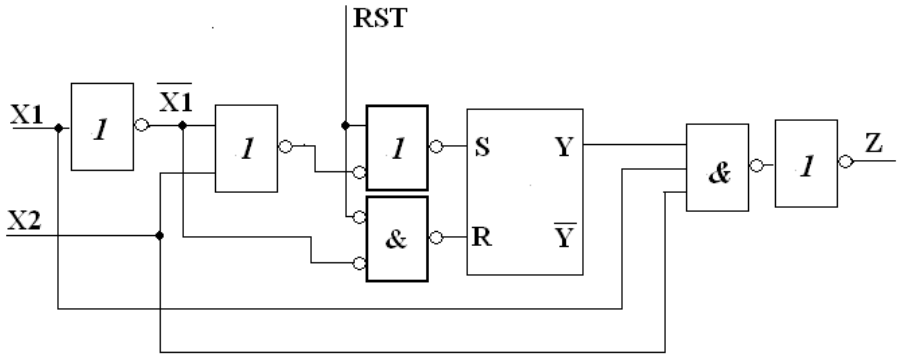

*2.56. ábra. A sorrendi és S-R tárolós megvalósításának K táblái*




$$S = X1\,\overline{X2} \qquad R = \overline{X1} \qquad Z = X1\,X2\,Y^{v}$$

*2.57. ábra. A sorrendi ÉS S-R tárolós megvalósítása kezdeti állapot-beállítás nélkül*

A kezdeti állapot beállítását könnyű megoldani az **S** és az **R** bemeneteken. Ha az **RST (RESET)** jelet felemeljük, akkor az **S** bemenetre **0**-t, az **R** bemenetre **1**-et kényszerítünk, így állítjuk be az **Y = 0** kezdeti állapotot (*2.58. ábra*).




*2.58. ábra. A sorrendi „ÉS” S-R tárolós megvalósítása kezdeti állapot-beállítással*

### 2.5.2.8. Lényeges hazárdok aszinkron hálózatokban

Eddigi aszinkron hálózat-tervezési példáink megoldása során csak a szekunder változók versengése miatt kialakuló hibákkal és azok kiküszöbölésével foglalkoztunk. Ez csak akkor tekinthető korrekt eljárásnak, ha garantálni tudjuk azt, hogy a bemeneti jelek változása okozta események a szekunder változók értékeinek megváltozásának kezdete előtt már lezajlanak.

Ez a feltételezésünk abban is megnyilvánul, hogy amikor az állapottáblán követjük az aszinkron hálózat működését, egyik oszlopról a másikra térünk át, és csak ezután vizsgáljuk a tranzienseket. A valóságban ez a feltételezés nem mindig jogos. A szekunder változók és egyik bemeneti változó kritikus versenyhelyzete úgynevezett lényeges hazárd veszélyével jár. Ennek kiküszöbölése időkésleltetési manipulációkat igényel.

## 2.6. SORRENDI HÁLÓZATOK TERVEZÉSÉNEK FOLYAMATA

### 2.6.1. Szinkron szekvenciális hálózatok tervezésének fő lépései

1. A szimbolikus előzetes állapottábla felvétele
2. Állapot-összevonás
3. Állapotkódolás
4. Összevont kódolt állapottábla felvétele
5. Döntés az állapotregiszter flip-flopjainak fajtájáról
6. Vezérlési tábla felvétele
7. Vezérlő jelek logikai függvényeinek lefedése
8. Kimeneti hálózat logikai függvényének lefedése
9. A kezdeti állapot beállításáról való gondoskodás

### 2.6.2. Aszinkron szekvenciális hálózatok tervezésének fő lépései

1. A szimbolikus előzetes állapottábla felvétele
2. Állapot-összevonás
3. Állapotkódolás, a kritikus versenyhelyzetekre figyelemmel.
4. Kódolt állapottábla felvétele
5. Közvetlenül visszacsatolt kombinációs hálózat, vagy S-R tárolós visszacsatolás. (Döntés)

6. Szekunder változók lefedése, vagy a vezérlési tábla felvétele és a vezérlő jelek lefedése.

7. A hálózat elemzése a lényeges hazárdok kiküszöbölésére. Késleltetések beiktatása

8. Kimeneti hálózat logikai függvényének lefedése

9. A kezdeti állapot beállítása

## 2.7. SORRENDI HÁLÓZATOK KEZDETI ÁLLAPOTÁNAK BEÁLLÍTÁSA

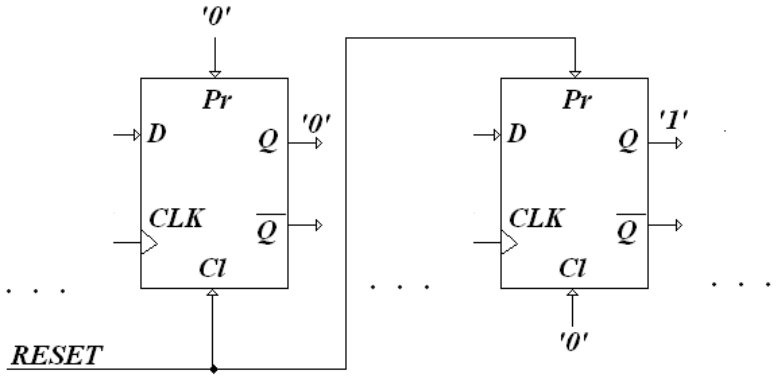
A kezdeti állapot-kódok kezdeti beállítását tulajdonképpen nemcsak utólag, hanem a tervezéssel párhuzamosan is elvégezhetnénk, ha a bemenő jelek listájára felvennénk az **R** jelet, és már a szimbolikus összevont állapottáblán is figyelembe vennénk a lehetséges bemeneti kombinációk között. Ennek hátránya, hogy egyetlen járulékos bemeneti jel is jelentősen bonyolítja a tervezési folyamat valamennyi fázisát. Ezért célszerű ezt a lépést a tervezési folyamat végére hagyni, és a lehetséges megoldásokat részleteiben megvizsgálni.

### 2.7.1. Szinkron sorrendi hálózatok kezdeti állapotának beállítása

#### 2.7.1.1. Beállítás a PRESET (Pr) és a CLEAR (Cl) bemenetek kihasználásával

Szinkron hálózattervezési minta-feladataink megoldásában a kezdeti állapot beállítását lehetővé tevő kiegészítések megtervezésekor kihasználtuk a flip-flopok **PRESET** és **CLEAR** bemeneteit. A kezdeti állapot kódja példáinkban mindig csupa **0**-ból állt, így a flip-flopot **PRESET** bemenetét **0**-ra kapcsoltuk, és valamennyi **CLEAR** bemenetére rákapcsoltuk a kezdeti állapotot kikényszerítő **R (RESET)** bemeneti jelet.

Ebben a pontban ezt a módszert általánosítjuk tetszőleges kezdeti állapot kódra. A *2.59. ábra* egy elképzelt flip-flop sorban (állapot-regiszterben) két különböző kezdeti értékű flip-flopot mutat. Nyilvánvaló, hogy a **0** kezdeti értékűek **CLEAR**, míg az **1** kezdeti értékűek **PRESET** bemenetét aktivizáljuk az **R** jellel.




*2.59. ábra. A PRESET és CLEAR bemenetek felhasználása a kezdeti állapot kódjának beállítására.*

**2.7.1.2 Beállítás az $f_y$ hálózat kiegészítésével**

**2.7.1.2.1 Kiegészítő hálózat D tárolók esetén**

A **D** flip-flopok esetén alkalmazandó kiegészítő hálózatok igazságtáblái láthatók. A $D'_i$ olyan flip-flop bemenete, amelyet ***0***-ba, a $D'_j$ olyan flip-flop bemenet, amelyet ***1***-be kell állítani kezdetben. Az igazságtáblák láthatók a *2.60. ábrán*. A *2.61. ábra* mutatja a segédhálózatok beillesztését.

| $D_i$ | RESET | $D_i'$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

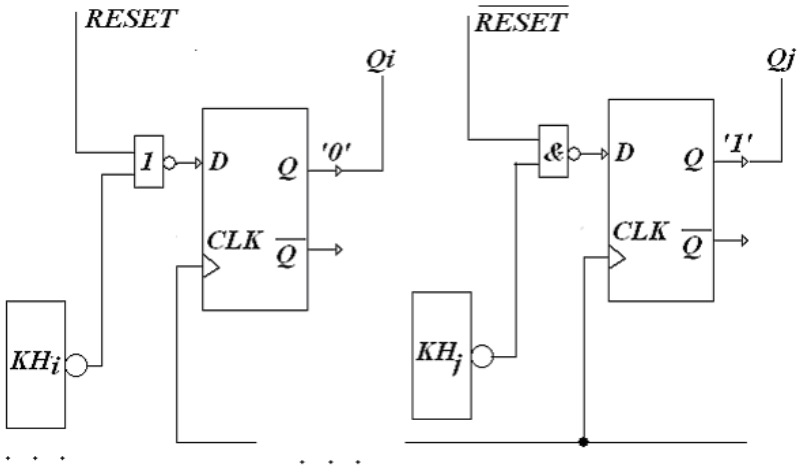
$D_i' = D\,\overline{RESET}$

$= \overline{\overline{D} + RESET}$

| $D_j$ | RESET | $D_j'$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

$D_j' = \overline{\overline{D}\;\overline{RESET}}$

*2.60. ábra. A D- tároló kezdeti állapot beállításához szükséges igazságtáblák és az egyszerű realizált logikák algebrai kifejezései*




*2.61. ábra. A kiegészítő hálózatok beillesztése D flip-flopok kezdeti állapotainak beállítására*

**2.7.1.2.2. Kiegészítő hálózat J-K tárolók esetén** *(2.6., 2.63. ábrák)*

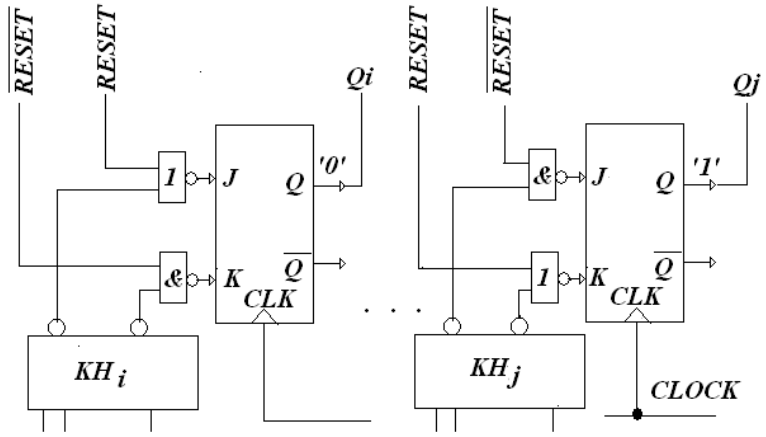
| $J_i$ | RESET | $J'_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| $K_i$ | RESET | $K'_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $J_j$ | RESET | $J'_j$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $K_j$ | RESET | $K'_j$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*2.62. ábra. J-K flip-flopok kezdeti állapot beállító kiegészítő hálózatainak igazságtáblái*




*2.63. ábra. J-K flip-flopok kezdeti állapotainak beállítása a kiegészítő hálózatokkal*

### 2.7.2. Aszinkron hálózatok kezdeti állapotának beállítása

#### 2.7.2.1. Közvetlenül visszacsatolt kombinációs hálózattal megvalósított aszinkron hálózat kezdeti állapotának beállítása

Szekunder változónként felhasítjuk a visszacsatolást, és beillesztünk egy-egy olyan kiegészítő hálózatot, amely **RESET = '0'** esetén a kiszámolt szekunder változó értéket, **RESET = '1'** esetén a kívánt kezdő értéket (**0-t** vagy **1**-t) helyezi a megfelelő kimenetre. Jelöljük ezt a kimenetet $\mathbf{Y}^*$-val. A kezdeti állapotot beállító segéd hálózatok igazságtáblái és logikai megoldásai láthatók *2.64. ábrán*.

Gondolnunk kell arra, hogy a kezdeti állapotot beállító **RESET** jelnek önmagában garantálnia kell, hogy a kezdeti állapot megjelenik a kombinációs hálózat, illetve az **S-R** tárolók kimenetén. Azt azonban, hogy ez a **RESET** jel eltűnése után stabilan meg is maradjon, azt a bemeneti kombinációt kell alkalmazni, amelyre a stabil kezdőállapotot előírtuk.

| $Y_i$ | RESET | $Y_i^*$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| $Y_j$ | RESET | $Y_j^*$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

$$Y_i^* = Y_i\,\overline{RESET} = \overline{\overline{Y_i} + RESET}$$

$$Y_j^* = Y_j + RESET = \overline{\overline{Y_j}\;\overline{RESET}}$$

*2.64. ábra. Közvetlenül visszacsatolt aszinkron hálózat kezdeti állapotának beállítása*

### 2.7.2.2. S-R tárolókkal visszacsatolt aszinkron hálózatok kezdeti állapotának beállítása

A *2.65. ábrán* az igazságtáblák, a *2.66. ábrán* a realizáció látható. Az ***i*** indexű **S** és **R** bemenetű tárolót **0**-ba, a ***j*** indexű **S** és **R** bemenetű tárolót **1**-be kell állítani.

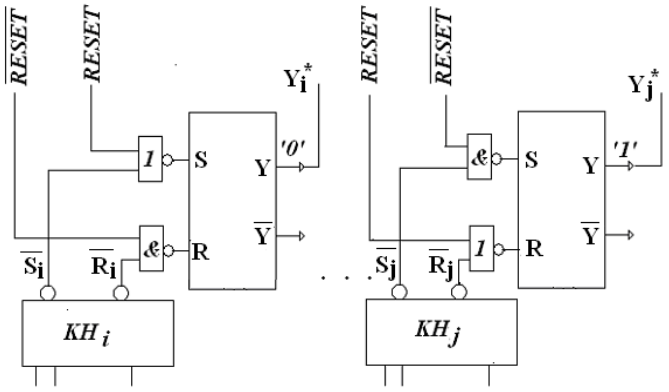
| $S_i$ | RESET | $S_i^*$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| $R_i$ | RESET | $R_i^*$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $S_j$ | RESET | $S_j^*$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $R_j$ | RESET | $R_j^*$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*2.65. ábra. Az S-R tárolóval visszacsatolt aszinkron hálózat kezdeti állapotának beállítása : igazság-táblák*



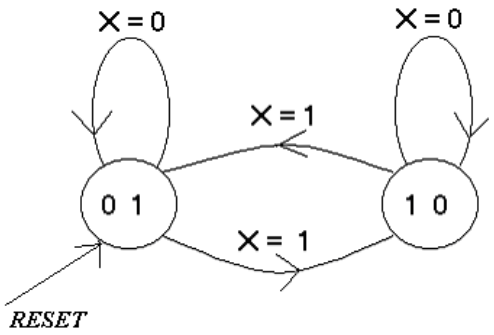

*2.66. ábra. Az S-R tárolóval visszacsatolt aszinkron hálózat kezdeti állapotának beállítása : segéd-hálózatok beillesztése*

*TERVEZÉSI FELADATOK 3.*

**1.** Két-bemenetű (***X1, X2***) és egy-kimenetű (***Z***) szinkron sorrendi hálózatunknak a kimenetén ***1***-el jeleznie kell, ha egymás után kétszer egymással ellentétes bit-értékű ( ***01*** vagy ***10*** ) kombináció érkezik érkezik a bemenetekre. Csak ***Preset*** és ***Clear*** bemenetekkel is rendelkező ***J-K*** tárolóink és ***NAND*** kapuink, valamint invertereink vannak. Tervezzük meg a hálózatot a kezdeti állapot beállításával együtt.

**2**. Két-bemenetű (***X1, X2***) és egy-kimenetű (***Z***) szinkron sorrendi hálózatunknak a kimenetén ***1***-el jeleznie kell, ha egymás után kétszer egymással egyező bit-értékű ( ***00*** vagy ***11*** ) kombináció érkezik a bemenetekre. Csak ***Preset*** és ***Clear*** bemenetekkel is rendelkező ***J-K*** tárolóink és ***NAND*** kapuink, valamint invertereink vannak. Tervezzük meg a hálózatot a kezdeti állapot beállításával együtt.

**3.** Tervezzen szinkron hálózatot ***Preset*** *és* ***Clear*** bemenetekkel rendelkező ***D*** flip-flopokkal a megadott állapot-gráf szerint. Az állapotokat a gráfban kódoltan adtuk meg.




**4.** ***Preset és Clear*** bemenetekkel ***nem rendelkező J-K MS*** flip-flopokkal tervezzük meg azt az egy-bemenetű ( ***X*** ) szinkron sorrendi hálózatot, amelynek két kimenete van, ***Z1*** és ***Z2***. A hálózat a ***Z1*** kimenetén ***1***-vel jelzi, ha kétszer egymás után ugyanolyan szintű bemenet érkezik, a másik kimeneten ( ***Z2*** ) pedig az a szint jelenik meg, amely kétszer egymás után jelentkezik a bemeneten. A kezdőállapot beállítására egy ***R*** jel áll rendelkezésre.

**5.** Kombinációs hálózat visszacsatolásával tervezzünk olyan két-bemenetű sorrendi áramkört, amely az ***X1 = 0 X2 = 0*** alaphelyzetben a kimeneten ***Z = 1*** értéket ad, és a működtetés során azt csak akkor változtatja ***0***-ra, ha ***X2*** fennmaradása mellett ***X1 0***-ra változik. A kezdeti állapot beállítását nem kell megoldania.

**6.** ***S-R*** tárolókkal történő visszacsatolással tervezzünk olyan két bemenetű ***(X1, X2)*** sorrendi áramkört, amely különböző szintű bemenetek esetén csak akkor ad a kimenetén ***( Z ) 1***-et, ha először az ***X1*** bemenet áll be arra a szintre, ahol az ***X2***-től való különbözőség fennáll. A kezdeti állapotban ***X1 =X2 = Z = 0*** legyen. A kezdeti állapotot egy ***R (Reset)*** jel segítségével kell beállítani.

**7.** Kombinációs hálózat visszacsatolásával tervezzünk olyan két bemenetű áramkört, az ***X1 = 1 , X2 = 0*** bemenetre ad a kimenetén ***1***-et, de csak akkor, ha először az ***X1*** bemenet áll be, majd azután az ***X2*** bemenet. Az kezdő állapot beállításáról nem kell gondoskodnia!

**8.** Kombinációs hálózat visszacsatolásával tervezzük meg azt az *egy bemenetű* és *egy kimenetű* aszinkron hálózatot, amely az ***X*** bemenet *minden második* felfutó-élére ellenkezőjére változtatja a ***Z*** kimenet szintjét, amelynek alapállapota az ***X = 0*** esetben ***Z = 0***. A kezdőállapotot egy ***R*** jellel kell beállítani.

**9.** Kombinációs hálózat visszacsatolásával tervezzünk olyan két-bemenetű sorrendi áramkört, amely az ***X1 = 1 X2 = 1*** alaphelyzetben a kimeneten ***Z = 0*** értéket ad, és a működtetés során azt csak akkor változtatja ***1***-re, ha ***X1*** fennmaradása mellett ***X2 0***-ra változik. A kezdőállapotot egy ***R*** jellel kell beállítani.

## 2.8. ÁLLAPOT-ÖSSZEVONÁSI MÓDSZEREK

### 2.8.1. Állapot-összevonás teljesen specifikált szimbolikus előzetes állapottáblán

#### 2.8.1.1 Az összevonhatóság feltétele

Általános megfogalmazást adunk arra, mikor tekinthetünk két szimbolikus állapotot összevonhatónak *egy teljesen specifikált előzetes, szimbolikus állapottáblán.* Teljesen specifikált az állapot-tábla (*TSH*) akkor, ha nem tartalmaz egyetlen „közömbös” bejegyzést sem. *Két állapot a TSH állapottábláján nem megkülönböztethető (NMK), ha a két állapotból elindulva bármely bemeneti sorozatra ugyanazt a kimeneti sorozatot látjuk.*
Ebből a magától értetődő definícióból kiindulva bizonyítható, hogy két állapot összevonható, *ha a két állapotból bármely bemeneti kombinációra adott kimeneti kombinációk megegyeznek, és NMK állapotokra vezetnek.*

#### 2.8.1.2. A nem-megkülönböztethetőség, mint bináris reláció

Ha a TSH NMK állapot-párjait megvizsgáljuk, azt tapasztaljuk, hogy az NMK pár-alkotás, mint RELÁCIÓ

- **reflexív**
- **szimmetrikus**
- **tranzitív**

A reflexívitás jelentése az a trivialitás, hogy egy szimbolikus állapot sajátmagától nem különböztatható meg, azaz

**a≡a**

Szimmetrikusak azok a bináris relációk, amelyere igaz, hogy amennyiben

**a≡b** akkor bizonyosan fennáll a
**b≡a** reláció is.

Tranzitív relációk esetén igaz, hogy amennyiben:

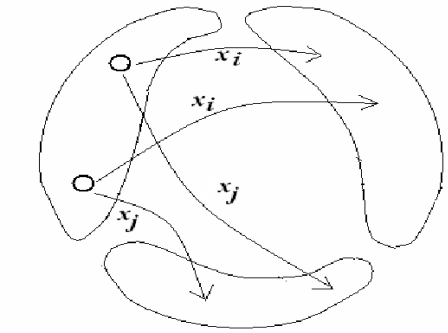
**a≡b és b≡c,** akkor teljesül az

**a≡c** is.

Magától értetődő, hogy a nem-megkülönböztethetőség reflexív, szimmetrikus és tranzitív. Az ilyen relációkat ekvivalencia-típusú relációknak nevezzük. Szokás a *TSH*-n ezt a tulajdonságot állapot-ekvivalenciának is nevezni, azaz az *NMK* állapotpárok tagjait *ekvivalenseknek* mondjuk.
A *nem-NMK* (*MK*), azaz a megkülönböztethető állapot-párok tagjait *antivalens* állapotoknak nevezzük. Az állapot-összevonás szempontjából fontos tétel, hogy egy adott halmaz elemein értelmezett ekvivalencia-típusú reláció a halmazt diszjunkt részhalmazokra bontja fel. Így az állapot-ekvivalencia az *TSH* állapothalmazát olyan, közös elemeket nem tartalmazó részhalmazokra bontja fel, amelyek az összevont állapothalmazt alkotják. Ezt szemléletesen mutatja a *2.67. ábra*, amelyen a diszjunkt részhalmazok egyikében két tetszőleges állapotból láthatjuk az $x_i$ és $x_j$ bemeneti kombinációkra történő elágazásokat. A lényeg, hogy bemenő-kombinációnként azonos részhalmazba ágaznak el az ekvivalens állapotok.
Az ekvivalencia kimutatására állapottábla alapján páronként kell vizsgálnunk az ekvivalencia vagy antivalencia tényét. Az alkalmazott jelölések a következők:
- **a ≡ b** : a és b ekvivalensek
- **a < > b** : a és b állapotok antivalensek

Az állapottábla analízise során sokszor nem lehet eldönteni azonnal az ekvivalencia vagy az antivalencia fennállását. Ezért, ha nem látjuk két állapotról azonnal, hogy antivalensek, akkor feljegyezzük azokat a feltételeket, amelyek fennállása esetén a két állapot ekvivalens.
A feltételes ekvivalenciát magával a feltétellel jelöljük. Például, ha a jelölés a következő felsorolás : (**ab, cd** . . .) akkor az a két állapot, amelyekre ez

vonatkozik, feltételesen ekvivalensek, azaz csak akkor ekvivalensek, ha **a ≡ b** *és* **c ≡ d**.

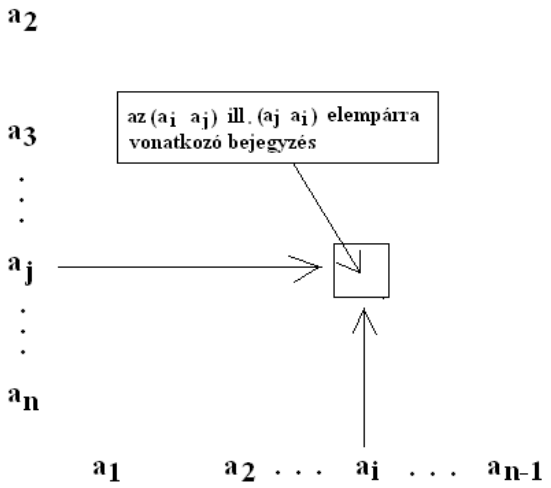



*2.67. ábra. Elágazások ekvivalens állapotokból*

Meg kell jegyeznünk, hogy az első sorrendi-hálózat tervezési feladataink megoldása során ennél szigorúbb feltételt alkalmaztunk, nevezetesen bemeneti kombinációnként megköveteltük mind a kimeneti kombinációk, mind a következő állapotok azonosságát. Az összevonhatóságot most sokkal mélyebben vizsgáltuk, így megfogalmazhattuk az összevonhatóság *szükséges és elégséges* feltételeit.

### 2.8.1.3. Lépcsős tábla az állapotok páronkénti vizsgálatára

Ha egy halmaz ***n*** elemből áll, akkor egy ***n*** x ***n*** méretű négyzetrács négyzeteibe bejegyezhetjük egy bináris reláció teljesülését, nem teljesülését, vagy a teljesülés feltételeit. Ha a reláció szimmetrikus, elég ehhez az átló menté felezett négyzetrács tábla, amit jellegzetes alakjáról lépcsős táblának nevezünk. A lépcsős tábla formája egy ***a1,a2, . . . an*** elemhalmaz esetén a *2.68. ábrán* látható: A lépcsős táblás állapot-összevonás természetesen nem áll meg a páronkénti ekvivalencia megállapításánál. Ha a tranzitivitást a megállapított állapot-párok között érvényesítjük, akkor kialakulnak a *maximális ekvivalencia osztályok*,




*2.68. ábra. A lépcsős tábla struktúrája*

azaz azok a diszjunkt állapot-halmazok, amelyeknél nagyobbakat már nem lehet találni. Így valamennyi állapot bekerül egy elvivalencia-osztályba, és nincs egyetlen olyan állapot sem, amely egynél több osztályba bekerülne.

### 2.8.1.4. Mintapélda megoldása lépcsős táblán

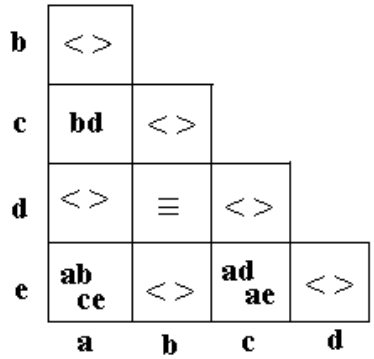
Tekintsük a *2.69. ábra* szerinti előzetes szimbolikus állapottáblát. Ennek alapján készül el az *első lépcsős tábla*, amely az ábra jobb oldalán látható. Kitöltésekor a következő módszert alkalmazzuk:

- Ha a kimenetek is és a következő állapotok is bemeneti kombinációnként azonosak, akkor a keresztezési cellába beírjuk az ekvivalencia szimbólumát ( ≡ ).
- Ha a kimeneti értékek legalább egy bemeneti kombinációra különbözők, akkor a keresztezési cellába beírjuk az antivalencia szimbólumát ( <> ).
- Ha a kimeneti értékek bemeneti kombinációnként megegyeznek, de a következő állapotok legalább egy bemeneti kombinációnál eltérnek, akkor feltétel bejegyzés kerül a keresztezési cellába. Egynél több eltérés esetén a rész-feltételek *és*-kapcsolatba kerülnek.

Példaképpen nézzünk meg néhány bejegyzést. Az elsőként vizsgált (**a b**) párról azonnal megállapítható az antivalencia, hiszen mind az **X = 0**-ra, mind az **X = 1**–re más és más kimeneti szintet ad a tábla.

Ugyanakkor az (**a c**) pár vizsgálatakor a kimenetekkel nincs baj, de a következő állapotok (**a c**) és (**d b**) ugyan nem azonosak, de ekvivalensek még lehetnek! Sőt az (**a c**) ekvivalenciájához feltételként az (**a c**) feltételt beírni tautológia, tehát csak a (**b d**) párost írjuk be. A (**b d**) ekvivalenciája a tautológia, illetve a reflexivitás miatt azonnal megállapítható.

| | X = 0 | X = 1 |
|---|---|---|
| a | **c** / 0 | **b** / 1 |
| b | **d** /1 | **e** / 0 |
| c | **a** / 0 | **d** / 1 |
| d | **b** / 1 | **e** / 0 |
| e | **e** / 0 | **a** / 1 |

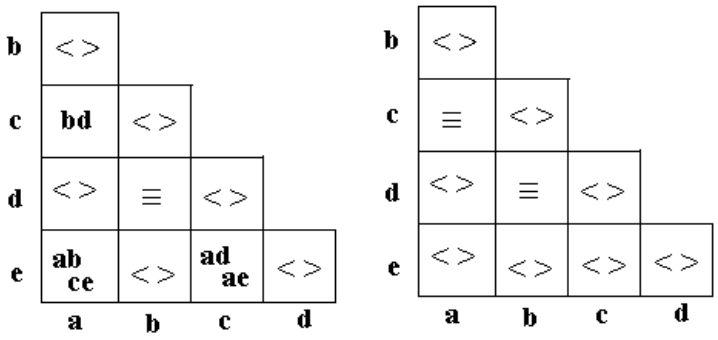



*2.69. ábra. Előzetes, szimbolikus állapottábla és a hozzá tartozó első lépcsős tábla*

A *második lépcsős táblát* az elsőből kiindulva alkotjuk meg a következő szabályok alapján:
A *2.70. ábra* baloldala a kiindulási *1. lépcsős tábla*, a jobboldali az ebből előállítható *2*. lépcsős tábla.

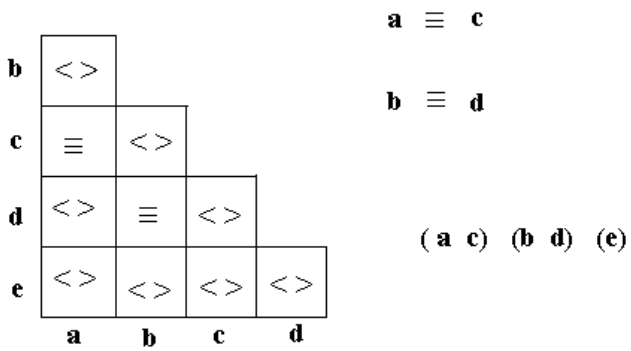
- Minden egyes antivalencia-bejegyzés következményeit érvényesítjük a táblán. Azaz, ha két állapot antivalens, és kettejük ekvivalenciája két másik állapot ekvivalencia-feltételeként szerepel valahol, oda antivalens bejegyzést kell tennünk.
- Ezután ezeket a második-generációs antivalencia bejegyzéseket is érvényesíteni kell, és mindezt addig kell ismételni, amíg érvényesítetlen antivalencia-bejegyzés van a táblán.

Példák a *2. lépcsős tábla* megszerkesztéséből: Az (**a b**) antivalenciájának következménye az (**a e**) pár antivalenciája, az (**a d**) antivalenciáé pedig az (**e c**) antivalencia. A (**b d**) ekvivalencia lehetővé teszi az (**a c**) ekvivalenciát.




*2.70. ábra. Az első és a második lépcsős tábla*

Példánkban a *2.tábla* már kizárólag ekvivalencia és antivalencia bejegyzéseket tartalmaz, így az ekvivalens párok és a tranzitivitás figyelembevételével a maximális ekvivalencia-osztályok könnyen kialakíthatók (*2.71. ábra*)




*2.71. ábra. A csak ekvivalenciákat és antivalenciákat tartalmazó lépcsős tábla*

Kiolvashatjuk az összes ekvivalens párt. A következő maximális ekvivalencia-osztályokat kaptuk :

**(a c)**, **(b d )**, **(e)**

### 2.8.1.5. A mintapélda összevont állapottáblájának szerkesztése

*(2.72. ábra)*

A maximális ekvivalencia-osztályokat állapotoknak tekintjük, és valamennyire egyenként, bemeneti-kombinációnként előírjuk a következő állapotot azzal, hogy megnézzük, az eredeti állapottábla valamely ebbe az osztályba tartozó állapotának következő állapota melyik osztályba tartozik. A kimeneteket hasonlóan rögzítjük. Példánkban az összevont állapotok jelölése:

**(a c ) → s1**

**(b d ) → s2**

**(e) → s3**

| | X = 0 | X = 1 |
|---|---|---|
| a | **c** / 0 | **b** / 1 |
| b | **d** /1 | **e** / 0 |
| c | **a** / 0 | **d** / 1 |
| d | **b** / 1 | **e** / 0 |
| e | **e** / 0 | **a** / 1 |

| | X = 0 | X = 1 |
|---|---|---|
| s1 | s1/ 0 | s2/ 1 |
| s2 | s2/1 | s3/ 0 |
| s3 | s3/ 0 | s1/1 |

*2.72. ábra. Az összevont állapottábla szerkesztése az összevonás és az előzetes alapján*

## 2.8.2. Állapot-összevonás nem teljesen specifikált szimbolikus előzetes állapottáblán

### 2.8.2.1. Állapot-kompatibilitás nem teljesen specifikált állapottáblán

*Egy nem teljesen specifikált szimbolikus előzetes állapottáblával megadott hálózat (NTSH) adott állapotához tartozó specifikációs bemeneti sorozat az, amelyre a hálózat minden állapotátmenete és kimenete specifikálva van.*

*Két szimbolikus állapot az NTSH állapottábláján csak akkor megkülönböztethető, (MK), ha létezik legalább egy olyan specifikált bemeneti*

*sorozat, amely mindkét állapotra érvényes, és amelynek legalább egy elemére más kimeneti kombináció adódik.*

*Ha ilyen specifikációs bemeneti sorozat nem létezik, a két állapot NMK.*

Az *NTSH* állapot-párjaira érvényes NMK bináris reláció a következő tulajdonságokat mutatja:

- *reflexív*
- *szimmetrikus*
- *nem tranzitív*

Az ilyen relációkat *kompatibilitás-típusú* relációknak nevezzük A *NTSH* állapot-párjaira fennálló *NMK* tulajdonságot röviden *kompatibilitásnak*, illetve a pár tagjait *kompatibilis állapotoknak* fogjuk nevezni.

A kompatibilitás elégséges feltételei:

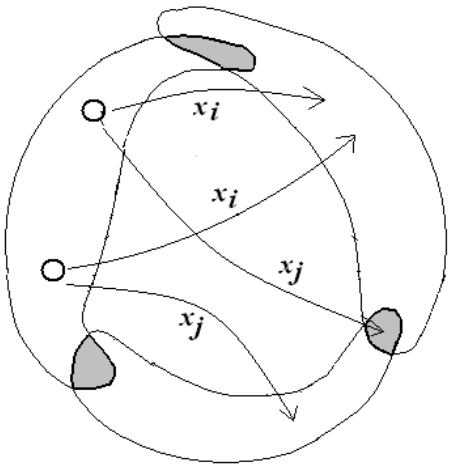
- *Ha nincs olyan bemeneti kombináció, amelyre mindkét állapotból specifikált következő állapot és specifikált kimenet lenne az állapottáblán, ha pedig*
- *létezik minkét állapotra specifikált kimeneti kombinációt és következő állapotot definiáló bemeneti kombináció, akkor arra a két állapothoz tartozó kimeneti kombinációk megegyeznek, és a két állapothoz tartozó következő állapotok kompatibilisek.*

Az előző fejezetben megismert lépcsős táblának természetesen a kompatibilitás vizsgálatakor is fontos szerep jut. A következő jelöléseket fogjuk alkalmazni a táblázat celláiban:
**a~ b** : a *és* b *állapotok kompatibilisek*
**a /~b** : a *és* b *állapotok nem kompatibilisek*
Feltételes kompatibilitás : **ab**, **cd** . . . *az a két állapot, melyekre ez a bejegyzés vonatkozik, feltételesen kompatibilis, azaz csak akkor kompatibilis, ha* **a~b** *és* **c~d**. . .

### 2.8.2.2. A kompatibilitási osztályok *zárt* halmaza

*A fenti kompatibilitási reláció az állapothalmazt nem diszjunkt osztályokra bontja,* azaz lehetnek az osztályoknak közös elemeik is. Egy ilyen osztály valamennyi lehetséges állapot-párjára fennáll a kompatibilitás. Ezek az osztályok akkor maximálisak, ha további elemek egyetlen osztályba sem vonhatók be.
Belátható, hogy a maximális kompatibilitási osztályok *zárt halmazt alkotnak*.
*A kompatibilitási osztályok egy adott halmaza zárt, ha a halmazban szereplő bármelyik osztály tetszőleges két állapotából kiindulva minden olyan bemeneti*

*kombinációra, amely mindkét állapotból specifikált következő állapotot ír elő, a következő állapotok is együtt szerepelnek a halmaznak* ***legalább egy osztályában***. A *2.73. ábra* a zártságot szemlélteti. A bal felső osztályból két állapotot ragadtunk ki. Ezek közül az egyik utód-állapota az $x_j$ bemeneti kombinációra két másik osztály közös részében van, de ezek közül az egyik osztály azonos azzal az osztállyal, amelyben a másik állapot $x_j$-re adott utódja helyezkedik el.




*2.73. ábra. A kompatibilitási halmazok zártságának szemléltetése*

**2.8.2.3. Kevesebb, vagy kisebb elemszámú zárt kompatibilitású osztályok keresése.**

Az *NTSH* állapottáblán végzett állapot-összevonás fő nehézsége az, hogy a lépcsős tábla alapján felvett maximális kompatibilitási osztályok halmazán kívül még más, esetleg kedvezőbb zárt osztály-halmazok is lehetnek. Ezért a maximális kompatibilitási osztályok halmazát felülvizsgáljuk, és megpróbáljuk a *zártság* és az *állapotok teljes lefedésének* fenntartása mellett egyszerűsíteni. Ez utóbbi, mármint a teljes lefedés azt jelenti, hogy minden állapotnak szerepelnie kell legalább egy osztályban. A egyszerűsítés lehet az *osztályok számának csökkentése*, vagy az *osztályok elemszámának* csökkentése.

*ELJÁRÁS A LEGKEVESEBB, LEGKISEBB ELEMSZÁMÚ KOMPATIBILITÁSI OSZTÁLYBÓL ÁLLÓ ZÁRT HALMAZ ELŐÁLLÍTÁSÁRA*

*1. A lépcsős táblából kialakított maximális kompatibilitási osztályok közül válasszuk ki a lehető legkevesebbet úgy, hogy az előzetes állapottábla minden állapota legalább egy osztályban szerepeljen.*

*2.Ha ez a halmaz zárt, hagyjuk el a 3. lépést. Ha nem, csináljuk a 3.-at*

*3.Vizsgáljuk meg, hogy a több osztályban szereplő állapotok egyes osztályokból való elhagyásával zárttá tehető-e a halmaz.*

- *Ha igen, tegyük ezt meg, és ugorjunk a 4.-re.*
- *Ha nem, részben csináljuk vissza amit az 1.es lépésben csináltunk, azaz egészítsük ki a halmazt a zártságot biztosító osztályokkal, és csináljuk 4-t!*

*4. Hagyjuk el az osztályokból a zártság fenntartásával elhagyható állapotokat.*

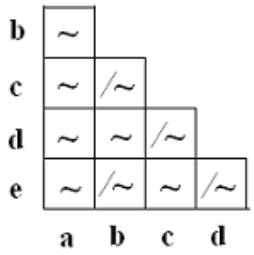
### 2.8.2.4. Az összevont állapottábla szerkesztése

Az összevont állapottábla szerkesztése elvi nehézséget nem okoz. Egy egyszerű alkalmazási példán illusztráljuk az eljárást.

### 2.8.2.5. Példa NTSH állapottáblázaton történő állapot-összevonásra

| X1 X2 bem. / akt. áll | szimb.köv.áll. / kim. | | | |
|---|---|---|---|---|
| | 0 0 | 0 1 | 1 0 | 1 1 |
| a | (a)/0 | b/0 | c/0 | -/- |
| b | a/0 | (b)/0 | -/- | d/0 |
| c | a/0 | -/- | (c)/0 | e/1 |
| d | -/- | b/0 | c/0 | (d)/0 |
| e | -/- | b/0 | c/0 | (e)/1 |

*2.74. ábra. NTSH állapottábla, az állapot-összevonás bemutatására*




kompatibilis párok:
(a b), (a c), (a d), (a e), (b d), (c,e)

maximális kompatibilitási osztályok:
(a b d), (a c e)

*2.75. ábra. A maximális kompatibilitási osztályok megkeresése lépcsős tábla segítségével*

A *2.74.ábra* alapján a 2.75. *ábra* lépcsős táblájából kapott maximális kompatibilitási osztályokból elindítva a leírt eljárás szerinti vizsgálatokat, azonnal belátható, hogy ha bármelyik osztályt elhagyjuk, állapotok maradnak lefedetlenül, tehát marad a közös állapotok elhagyásával való kísérletezés. Két zárt osztályhalmazt kaphatunk így, az **(a,b,d), (c,e),** és a **(a,c,e), (b,d)** osztályhalmazokat. Az első osztályhalmaz zártságáról az állapottábla alapján meggyőződhetünk, és belátható, hogy az **(a,b,d)** minden eleme bemeneti kombinációnként ugyanabba az osztályba képződik le, illetve ez a **(c,e)** osztály elemeire is igaz. Hasonlóan bizonyítható a második osztályhalmaz zártsága is. Ebből az következik, hogy a példának kétféle állapot-összevonása is jó megoldáshoz vezet.
Mindezek alapján két összevont állapottáblát szerkeszthetünk, (*2.76. ábra*) és nekiláthatunk a kódolt állapottábla megszerkesztéséhez és a realizációhoz.
Példánk egyik megoldásában egyébként felismerhetjük a már megvalósított speciális sorrendi ***ÉS*** áramkörünket. Ezúttal egy másik, a korábban bemutatottól eltérő állapot-összevonási lehetőséget is találtunk.

| | 0 0 | 0 1 | 1 0 | 1 1 |
|---|---|---|---|---|
| s1 a b d | (s1)/0 | (s1)/0 | s2/0 | (s1)/0 |
| s2 c e | s1/0 | s1/0 | (s2)/0 | s2/1 |

| | 0 0 | 0 1 | 1 0 | 1 1 |
|---|---|---|---|---|
| s1 a c e | (s1)/0 | s2/0 | (s1)/0 | (s1)/1 |
| s2 b d | s1/0 | (s2)/0 | s1/0 | (s2)/0 |

*2.76. ábra. Sorrendi ÉS két lehetséges állapot-összevonással*

### 2.8.3. Összefoglalás az állapot-összevonási módszerekről

***a.*** *Állapot-összevonás teljesen specifikált előzetes állapottáblából:*

*a.1.* Az ekvivalens és antivalens állapot-párok megkeresése lépcsős tábla segítségével

*a.2.* A maximális ekvivalencia-osztályok meghatározása

*a.3.* A maximális ekvivalencia-osztályoknak megfelelő állapotokkal az összevont állapottábla elkészítése.

***b****. Állapot-összevonás nem teljesen specifikált előzetes állapottáblából:*

*b.1.* Valamennyi kompatibilis és inkompatibilis állapot-pár megkeresése a lépcsős tábla segítségével

*b.2.* A lépcsős tábla alapján a maximális kompatibilitási osztályok megkeresése

*b.3.* A kompatibilitási osztályok legkedvezőbb zárt halmazainak megkeresése

*b.4.* A legkedvezőbb zárt halmaz osztályainak egy-egy állapotot rendelve az összevont állapottábla szerkesztése

## 2.9. ÁLLAPOT-KÓDOLÁSI MÓDSZEREK

Az állapot-kódolási módszerek bemutatásakor élesen szét kell választani a szinkron és aszinkron eseteket. Más a cél az egyiknél, más a másiknál. Szinkron hálózatnál nincs versenyhelyzet veszély, így az állapotkódolás arra irányul, hogy a legegyszerűbb struktúrát alakítsuk ki. Aszinkron hálózatok esetében viszont legfontosabb cél a kritikus verseny- helyzetek elkerülése.

### 2.9.1 Szinkron hálózatok állapot-kódolási módszerei

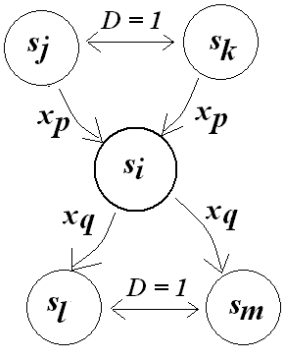
#### 2.9.1.1 A szomszédos állapot-kódok választásának elvei

Belátható, és a gyakorlati tervezés során tapasztalható, hogy az $f_y$ hálózat egyszerűsítésére jótékony hatással vannak a következő állapot-kód viszonyok.

- Egyszerűbb lesz a hálózat, *ha egy adott állapot következő állapotainak kódjai bemenő-kombinációnként szomszédosak*.

- Ugyancsak előnyös, ha *azok az állapotok, amelyek valamely adott állapotnak az előde, bemenő-kombinációnként szomszédos kódúak*.

A *2.77. ábra* bemutatja a szomszédos kódolást kifejező állapot-gráfon a leírt előnyös kódolási helyzeteket. Az ábra jobb oldalán egy előzetes állapottábla látható, amelyen megkísérlünk szomszédos állapot-kódokat találni.




| akt. áll. \ x | köv.állapot 0 | 1 |
|---|---|---|
| a | c | d |
| b | a | e |
| c | a | b |
| d | c | e |
| e | a | b |

*2.77. ábra. Szomszédos állapot-kódok és egy előzetes állapottábla*

A fenti feltételek együtt nyilvánvalóan nem mindig teljesíthetők Ellentmondás esetén az egyszerűbb megoldásra vezető feltétel teljesítését kell előnyben részesíteni. Mivel a K-táblán a szomszédosság előnyösen szemléltethető, a szekunder változók számának megfelelő méretű K-táblán ábrázolhatjuk a feltételek szerint valamilyen mértékben teljesített követelményeket, és az állapotkódokat egyszerűen kiolvashatjuk.

| állapotok | utódok X = 0 | X = 1 |
|---|---|---|
| a | c | d |
| b | a | e |
| c | a | b |
| d | c | e |
| e | a | b |

| állapotok | elődök X = 0 | X = 1 |
|---|---|---|
| a | b,c,e | |
| b | | c,e |
| c | a,d | |
| d | a | |
| e | | b,d |

*2.78. ábra. Utódok és elődök a 2.77 ábra állapottáblája alapján*

A példa szerinti állapottáblát a *2.78. ábrán* látható formába dolgozzuk át, azaz elkészítjük az állapotok utódjait és az állapotok elődeit bemutató táblázatokat. A *2.79. ábra* mutatja az állapotkódok optimális elhelyezését.

Közös utódja van X = 0ra:
b,c
b,e
c,e
a,d

Közös utódja van X = 1-re:
b,d
c,e

Közös elődje van X = 0-ra:
c,d

Közös elődje van X = 1-re
-

Szomszédos kódok:

| *mindkettő* | *közös utód* | *közös előd* |
|---|---|---|
| — | a,d | c,d |
| | b,c | |
| | b,d | |
| | b,e | |
| | c,e | |

| Q3 \ Q1Q2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | e | b | | |
| 1 | c | d | a | |

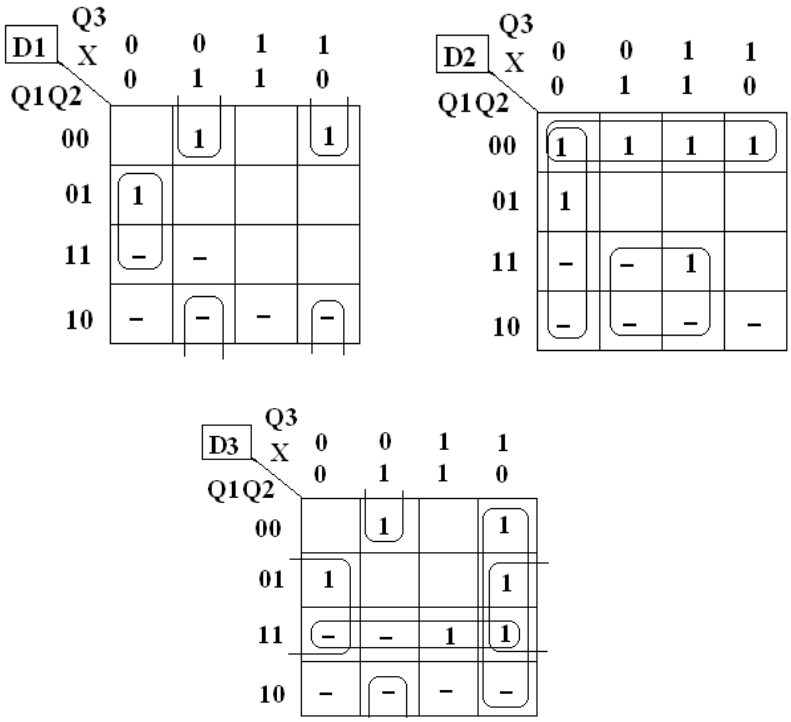
*2.79. ábra. Az állapotok elhelyezése K táblán, az előnyös szomszédosságok legnagyobb arányú biztosítására*

a 111
b 010
c 001
d 011
e 000
100
110
101

| áll. \ $Q_1Q_2Q_3$ | X | $D_1$ $D_2$ $D_3$ | |
|---|---|---|---|
| | | 0 | 1 |
| *a* | 111 | 001 | 011 |
| *b* | 010 | 111 | 000 |
| *c* | 001 | 111 | 010 |
| *d* | 011 | 001 | 000 |
| *e* | 000 | 010 | 111 |
| | 100 | – – – | – – – |
| | 110 | – – – | – – – |
| | 101 | – – – | – – – |

*2.80. ábra. Az állapot-kód és a kódolt állapottábla*

A *2.80. ábra* a kódolt állapottáblát, a *2.81. ábra* pedig a realizáció K-tábláit mutatja.




*2.81. ábra. A realizáció K-táblái*

### 2.9.1.2. A szekunder változók csoportosítása: Önfüggő csoport keresése

Tegyük fel, hogy egy megvalósított szinkron szekvenciális hálózat szekunder változóinak halmazát ***k*** számú csoportra osztjuk, és ennek megfelelően újra jelöljük azokat a következőképpen:

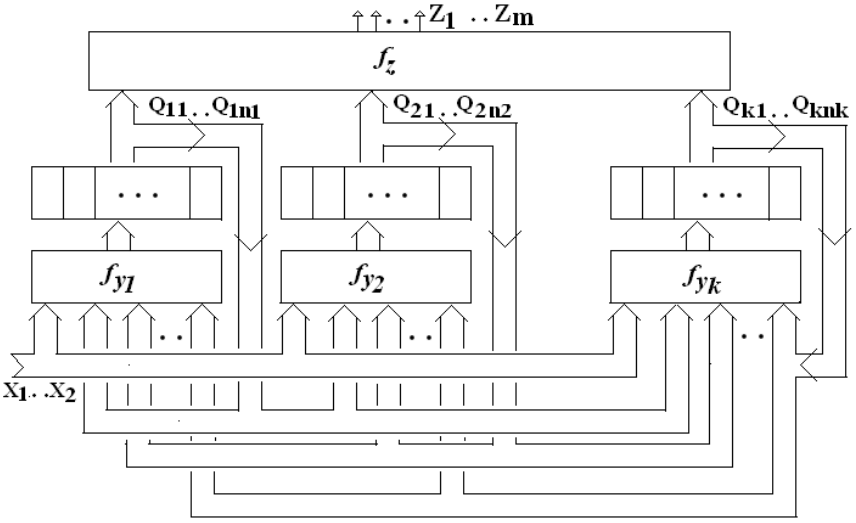
$$(Q_{11}, Q_{12}, \ldots Q_{1n1}, ), (Q_{21}, Q_{22}, \ldots . Q_{2n2}), \ldots . (Q_{k1}, Q_{k2}, \ldots Q_{knk})$$

Az *2.82. ábrán* mutatjuk be, hogyan ábrázolható a hálózat új formájában a csoportosítás figyelembevételével.

Az ábra szerinti struktúra-megfogalmazás azt hangsúlyozza, hogy valamennyi szekunder változó-csoporthoz tartozó $f_y$ függvény-részekre valamennyi csoport tároló-bemenetei vissza vannak csatolva.

**2.9.1.2.1. A szekunder változók csoportosítása alapján definiált ekvivalencia reláció és a komponensekhez rendelt $\Pi_i$ partíció**

Ragadjuk ki az ***i.*** szekunder változó csoportot. (A rövidség kedvéért a szekunder változó csoportok flip-flopjait és a hozzájuk tartozó $f_{y\,i}$ hálózatokat együtt *komponensnek* nevezzük). Vezessünk be az állapothalmazon egy bináris relációt.




*2.82. ábra. A szekvenciális hálózat struktúrája a szekunder változók csoportosításával*

Ez a reláció akkor álljon fenn két állapot között, *ha azokat az* ***i.*** *csoport egyformán kódolja*, azaz nem különbözteti meg.

Belátható, hogy ez a reláció ekvivalencia-típusú, tehát az *állapothalmazt diszjunkt részhalmazokra bontja.* Az is nyilvánvaló, hogy a részhalmazok (osztályok) száma azonos az ***i.*** csoport által megkülönböztetett állapotok számával, azaz azonosíthatók az ***i.*** komponens állapotaival.

Vezessük be e felbontás jelölésére a $\Pi_i$ jelölést, amit az ***i***. szekunder változó csoport *partíciójának* nevezünk. A partíciót úgy adhatjuk meg, hogy az állapotokkal felsoroljuk az osztályait.

Például az ***(a, b, c, d, e, f g)*** állapothalmaz egy partíciója a következő:

$$\Pi = (\,(a, d, e),(b, c),(f),\,(g)\,)$$

Kitüntetett partíciók a $\Pi_0$ , és a $\Pi_e$

$$\Pi_0 = ( (a), (b), (c), (d), (e), (f), (g) )$$

$$\Pi_1 = ( ( a, b, c, d, e, f, g ) )$$

Bizonyítható, hogy az összes komponens partíció metszete a $\Pi_0$ .

**2.9.1.2.2. A komponensekhez tartozó környezeti partíció**

Valamennyi $\Pi_i$ partíció mellé rendelhető egy másik partíció is, amely az ***i.*** komponens környezetét írja le, pontosabban ezt *az a bináris reláció hozza létre, amely egy osztályba sorolja mindazon állapotokat, amelyeket az **i**. komponensre kapcsolódó szekunder változók egyformán kódolnak.* Nyilvánvaló, hogy ennek a környezeti partíciónak a finomsága attól függ, hogy az ***i.*** komponensre kapcsolódó szekunder változók közül mekkora azoknak a száma, amelyek kimaradnak a bemenetek közül. Könnyen belátható az is, hogy amennyiben a hálózat valamennyi szekunder változója rákapcsolódik az ***i.*** szekunder változó csoport bemenetére, ez a környezeti partíció a legfinomabb, azaz $\Pi_0$.

**2.9.1.2.3. A komponensekhez rendelt partíció-párok**

*Valamennyi komponenshez hozzárendelhető a ($\Pi_i^K$, $\Pi_i$) partíció kettős, amelyek együttesen azonosítják a mind az **i**. komponens állapotait, mind az **i.** komponens környezetének állapotait.*

Ennek a párosnak speciális tulajdonsága van, Nevezetesen:

> *A $\Pi_i^K$ egy tetszőleges osztályához tartozó állapotok következő állapotai bemeneti kombinációnként a $\Pi_i$ azonos osztályában vannak.*

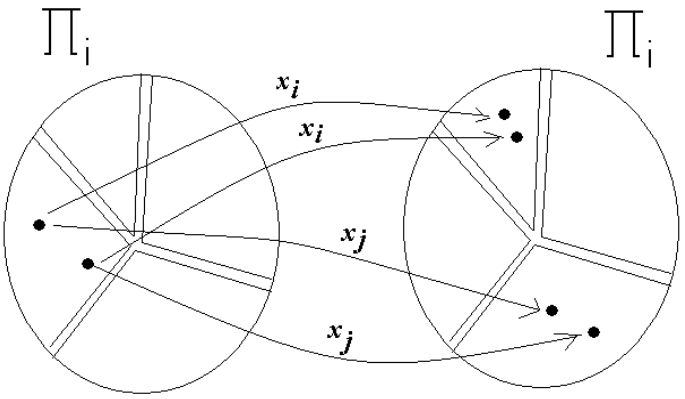
*Ha az állapothalmaz két partíciója között ez a tulajdonság fennáll, akkor a két partíció fentiek szerint rendezett kettősét partíció-párnak nevezzük.*

*A ($\Pi_i^K$, $\Pi_i$)* tehát *partíció-pár.* Ezek szerint *a komponensekre bontott hálózat minden komponenséhez rendelhető egy partíció-pár.*

A partíció-pár alapvető tulajdonságát könnyen magától értetődővé tehetjük. Ez abból következik, hogy a környezeti partíció egy osztályához tartozó állapotokat a komponens nem tudja megkülönböztetni, tehát szükségképpen ugyanarra a bemeneti kombinációra csak ugyanazzal a következő állapottal, azaz saját partíciójának ugyanazzal az osztályával tud válaszolni.

### 2.9.1.2.4. Helyettesítési tulajdonságú partíció

Az állapothalmaz azon partícióit, amelyek önmagukkal partíció-párt alkotnak, helyettesítési tulajdonságú partícióknak nevezzük. A helyettesítési tulajdonságú partíció (*HT*) tulajdonságait szemlélteti a *2.83. ábra.*




*2.83. ábra. Helyettesítési tulajdonságú (HT) partíció szemléltetése*

Ha egy komponens olyan, hogy a hozzá rendelt partíció-pár azonos partíciókból áll, akkor ez azt jelenti, hogy a komponensre csakis saját szekunder-változók vannak visszacsatolva, a többi komponensről egyetlen-egy sem. Ez azért előnyös, mert a komponensek közötti összeköttetések száma jelentősen kisebb lehet, és így a komponens $f_y$ függvénye leegyszerűsödik ahhoz képest, mintha minden egyes komponens minden egyes komponenssel összeköttetésben lenne. Azokat a szekunder változókat, amelyek csak egymással állnak kapcsolatban, önfüggő szekunder változóknak nevezzük.

Ez adja a következőkben ismertetendő állapotkódolási eljárás alapelvét: Keressünk olyan állapothalmaz partíciókat, amelyek nem triviális *HT* partíciók, azaz nem a $\Pi_0$ és nem a $\Pi_1$, és tartozik hozzájuk egy olyan másik partíció is, amellyel alkotott metszetük a $\Pi_0$-t adja. Ha találunk ilyen *HT* partíciókat akkor ezek mindegyike egy önfüggő és egy nem-önfüggő szekunder-változó halmazt definiál. Ezek közül válasszuk azt, amellyel a realizáció egyszerűbb.

### 2.9.1.2.5. Egy HT-partíciós állapotkódolási példa

Végezzünk *HT*-partíciós állapotkódolást a *2.84. ábra.* szerinti állapot-tábla alapján:

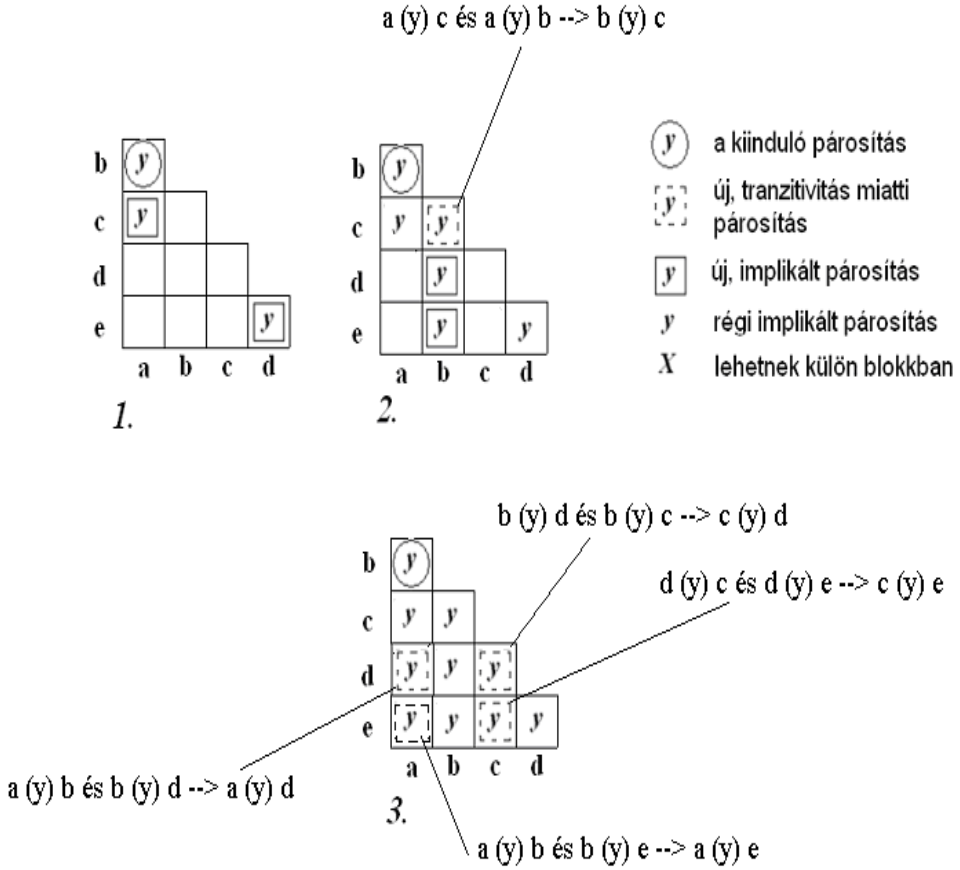
| akt. áll. \ x | köv.állapot | |
|---|---|---|
| | 0 | 1 |
| a | c | d |
| b | a | e |
| c | a | b |
| d | c | e |
| e | a | b |

*2.84. ábra. Állapottábla HT partíciós állapotkódoláshoz*

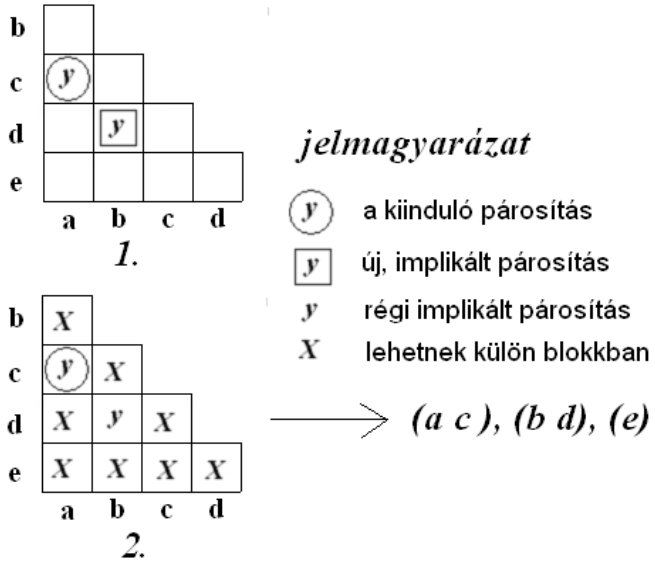
A *HT* partíciók keresését lépcsős táblán végezzük. Először kitöltjük a lépcsős tábla celláit, milyen párosítási feltételek következnek a cellához tartozó két állapot egy osztályban való megjelenéséből. Ezután feltételezzük két állapotról, hogy egy osztályban vannak. (jelölés : ***y***, karikában) Ezután minden olyan cellát „***y***”-vel jelölünk, amelyekhez tartozó állapotok összetartozását a kiindulási párosítás implikálja. A jelöléskor figyelembe kell vennünk a tranzitivitást is.
Ha már nincs újabb implikált pár, az üresen maradt cellákba X – t teszünk.
Ezután az ismert eljárással felvesszük a nem-diszjunkt osztályokat, majd a tranzitivitás alapján egyesítjük azokat, amelyeknek van közös elemük.
Ha az összevonás után a teljes állapothalmazt kapjuk, ez triviális (egyblokkos) *HT* partíció, és nem használható. Ilyenkor új kiindulási párt kell választani, és az eljárást erre az új párra kell megismételni. Ha valamennyi lehetséges nem triviális HT partíciót előállítottuk, válogatunk.
*Világos, hogy egy adott HT partíciót kiválasztva annak blokkszáma (**B**) adja az önfüggő csoport állapotainak számát, a blokkokban előforduló maximális állapotszám (**A**) pedig a maradék szekunder változó-csoport által képviselt állapotszámot adja. Így a HT partícióhoz szükséges szekunder változók száma,*

$$p = \log_2 B + \log_2 A.$$

A *2.85. ábra* lépcsős első tábláit először azzal a feltételezéssel töltjük ki, hogy az **a** és a **b** állapotok egy osztályban vannak. A következményeket látjuk az *1*-es jelű táblán, majd tovább lépve a *2*-es és *3*-as táblákra, látjuk, hogy triviális partíciót kaptunk, hiszen az adódott, hogy minden állapot ugyanabba az osztályba tartozik. A *2.86. ábra* szerinti feltételezés, miszerint az **a** és a **c** legyenek egy osztályban, nem triviális *HT* partícióra vezet : **(a c), (b d), (e)**




*2.85. ábra. Az a és b ekvivalenciájának feltételezése triviális HT partícióra vezet*




*2.86. ábra. Ekvivalencia-osztályok az (a c) párosításból kiindulva*

A *2.87 ábra* szerinti állapotkód önfüggő csoportjában a szekunder-változók száma *2,* hiszen három osztályt kaptunk. Ezeken az osztályokon belüli kód finomításához már csak egyetlen változó kell, hiszen a legnépesebb osztály állapot-száma *2*. Az állapotok kódjainak megválasztásakor egyetlen szabálya, hogy az egy osztályban szereplőket az önfüggő csoport állapot-változói egyformán kódolják.

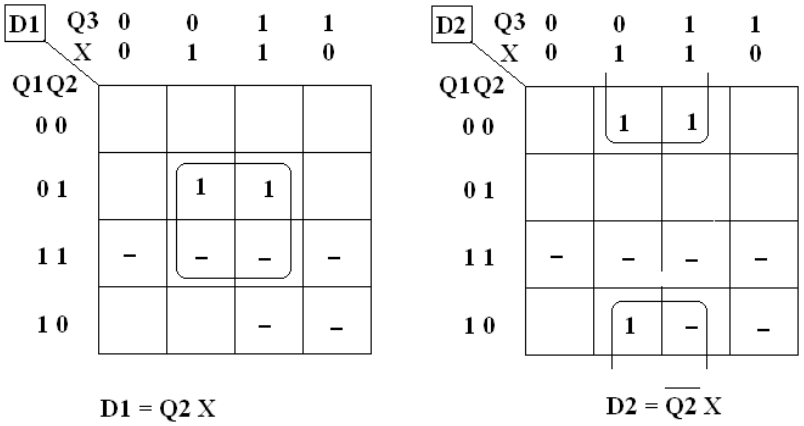
| kód / áll. | Q1 | Q2 | Q3 |
|---|---|---|---|
| *a* | *0* | *0* | *0* |
| *c* | *0* | *0* | *1* |
| *b* | *0* | *1* | *0* |
| *d* | *0* | *1* | *1* |
| *e* | *1* | *0* | *0* |

*2.87. ábra. Állapotkód, önfüggő változókkal*

A *2.88.-2.89. ábrák* a realizációt mutatják. Látható, hogy sem a ***D1***, sem a ***D2*** nem függ ***Q3***-tól. Az „önfüggés” tehát igazolódott.

| áll. | X | $D_1$ $D_2$ $D_3$ | |
|---|---|---|---|
| | $Q_1 Q_2 Q_3$ | 0 | 1 |
| *a* | 0 0 0 | 0 0 1 | 0 1 1 |
| *b* | 0 1 0 | 0 0 0 | 1 0 0 |
| *c* | 0 0 1 | 0 0 0 | 0 1 0 |
| *d* | 0 1 1 | 0 0 1 | 1 0 0 |
| *e* | 1 0 0 | 0 1 0 | 0 0 0 |
| | 1 0 1 | – – – | – – – |
| | 1 1 0 | – – – | – – – |
| | 1 1 1 | – – – | – – – |

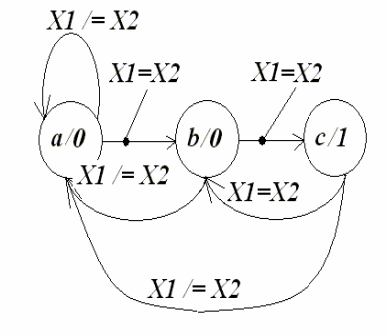
*2.88. ábra. Kódolt állapottábla*




*2.89. ábra. Az önfüggés igazolása K-táblákkal*

### 2.9.1.3. Szinkron hálózatok 1-es súlyú állapotkódolással

Szinkron hálózatok VLSI megvalósításakor gyakran igen gyorsan célravezető egy olyan állapotkód, amikor minden egyes szimbolikus állapothoz egy ***D–MS*** flip-flopot rendelünk. Minden egyes állapot kódjában az ***1***-esek száma legyen egy. Ezért nevezzük ezt ***1***-es súlyú kódnak.




| akt. áll./ /kim. | köv.állapot | |
|---|---|---|
| | X1/=X2 | X1=X2 |
| a/0 | a | b |
| b/0 | a | c |
| c/1 | a | b |

*2.90. ábra. Egy korábbi feladat Moore-típusú realizációjának állapot-gráfja és előzetes szimbolikus állapottáblája, 1-es súlyú, D-flip-flopos implementációra*

A *2.90. ábrán* látható állapotgráf és tábla alapján így megvalósítandó ***1***-es súlyú állapotkód három **D-MS** flip-flopot igényel. Feltéve, hogy e az bemenetek **EXOR** függvénye, a kódolás egy lehetséges változata a következő:

| | Qa | Qb | Qc |
|---|---|---|---|
| **a** | **1** | **0** | **0** |
| **b** | **0** | **1** | **0** |
| **c** | **0** | **0** | **1** |

A **D** bemenetek vezérlésének megvalósítása ilyenkor rendkívül egyszerű, és az állapotgráf alapján elvégezhető.

Minden egyes **D** bemenetre akkor és csak akkor kell ***1***-t kapcsolni, amikor az általa reprezentált tároló kimenetnek **0**-ról **1**-re kell változnia, azaz amikor az adott flip-flop által reprezentált állapotnak be kell állnia. Ez pedig az

állapotgráfból kiolvasható. További előny, hogy az alapállapotot beállító **R** jelet is azonnal „bedolgozhatjuk”.

$$D_a = R + \overline{R}(Q_a\ \overline{e} + Q_c\ \overline{e}) = R + \overline{e}\,(Q_a + Q_c)$$

$$D_b = \overline{R}\,e\,(Q_a + Q_c)$$

$$D_c = \overline{R}\,Q_b\,e$$

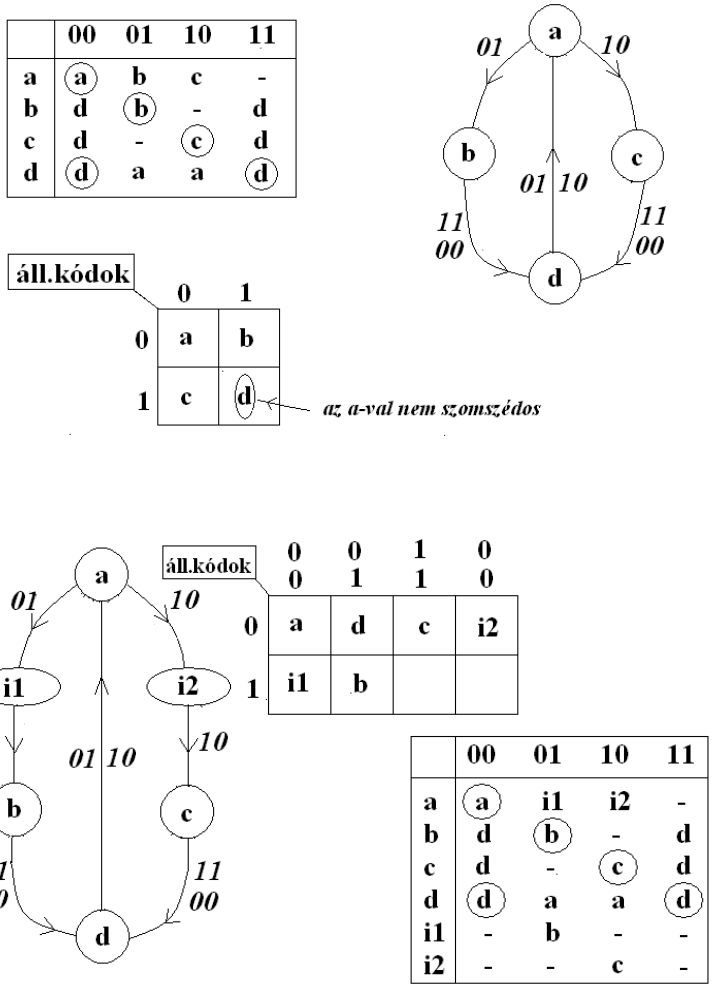
*FELADAT:*
K-tábla segítségével lássuk be, hogy a követett eljárásunk korrekt. Ne feledkezzünk meg a nem használt kódok következtében előálló közömbös következő-állapot kódokról!

### 2.9.2. Aszinkron sorrendi hálózatok állapotkódolása

Aszinkron hálózatok állapotkódolásakor sajnos nem a legolcsóbb megoldás magtalálása a legfontosabb probléma. A kritikus versenyhelyzetek elkerülése, tehát kritikus versenyhelyzet mentes kódolás minden más szempontot megelőz, hiszen ezen nem az áramkör ára, hanem a működőképessége múlik. Itt is két módszert mutatunk be. Az egyik inkább próbálgatásos, intuitív módszer, a másik elméletileg megalapozott, szisztematikus eljárás.

#### 2.9.2.1. Instabil állapotok beillesztése a kritikus versenyhelyzetek kiküszöbölésére

Már ismert módszer, hogy minden állapot-átmenetre biztosítjuk a kódok egyetlen szekunder változó értékében való változást, hiszen a kritikus versenyhelyzet forrása éppen a többszörös változás. Néhány tervezési feladatunkban ezt a módszert alkalmaztuk az egymást követő stabil állapotok kódjának megválasztásakor. Vannak azonban esetek, amikor az állapotok számának kettes alapú logaritmusa és a szekunder változók száma közeli értékek, így „be vagyunk szorítva”, és az ilyen kódolás nem is létezik. Ilyenkor növelni kell a szekunder változók számát. Segítségükkel *instabil állapotokkal bővítjük* az összevont állapottáblát. Az instabil állapotokkal való bővítést úgy kell megoldani, hogy a stabil állapotok egymás utáni sorrendje ne változzék, ugyanakkor az új instabil állapotokat úgy kell kódolni, hogy az egymás után következő tranziens kódok között csak egy szekunder változó értékében legyen különbség.
Az eljárást egy igen egyszerű példán szemléltetjük (*2.91 ábra*).




*2.91 ábra. Bővítés szomszédos kódú instabil állapotokkal*

Az előzetes állapottábla alapján felvett állapot-gráf mutatja, hogy az **a** állapottal a **d** állapot kódját nem lehet szomszédossá tenni, ha ragaszkodunk a két szekunder változóhoz. Be kell szúrnunk két instabil állapotot (**i1 és i2**), és

ezzel egy új szekunder változót is! Az így kibővített állapot-halmaz elemeinek kódjait már meg lehet úgy választani, hogy az egymás után következő tranziens állapotok kódjai szomszédosak legyenek. Hangsúlyozzuk, hogy sem az **i1**, sem az **i2** állapot soha sem stabilizálódik, de áthidaló tranziens szerepet töltenek be. Ez jól követhető az új állapot-átmeneti táblázat segítségével is.

#### 2.9.2.2. Tracey és Unger módszere a kritikus versenyhelyzetek kiküszöbölésére

Kritikus versenyhelyzet akkor áll elő, ha egy stabil állapotból kiindulva megváltoztatjuk a bemeneti kombinációt, és ennek hatására olyan átmeneti állapotkód áll elő, amelynek sorában és az adott bemeneti kombináció oszlopában ez az állapotkód szerepel.
A nem kívánt átmeneti állapotkódot HAZÁRD-KÓDNAK nevezzük.
A TRACEY-UNGER módszer lényege, hogy a normális (tervezett) állapotátmenethez tartozó kiinduló és cél állapotok kódjai legalább *egy adott szekunder változóban mindketten különbözzenek a hazárd kódtól*.
Ilyenkor ugyanis ez a szekunder változó az átmenet során állandó marad, és így soha sem áll elő a hazárd állapot kódja.
A kódolási eljárást az összevont szimbolikus állapottábla analízisével kezdjük. Egy listára felvesszük a stabil-stabil állapot-átmeneteket, és hozzájuk írjuk azoknak a „leselkedő" hazárd állapotoknak a nevét, amelyek az átmenet során felléphetnek, azaz azokat, amelyek a stabil célállapot oszlopában szerepelnek. Egy ilyen átmenet és a leselkedők listája egy kódolási szabályt definiál. Ez úgy hangzik, *hogy a kiindulási és a cél állapot legalább egy szekunder változó értékében egyezzen, de ebben a változóban mindketten különbözzenek a „leselkedőktől"*.
Miután az összes állapotátmenetet ilyen módon listáztuk, akkor ***M*** számú kódolási szabályunk lesz.
Rendeljünk minden szabályhoz egy szekunder változót, és a szekunder változókkal oszlopokat, a kódolandó szimbolikus állapotokkal sorokat alkotva írjunk fel olyan kódot, amely a megállapított kódolási szabályoknak eleget tesz. A szabályok csak a kötelező különbségtételt írják elő, azt hogy melyik szekunder változó legyen **1** és melyik legyen **0**, azt nem. Ügyeljünk azonban arra, hogy ahol az adott szabály nem kényszerít az adott állapotváltozóra értéket, oda közömbös bejegyzést tegyünk.
Belátható, hogy a közömbös bejegyzések tetszőleges konkretizálásával máris kritikus versenyhelyzettől mentes állapotkódot kaptunk, de túl sok szekunder változó bevezetése volt az ár. Ugyanakkor felismerhetjük, hogy a közömbös bejegyzések kihasználásával az oszlopokat összevonhatjuk. Ha az összevonás nehézségekkel jár, cseréljük fel szabadon az egyes oszlopokban az **1** és a **0**

bejegyzéseket, és próbálkozzunk újra az összevonással. Tekintsük most a *2.92. ábra* szerint előzetes állapot-táblát.

| X1X2 / áll. | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a | ⓐ/0 | c/- | b/0 | ⓐ/0 |
| b | d/- | d/- | ⓑ/0 | ⓑ/0 |
| c | d/1 | ⓒ/1 | ⓒ/1 | ⓒ/1 |
| d | ⓓ/1 | ⓓ/1 | b/- | a/- |

*2.92. ábra. Szimbolikus előzetes állapot-tábla a Tracey-Unger módszer bemutatására*

Első lépésként analizáljuk a a stabil-stabil állapot-átmeneteket, és listázzuk a hozzájuk tartozó lehetséges hazárdokat, a „leselkedőket”:

ANALÍZIS :
Bemenő kombináció változásonként vizsgáljuk az állapot-átmeneteket, és a „leselkedő” hazárd állapotokat. Az analízis eredményét a *2.93. ábra* ***a*** táblázata mutatja. Láthatjuk például, hogy a **00 → 01** megengedett bemeneti váltáshoz egyetlen stabil-stabil állapot-átmenet olvasható ki, nevezetesen az **a** állapotból a **c** állapotba való átmenet. Ezt egyedül a **d** állapot tranziens kódja veszélyezteti, tehát itt a leselkedő állapot a **d**. Az ábra ***b*** táblázatán az ennek megfelelő kódolási szabályt az **Y1** szekunder változó képviseli, mégpedig azzal, hogy az oszlopában feltüntetett állapot-kódban mind az **a**, mind a **c** **0**-val jelenik meg, a leselkedő **d** viszont **1**-vel! A **b** állapot **Y1** pozíció-beli kód-része közömbös. Így szerkesztjük végig a táblázatot, ismétlésekbe nem bocsátkozunk, és a fordított irányú de azonos leselkedőt mutató átmeneteket is értelem-szerűen csak egyszer tüntetjük fel (ld az áthúzott táblázat-elemek).
A ***c*** táblázat az összevont oszlopokat mutatja. Az összevont állapot-változók indexei mutatják, mely oszlopokat sikerült összevonni. Ezzel kialakult, hogy a versenyhelyzet-mentes kód végül is négy szekunder változóval biztosítható.

| 00→01 | | 00→10 | | 01→00 | | 01→11 | | 10→00 | | 10→11 | | 11→01 | | 11→10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a→c | *d* | d→a | *b*<br>*c* | c→d | *a* | d→b | *c* | b→d<br>c→d | *a*<br>*a* | a→b | *c* | b→d | *c* | |

*a.*

| sz.v. / áll. | Y1 | Y2 | Y3 | Y4 | Y5 | Y6 |
|---|---|---|---|---|---|---|
| a | 0 | 0 | 0 | - | 0 | 0 |
| b | - | 1 | - | 1 | 1 | 0 |
| c | 0 | 1 | 1 | 0 | - | 1 |
| d | 1 | 0 | 1 | 1 | 1 | - |

*b.*

| sz.v. / áll. | Y14 | Y36 | Y2 | Y5 |
|---|---|---|---|---|
| a | 0 | 0 | 0 | 0 |
| b | 1 | 0 | 1 | 1 |
| c | 0 | 1 | 1 | - |
| d | 1 | 1 | 0 | 1 |

*c.*

*2.93. ábra. A Tracey-Unger módszer végrehajtása során keletkező táblázatok*

*TERVEZÉSI FELADATOK 4.*

**1.** Minimális számú állapottal tervezzük meg a következő állapottábla szerinti szinkron sorrendi hálózatot élvezérelt **, *Preset és Clear*** bemenetekkel nem rendelkező ***J-K MS*** flip-flopokkal. A kezdő-állapot ***a*** legyen.

| | ***X = 0*** | ***X = 1*** |
|---|---|---|
| ***a*** | ***c / 0*** | ***d / 1*** |
| ***b*** | ***f / 0*** | ***d / 0*** |
| ***c*** | ***a / 0*** | ***b / 1*** |
| ***d*** | ***e / 0*** | ***b / 0*** |
| ***e*** | ***f / 0*** | ***c / 1*** |
| ***f*** | ***e / 0*** | ***a / 1*** |

**2**. Tervezzük meg ***Preset és Clear*** bemenetekkel nem rendelkező ***D-MS*** tárolókkal azt a két bemenetű szinkron hálózatot, amely a ***Z*** kimeneten ***1***-t szolgáltat, ha a bemenetén azonos szint jelenik meg, mint a megelőző órajel ciklusban. A kezdeti állapotot egy ***R*** jellel kell beállítani

**3.** Egy szinkron hálózat önfüggő szekunder-változói azonos kóddal jeleníti meg az ***(a, c, e)***, egy másik kóddal a ***(b, d, f),*** és egy harmadik kóddal a ***(g, h, i, j)*** állapotokat. Írjunk fel egy lehetséges, a fentieknek eleget tevő állapotkód sorozatot úgy, hogy a kódszavak súlyainak összege a lehető legkisebb legyen.

**4.** Egy szinkron hálózat tervezésekor talált helyettesítési tulajdonságú partícók a következők :

***(a, c, e, j), (b, d, , f, i), ( g, h)***

Kódoljuk a hálózatot a lehető legkisebb súlyú kóddal!

**5.** Egy szinkron hálózat tervezésekor talált helyettesítési tulajdonságú partícók a következők :

***1. (a, c, e), (b, d, f), ( g, h)***
***2. ( a, e), ( c, b), (d, f), (g,h)***

Kódoljuk a hálózatot a kedvezőbbel!

**6.** Egy szinkron hálózat szimbolikus, összevont állapottábláján a következő két helyettesítési tulajdonságú partíciót találtuk:

***( a c e j ) ( b d f i ) ( g h )***
***( a b g ) ( c d h ) ( e f) ( i j )***

Végezzünk el ennek alapján egy *két önfüggő csoportot* megvalósító állapotkódolást!

**7.** Egy szinkron hálózat önfüggő szekunder-változói azonos kóddal jeleníti meg az ***(a, c, e)***, egy másik kóddal a ***(b, d, f),*** és egy harmadik kóddal a ***(g, h, i, j)*** állapotokat. Írjunk fel egy lehetséges, a fentieknek eleget

tevő állapotkód sorozatot úgy, hogy a kódszavak súlyainak összege a lehető legkisebb legyen.

**8.** Kombinációs hálózat közvetlen visszacsatolásával tervezzük meg azt az *egy bemenetű* és *egy kimenetű* aszinkron hálózatot, amely az ***X*** bemenet *minden második felfutó-élére* ***1***-t szolgáltat a ***Z*** kimeneten, majd az ezután következő *második lefutó élre* a kimenet visszaáll ***0***-ra. Ezután ***X*** újabb változtatásával a ciklus ismétlődik. Ügyeljünk a kritikus versenyhelyzetek elkerülésére!

## *3. Rész. Összetett digitális-hálózati egységek*

A következő fejezetekben - a kapuszintű tervezésnél magasabb szintű tervezési eljárásokat megalapozandó – áttekintjük a fő építőelemek jellemzőit, alkalmazási területeit és azok kapuszintű felépítését. A magasabb szintű egységek feladatuk szerint csoportosítva:

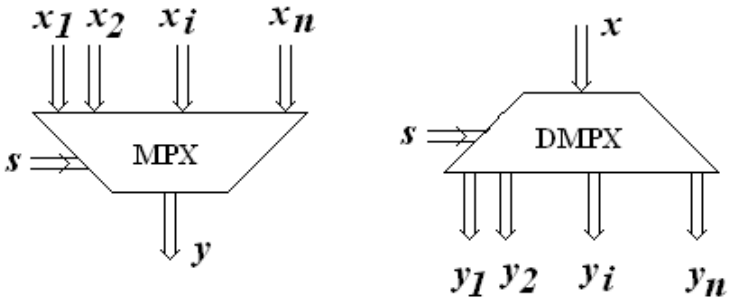
● ***multiplexerek, demultiplexerek***
*adatút szakaszokat jelölnek ki, a*

● ***regiszterek***
*adatokat tárolnak, és ezek elérését is biztosítják, és a*

● ***funkciós egységek***
*adatok közötti műveleteket végeznek.*

### 3.1. MULTIPLEXEREK, DEMULTIPLEXEREK

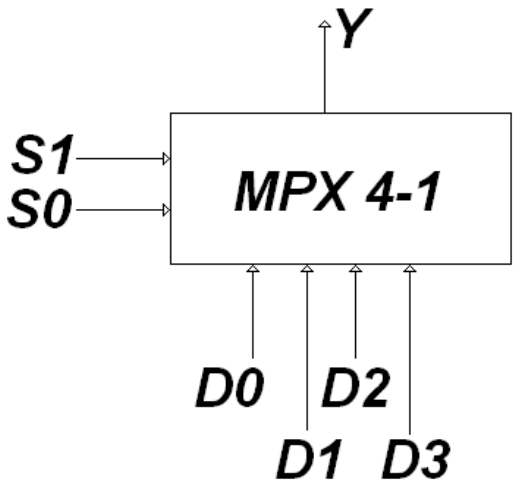
A multiplexereket és a demultiplexereket adat-utak kijelölésére használjuk. Az adat-utak kijelölése vezérlő-bemenetek segítségével, lényegében címzéssel történik. Multiplexereknél több forrás közül kijelöljük azt, amelynek adata a kimenetre kerül, míg a demultiplexerek esetén azt a kimenetet címezzük meg, amelyen az egyetlen forrás adatát meg kívánjuk jeleníteni. A címzés általában bináris kóddal történik, így eleve kizárt az ellentmondásos kettős címzés. A *3.1. ábra* mutatja, a multiplexerek, demultiplexerek szokásos szimbólumait az adat és a vezérlő vektorokkal.




*3.1. ábra. A multiplexerek és a demultiplexerek RT-szintű ábrákon használatos szimbólumai*

### 3.1.1. Egykimenetű, 4 bemenetű MPX

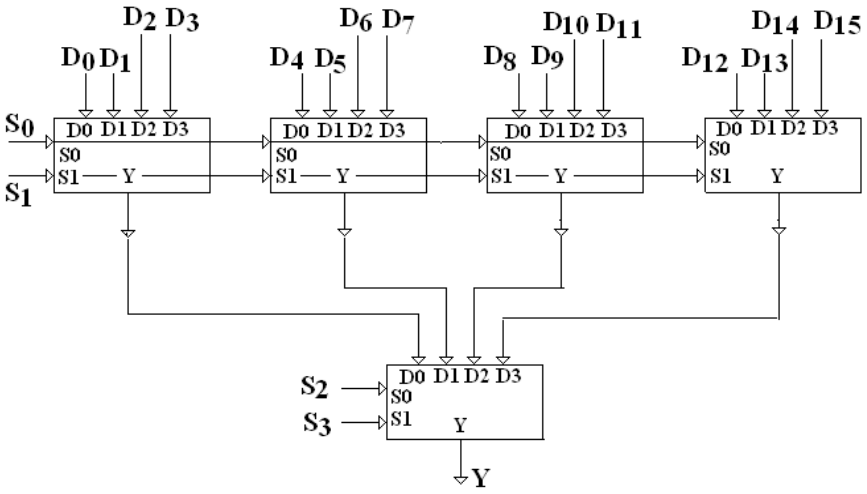
A közepes integráltságú logikai áramkör családok kifejlesztői leginkább egy négy bit adat-bemenettel és egy-bit adatkimenettel, valamint ennek megfelelően két-bit címző bemenettel rendelkező alaptípust gyártottak és gyártanak, ez pedig többféle módon bővíthető. (*3.2. ábra*)




*3.2 ábra Tipikus közepes integráltságú 4-1(négyből-egy) multiplexer szimbóluma.*

### 3.1.2. Bővítés a bemeneti adatok számának növelése céljából

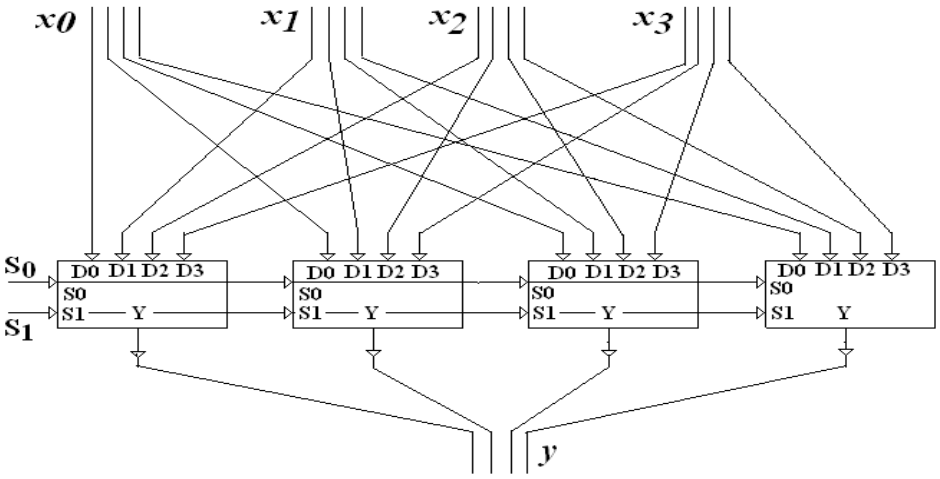
Ha a bemeneti adatunk négynél több bitből áll, úgy a *3.3. ábra* szerinti bővítési sémát alkalmazzuk. Láthatjuk, itt *16-1 multiplexert* alkottunk *4-1* egységekből. Látható, hogy nemcsak vízszintes irányban kellett bővíteni az egységet, hanem mélységében is. Felhívjuk a figyelmet az első sor összekötött címző-bemeneteire, amelyen megjelenő cím-vektor rész mind a négy egység bemenetei közül ugyanazt a sorszámú bemenetet címzi meg, így a második sor egyetlen egységének cím-vektora ebből a négy adatból választ ki egyet.




*3.3. ábra. 16-1 MPX építése öt 4-1-es egységből.*

### 3.1.3. Bővítés sínek közötti választás céljából (BUS-MPX)

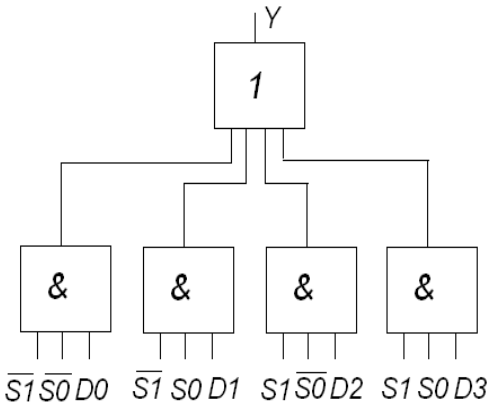
A négy, négy-bites sínből egyet kiválasztó multiplexer összeállítása vízszintes irányú bővítéssel oldható meg *(3.4 ábra).* Természetesen gyakran kell a kétféle bővítési módszer kombinációját alkalmaznunk.




*3.4. ábra. Négy, 4-bites sínből egyet kiválasztó multiplexer 4-1 alapegységekből.*

### 3.1.4. A multiplexerek felépítése

A multiplexerek ***ÉS-VAGY*** illetve ***NÉS-NÉS*** kétszintű kapuhálózatokkal építhetők fel. A ***4-1*** MPX négy ***3*** bemenetű ***ÉS*** kapuból és egy ***4***-bemenetű ***VAGY*** kapuból, illetve az ezeknek megfelelő bemenetszámú NÉS kapukból áll. A kiválasztó bemenetek negált szintjei természetesen invertereket igényelnek. (*3.5. ábra*) A multiplexer ***NÉS-NÉS*** formában is felrajzolható.



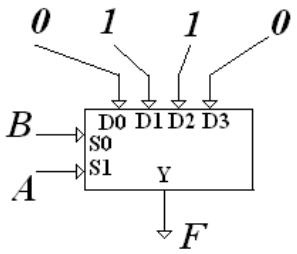

*3.5. ábra. A 4-1 multiplexer egység kapuszintű struktúrája*

### 3.1.5. A multiplexer, mint programozható logikai hálózat

A bit-szintű multiplexerek ***ÉS-VAGY*** struktúrája lehetővé teszi, hogy azokat függvények megvalósítására használjuk fel. Ilyenkor a függvény mintermjeit a címző-bemenetekre adott címek képviselik, és a megcímzett adat-bemenetre rá kell kapcsolnunk az adott mintermhez tartozó logikai értéket. Ezeknek a logikai konstansoknak a bemenetekre való kapcsolását a multiplexer programozásának tekinthetjük. Példánkon, a *3.6. ábrán* egy két-bemenetű ***EXOR*** függvény multiplexeres megvalósítását láthatjuk.

$$F = A\,\overline{B} + \overline{A}\,B$$

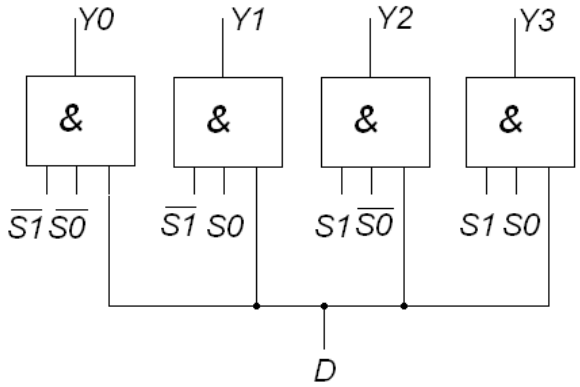
| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




*3.6. ábra. EXOR függvény 4-1 multiplexerből.*

### 3.1.7. Demultiplexerek

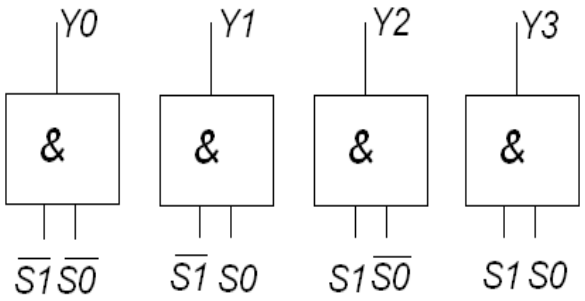
A demultiplexerek funkciója egy adat több lehetséges irány egyikébe történő továbbítása. Az *1-4* (egyet a négyből az egyikre) méretű demultiplexerhez négy, *3*-bemenetű ***ÉS*** kapu szükséges. (*3.7. ábra*)




*3.7. ábra 1-4 demultiplexer kapu-szintű struktúrája.*

### 3.1.8. Dekóderek

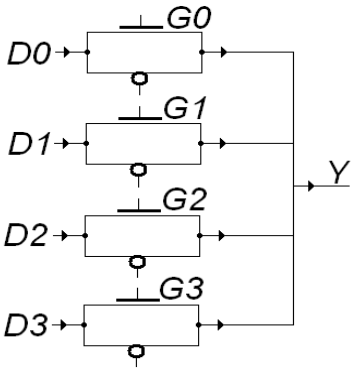
Ha egy *1-4* demultiplexer adatbemenetét állandó, logikai ***1*** szintre kapcsolunk, akkor ez egyenértékű azzal, hogy az ***ÉS*** kapuk bemenetei közül elhagyjuk az adatbemenetet. Az ilyen áramkör sajátossága, hogy a kiválasztó bemenetekre kapcsolt kombinációk csak egyetlen, a kiválasztó kódnak megfelelő kimeneten eredményeznek magas szintet. Az ilyen áramköröket *dekódereknek* nevezzük. (*3.8. ábra*)




*3.8. ábra. 2-bemenetű dekóder, ÉS kapukkal*

### 3.1.8. Multiplexerek és demultiplexerek CMOS átvivő kapukkal

A CMOS *átvivő-kapu*, (*transmission-gate, transfer-gate*) két **MOSFET** eszközből álló kapcsoló. A **MOSFET** eszközök analízisével belátható, hogy egy két feszültség szintet tartalmazó logikai rendszerben csak a párhuzamosan kapcsolt két **MOSFET** együttesen képes átvinni mindkét logikai szintet egyik oldalról a másikra. A vezérlőelektródákat bekapcsoláskor ellentétesen kell vezérelni, azaz a kis körökkel jelölt vezérlő bemenetek a ***G0 - G3*** vezérlők negáltjai. Az átvivő-kapukból felépített multiplexer (*3.9. ábra*) jellegzetessége, hogy kimenete képes a logikai harmadik állapot felvételére, azaz ha egyik kapcsolót sem nyitjuk, a kimenet „lebeg”

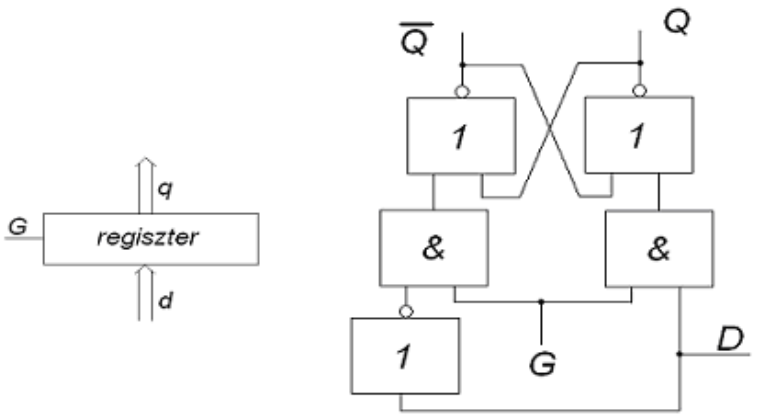



*3.9. ábra. C-MOS átvivő-kapus multiplexer*

## 3.2. REGISZTEREK

### 3.2.1. Szintvezérelt statikus regiszter modell

Erre a regiszterre egy bemeneti bit-vektor (***d)*** és egy logikai beíró-jel, (***G***) csatlakozik. A regiszter a ***G*** beíró jel magas szintjére a ***d*** értékét a tárolóba írja. A regiszter átlátszó, azaz amíg a ***G*** jel magasan van, ***d*** változásai késleltetve megjelennek. ***G*** lefutása után az utolsó, még hatásos bemeneti érték marad a regiszterben. A *3.10. ábrán* a regiszter szimbólumát, valamint egy-bitjének kapu-szintű struktúráját látjuk.

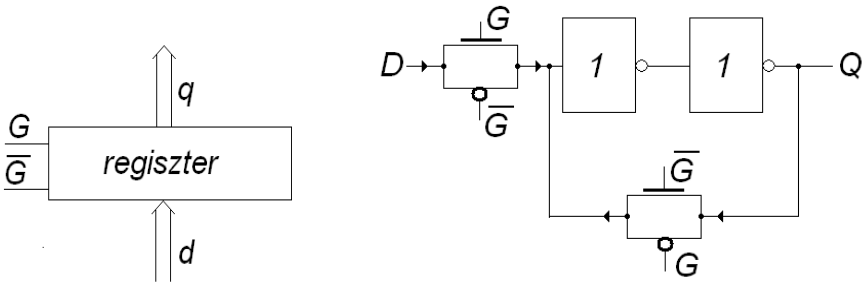



*3.10. ábra. Szintvezérelt regiszter szimbóluma és egy bitjének belső felépítése*

A kapu-szintű struktúrában visszaköszön a már korábban megismert aszinkron ***D-G*** tároló!

### 3.2.3. Szintvezérelt regiszter modell ponált és negált beírójellel

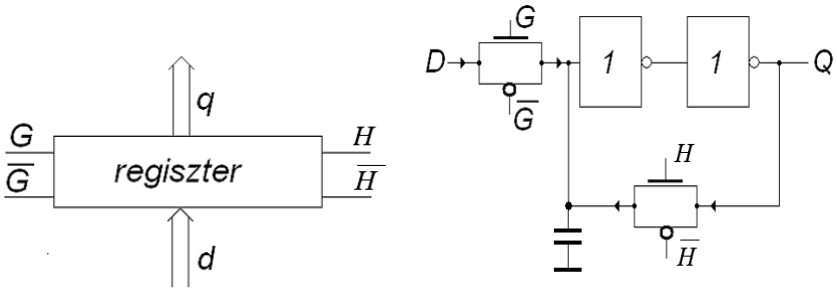
A CMOS technikában, különösen a ***VLSI*** áramkörökben előszeretettel alkalmazzák az átvivő-kapus tárolókból álló statikus regisztert. Az egy bitnyi tároló (***latch***) a két stabil állapotú (***bistabil***) inverter-gyűrű beírásának egy más módszerét alkalmazza, mint a ***NOR***-bistabilok. A beírás (***G=1***) alatt a visszacsatolás meg van szakítva, hiszen a visszacsatoló átvivő-kapu a ***G=0***-nál van bekapcsolva. Ez a beírás után azonnal bekövetkezik, és megvalósul a tárolás. Ennek megfelelően a regiszter bemenetei között a ***G*** vezérlő-vezetéknek mind a ponált, mind a negált változata megjelenik. (*3.11. ábra*)




*3.11. ábra. Ponált és negált beíró jellel vezérelt C-MOS regiszter szimbóluma és kapu-szintű struktúrája.*

### 3.2.4. Kvázistatikus regiszter modell

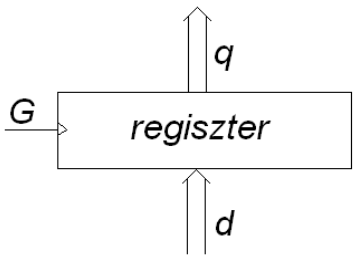
A kvázistatikus regiszter alapcellája (*1-bites egysége*) a CMOS inverter bemeneti kapacitásának átmeneti töltés-tároló képességét használja ki. Itt a ***G*** beírójel két felfutása, azaz két beírás között egy tartó (***H, HOLD***) impulzus rendszeres jelentkezése szükséges. A ***H*** impulzusok között a bemeneti kapacitás tárolja az utoljára beírt szintet, a két inverter pedig regenerálja azt. A *3.12. ábra* a kvázistatikus regiszter szimbólumát és egy-bitnyi struktúráját mutatja.




*3.12. ábra Kvázistatikus regiszter szimbóluma és egy-bitnyi struktúrája.*

### 3.2.5. Élvezérelt regiszter modell *(3.13. ábra)*

Az élvezérelt regiszterekben az átlátszóság a beíró-jel valamelyik éléhez kötődik. Ez lehet a beírójel felfutó éle, de lehet a lefutó él is. Az élvezérelt regiszterekből felépített digitális rendszer kevésbé érzékeny az órajelek időbeli elcsúszásából adódó aszinkronitásokra. Az élvezérlést a beírójel bemenetre elhelyezett speciális szimbólum jelzi. Elfogadott, hogy a felfutó-élre való átlátszóságot a beíró-jel ponált formája jelöli. A cella kapuszintű bemutatásától annak bonyolultsága miatt itt eltekintünk.

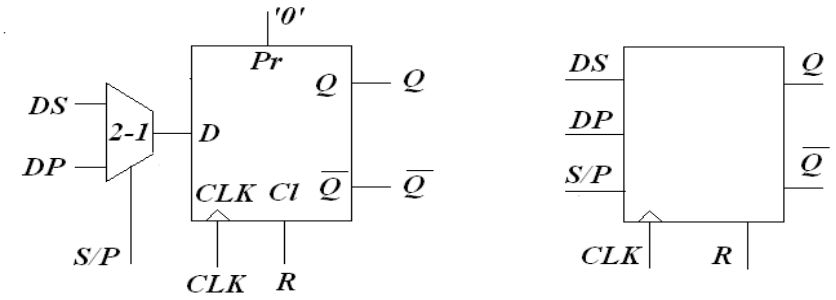



*3.13. ábra. Felfutó élre beíró regiszter szimbóluma*

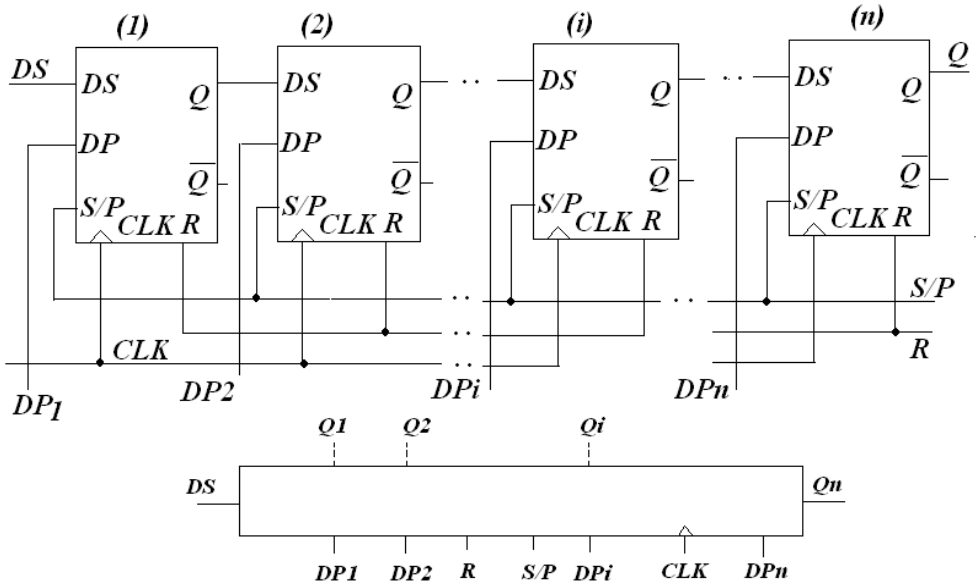
## 3.3. SOROS ELÉRÉSŰ TÁROLÓK

A soros elérésű memóriák alapeleme az él-vezérlésű vagy kétfázisú ***D MESTER-SZOLGA*** tároló. Ebből a tárolóból egy *2-1* multiplexer alkalmazásával olyan egységet kapunk, amelynek két adat-bemenete közül (***DS, DP)*** közül az ***S/P*** vezérlőjel szintjének egyike választ. A tároló ***PRESET***

bemeneteit konstans logikai alacsony szintre kötjük, a ***CLEAR*** bemeneteket ezzel szemben a kezdeti '***0***' állapot beállításra használni fogjuk. A *3.14. ábrán* látható egységet soros memóriák építőelemeként fogjuk felhasználni. Az ábra jobboldalán a komponensekből álló *séma*, a jobboldalon a kapott egység *szimbóluma* látható.



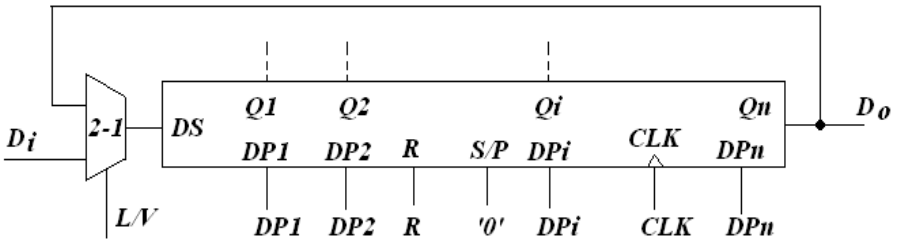

*3.14. ábra. Soros memória építő elemének sémája és szimbóluma*




*3.15. ábra. A soros memória sémája és szimbóluma*

### 3.3.1. 1-bites, párhuzamosan is betölthető soros memóriák

A *3.15. ábra* a 3.14. ábra szerinti egységekből felépített nyitott soros elérésű memória sémája. Az ***S/P*** vezérlő-bemenet állapotától függően az órajel-ciklusra vagy balról-jobbra léptetés, vagy párhuzamos betöltés történik. Az egység az ***R*** jellel alapállapotba hozható.

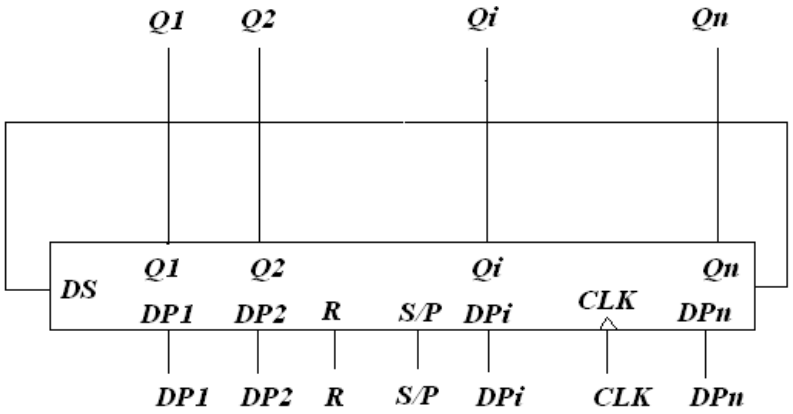



*3.16. ábra. Párhuzamosan betölthető, sorosan rátölthető soros memória.*

A *3.16. ábra* ennek az egységnek az a változata, amikor a memória tartalmát körben forgatva bármely beírt adat elérhető a kimeneten, de egy adat elvesztése árán új adatot is betölthetünk a ***Di*** bemenetről, az ***L/V*** vezérlő bemenet segítségével.

### 3.3.5. Gyűrűs számláló

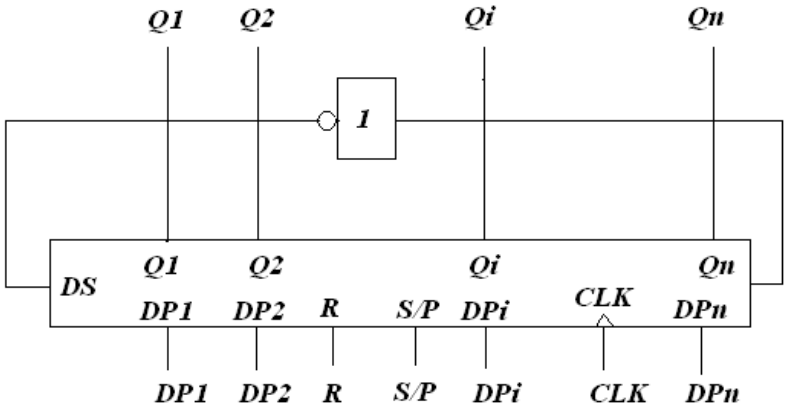
A visszacsatolási és soros beírási lehetőségétől megfosztott 1-*bit* szélességű soros memóriát olyan speciális szinkron sorrendi hálózatként üzemeltethetjük, amely egy párhuzamosan betöltött, egyes súlyú, azaz csak egy 1-est tartalmazó kezdeti bináris vektort forgat. Alakjáról *gyűrűs számlálónak* nevezzük. (*3.17. ábra*)




*3.17. ábra. Gyűrűs számláló*

### 3.3.6. Johnson számláló

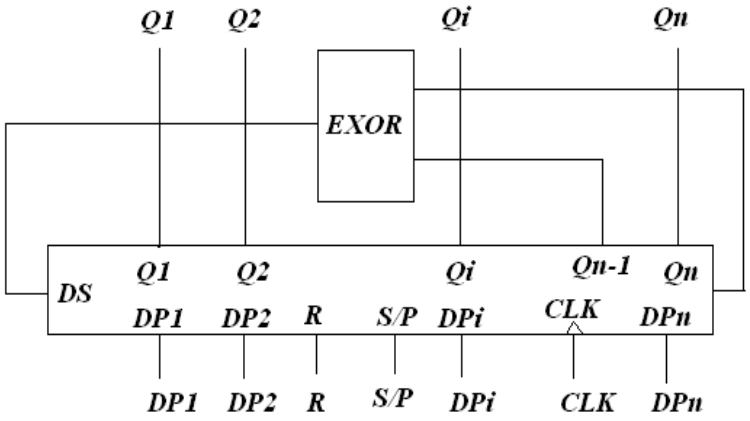
Ha a gyűrűs számláló visszacsatoló körébe egy invertert szúrunk be, sajátos ciklusú gyűrűs számlálót kapunk. Belátható, hogy az egyes súlyú kezdeti vektorral betöltött hálózat ***2n*** állapotszámú ciklust produkál, azaz a gyűrűs számláló ***n*** számú állapotát megduplázza. (*3.18. ábra*)




*3.18. ábra. Johnson vagy Mőbiusz-számláló*

### 3.3.7. Véletlenszám-generátor

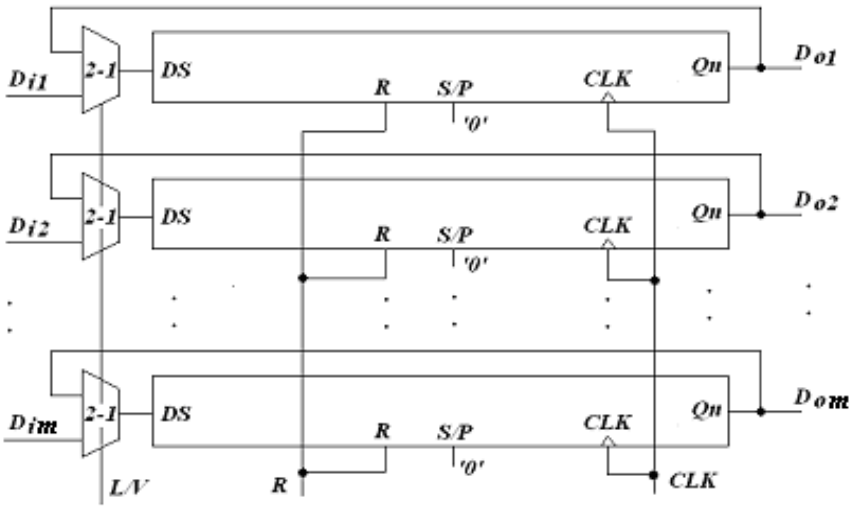
Ha a gyűrűs számláló visszacsatolásába **EXOR** kaput teszünk a *3.19. ábra* szerinti bemenetekkel vezérelve, egy párhuzamos betöltés utáni léptetés állapotai olyannak tűnnek, mintha ***n***-bites bináris számok véletlen-szerűen követnék egymást. Belátható, hogy valamennyi szám jelentkezik egy ciklusban, amely *(2)* $^{n}$ lépésből áll.




*3.19. ábra. Álvéletlen generátor*

### 3.3.5. Szószervezésű soros memóriák *(3.20.ábra)*

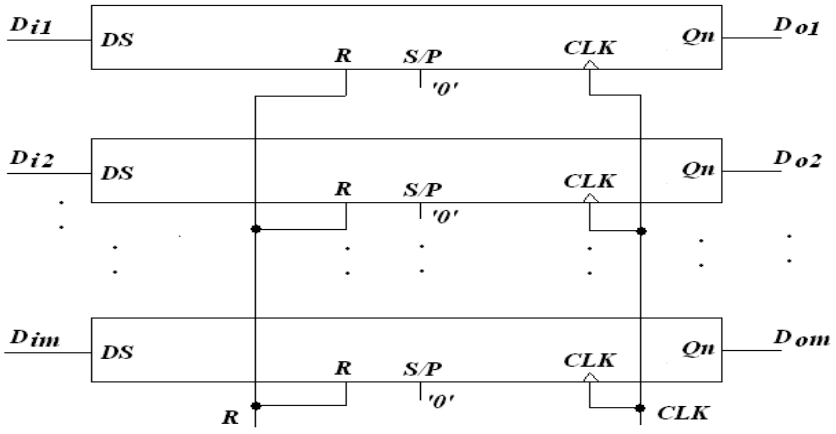
Ha a megismert építőelemből ***1***-nél nagyobb, pl. ***m*** szószélességű soros memóriát építünk, elhagyjuk a párhuzamos beírás lehetőségét, azaz az ***m*** számú gyűrűs számláló ***S/P*** bemeneteit soros üzemmódra állítjuk be. A memória ***L/V*** vezérlő-vezetékével beállíthatjuk, hogy a memóriában lévő adatokat forgatjuk, vagy új adatot szúrunk be a régiek közé.




*3.20. ábra. Egy m-bit szószélességű soros memória-egység*

### 3.3.6. FIFO memóriák

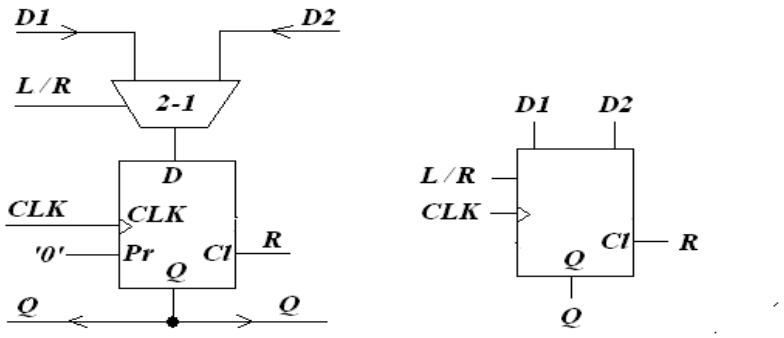
A ***FIFO*** olyan soros memória, amelyből az az adat olvasható ki először, amelyet elsőként töltöttünk be. (**F**irst **I**n **F**irst **O**ut).
Ha a fent bemutatott ***1-bit*** szélességű párhuzamosan betölthető soros memória egységekből elvesszük a párhuzamos betöltés lehetőségét, ***m***-számú ilyen egységből ***m***-bites szószélességű ***FIFO***-t csinálunk, ahogyan azt a *3.21. ábra* is mutatja.



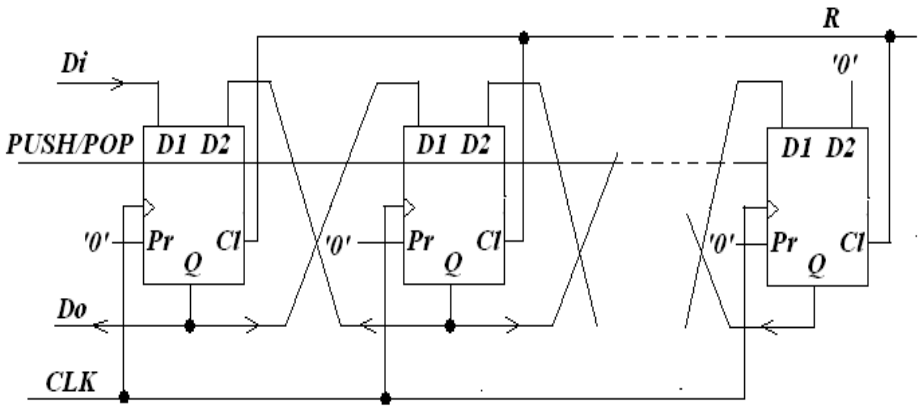

*3.21. ábra. Egy m-bit szószélességű FIFO memória*

### 3.3.7. LIFO memóriák

Nézzük a *3.22. ábra* szerinti alapelemet. Itt a ***D-MS*** flip-flop kimenetén az ***L/R*** vezérlőjeltől függően vagy a baloldali ***D1***, vagy a jobboldali ***D2*** bemenet szintje jelenik meg. Ez a ***LIFO*** tárolók alapeleme. A ***LIFO*** olyan soros memória, amelyből az az adat olvasható ki először, amelyet utoljára töltöttünk be. (**L**ast **I**n **F**irst **O**ut). Két vezérlő bemenete van. Betöltésre a ***BETÖLT (PUSH),*** kiolvasásra a ***POP (KIUGRAT)*** vezérlő-bemeneteket használjuk, természetesen egymás kizárásával. A ***LIFO*** egyetlen adat-csatlakozása tehát bemenet és kimenet szerepét is betölti.




*3.22. ábra. LIFO memóriaelem*




*3.23. ábra. LIFO-sor*

A ***LIFO*** elemekből alkotott ***1-bit*** szélességű ***LIFO*** memória a *3.23. ábrán* látható. Ebből könnyen alkothatunk ***m***-szószélességű ***LIFO*** memóriát.

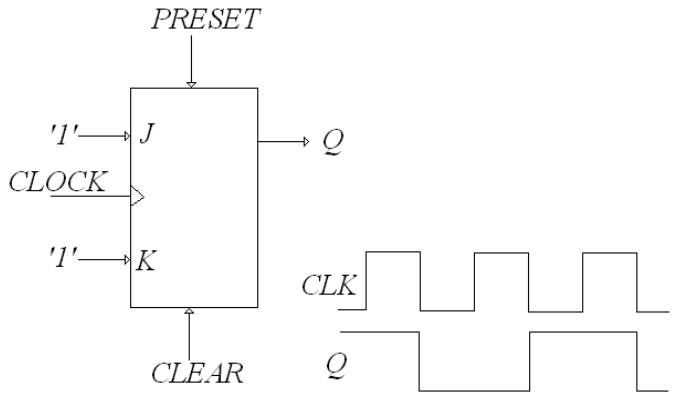
## 3.4. SZÁMLÁLÓK

A számlálók olyan összetett funkciójú digitális egységek, amelyeket az eredeti, hagyományos számlálási funkción túl digitális egységek időzítő-vezérlő áramköreiben való alkalmazása miatt tárgyalunk a regiszterek között. Mint látni fogjuk, az adat-folyamatokat vezérlő egységek időzítő áramköreiben a regiszterek alapvető feladatokat látnak el, a számlálók pedig rögzített funkciójú időzítőknek tekinthetők, amelyekből ugyanakkor rugalmasan tervezhetünk különféle időzítő egységeket.

### 3.4.1. A MESTER-SZOLGA J-K FLIP-FLOP, mint a számlálók alapeleme.

Az aszinkron ***PRESET*** és ***CLEAR*** bemenetekkel ellátott ***J-K MESTER-SZOLGA*** tárolót korábbi tanulmányainkból már jól ismerjük. Az élvezérelt flip-flop funkciót szinkron számlálókban használjuk ki, míg a kettes-osztó funkciót az un. *aszinkron számlálókban* (számláncok) alkalmazzuk. A *3.24. ábra* mutatja a kettes-osztót, idő-diagrammal. Beláthatjuk ugyanis, hogy amennyiben a ***MESTER*** tároló beírása az órajel felfutó, a ***SZOLGA*** beírása a lefutó élre történik, az órajel frekvenciája megfeleződik, azaz az így kapcsolt

***J-K*** flip-flop az órajel frekvenciát *2*-vel osztja. Megjegyezzük, hogy az aszinkron számláncokban az órajel logikai szerepet kap, ezért olyan tárolót kell választani, amelyben a ***CLEAR*** bemenet közvetlenül és azonnal hat a kimenetre, a órajel közre-működése nélkül.




*3.24. ábra. A J-K MESTER SZOLGA flip-flop kettes osztó funkciója*

A számlálókat két csoportra osztjuk, szinkron és aszinkron számlálókra. Mindkét típusú számlálót a *modulo* értékkel jellemzünk. A *modulo-m* (m-modulusú) számláló számlálási tartománya a természetes számok *m*-vel való osztása után kapott lehetséges maradékok tartományán halad át. **(*0, 1, 2, . . . m-1*)**.
Egy alapvető fogalmat kell még bevezetnünk. A ***CARRY-LOGIKA*** olyan kombinációs hálózat, amely a kimenetén ***1***-et ad, ha a számláló kimenetein az **(*m-1*)** kódja jelenik meg.

### 3.4.2. Szinkron számlálók modellje

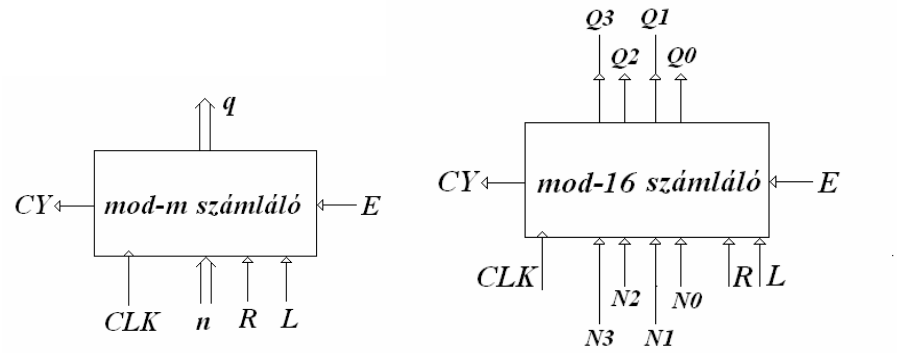
Szinkron számlálókat felépítő a ***J-K M-S*** *vagy* ***D-MS*** *flip-flopok* órajele azonos. A visszacsatoló hálózat a számláló állapot-átmeneti gráfja alapján tervezhető meg azokkal a módszerekkel, amelyeket a szinkron hálózatok tervezésének tárgyalásakor elsajátítottunk. Hangsúlyozandó, hogy számlálók esetén az állapotkód adott. A *betölthető*, *engedélyezhető*, *törölhető* szinkron *modulo-m* számláló vezérlő-jelei és funkciói a következők:

- Az ***R (RESET)*** magas szintje a rákövetkező órajel lefutására nullázza a számlálót. Az ***R*** jelnek a többi vezérlőjellel összehasonlítva abszolút prioritása van.

- Az ***L (LOAD)*** jel hatására, amennyiben nincs ***R***, a számláló tartalma a következő órajel legfutó élére az n bemenetre kapcsolt bit-vektor lesz.

- Az ***E (ENABLE)*** jel, amennyiben az ***R*** és az ***L*** bemenet is logikai ***0*** szinten van, engedélyezi, hogy a következő órajel lefutó élére a számláló tartalma 1-vel növekedjék (inkrementálás).

A *3.25. ábrán* bemutatjuk a fenti szinkron számláló szimbólumát, és az egyik leggyakoribb *modulo-16*-os változatot. Megjegyezzük, hogy ennél több funkciót is megvalósítanak, például a felfelé történő számlálás mellett lefelé való számlálást is. (***UP-DOWN COUNTERS***).
Ha a bemutatott *modulo-16*-os számlálót ***15***-ig (***1 1 1 1***) felszámoltattuk, akkor a ***CY*** (***CARRY***) kimeneten magas szint jelenik meg. A következő órajel lefutó élére a számláló ismét ***0***-ra áll (***0 0 0 0***), és a ***CY*** kimenet is alacsony szintre kerül. Ez ***CY*** arra alkalmas, hogy egy magasabb helyi-értékre helyezett számláló ***E*** bemenetét magas szintre állítsa, így lehetővé téve nagyobb modulusú számlálók kialakítását.

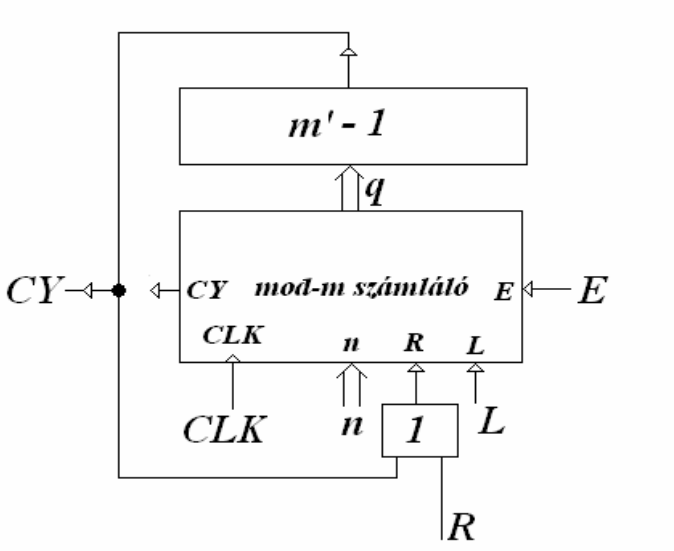



*3.25. ábra. Betölthető, engedélyezhető, törölhető szinkron modulo-m számláló általános szimbóluma és a gyakori 4-bites mod.-16-os alaptípus.*

### 3.4.3. Adott modulusú számláló átalakítása más modulusú számlálóvá

Ha egy *m-modulusú* szinkron számlálót át kívánunk alakítani ***m' < m*** modulusúvá, akkor új ***CY*** hálózatot kell kialakítani. Az új ***CY*** detektálja az új

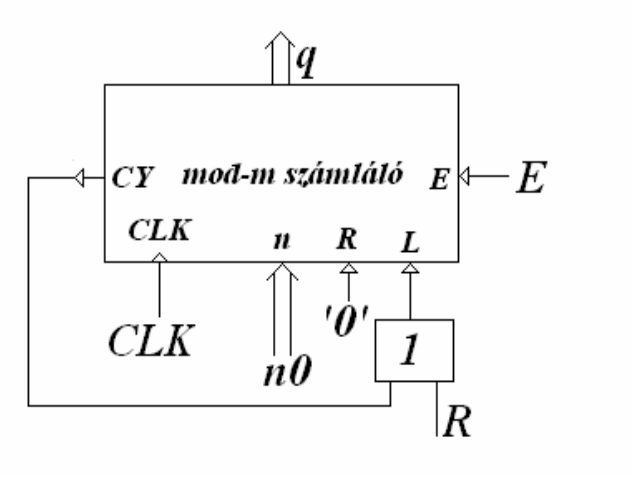
***m' -1*** értéket, és a soron következő órajel a számlálót a ***RESET*** bemenet segítségével törli (*3.26. ábra*)




*3.26. ábra. M- modulusú szinkron számláló átalakítása m' < m modulusú szinkron számlálóvá*

### 3.4.4. Számláló nullától különböző kezdő-értékének beállítása
*(3.27. ábra)*

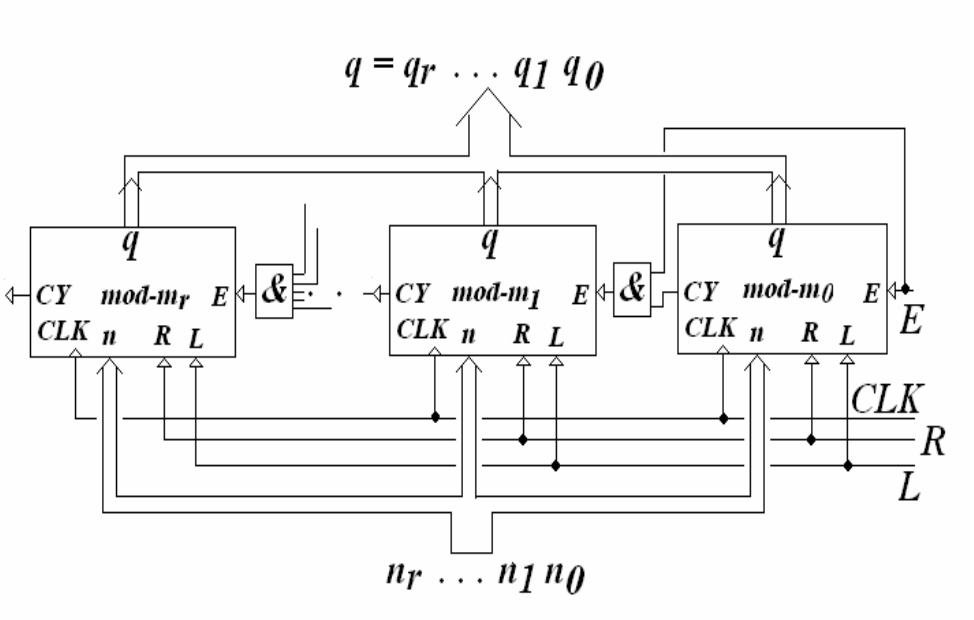
A nullától különböző, de a modulusnál kisebb kezdőértéket az eredeti számláló ***L*** jelének felemelésével, a ***CY =1*** feltétellel írjuk a számlálóba. Az új ***R*** bemenet ugyancsak az ***L*** bemenetre hat. Az eredeti ***R*** bemenetet nem használjuk, azt konstans logikai ***0*** értékre kötjük.




*3.27. ábra. 0-tól különböző kezdeti érték beállítása*

### 3.4.5. Szinkron számlálók kaszkádosítása (*3.28. ábra*)

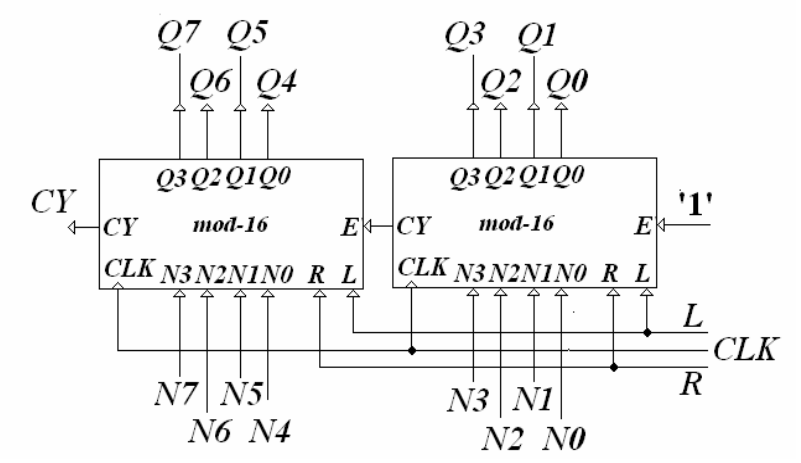
A kaszkádosítás célja, hogy a szinkron számláló modulusát több, kisebb modulusú, leggyakrabban a sorozatban gyártott számlálókból, mint komponensekből állítjuk össze. Minden modul csak akkor lép tovább a soron következő órajelre, ha a tőle jobbra állók mindegyike ***m-1*** értéket mutat. A ***CY*** kimenetek tehát ***ÉS*** kapcsolatban állnak egymással. Mind a kompozit számláló tartalmát, mind a betöltendő számot ilyenkor célszerű olyan számláncolattal jelölni, ahol a lánc minden „szeme” az adott komponensre jellemző, az adott modulusnál kisebb, vagy azzal egyenlő szám, ($q_r \ldots q_1\ q_0$), ($n_r \ldots n_1\ n_0$).




*3.28. ábra. Kompozit szinkron számláló, kaszkádosítással.*

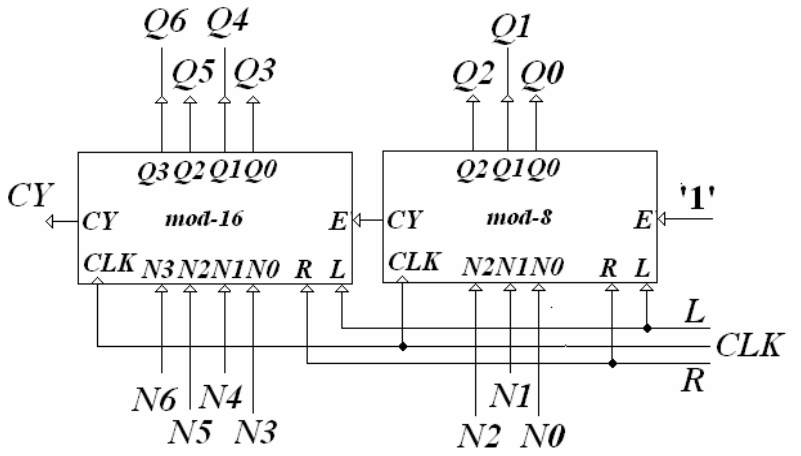
### 3.4.6. Példa : Modulo-16 szinkron számlálók kaszkádosítása modulo-256 számlálóvá.

Vizsgáljuk azokat a számláló struktúrákat, amelyekben **ENABLE** bemenetekre **CY** logikák csatlakoznak. Például, 4-bites, *modulo-16*-os számlálókból kialakított struktúra (*3.29. ábra*).




*3.29. ábra. Mod-256-os számláló kialakítása mod-16 modulokból.*

**3.4.7. Példa : Modulo-8 és modulo-16-os szinkron számlálók kaszkádosítása modulo- 128 számlálóvá** *(3.30. ábra)*




*3.30. ábra. Mod-128-as szinkron számláló mod-8-as és mod-16-os kaszkáddal.*

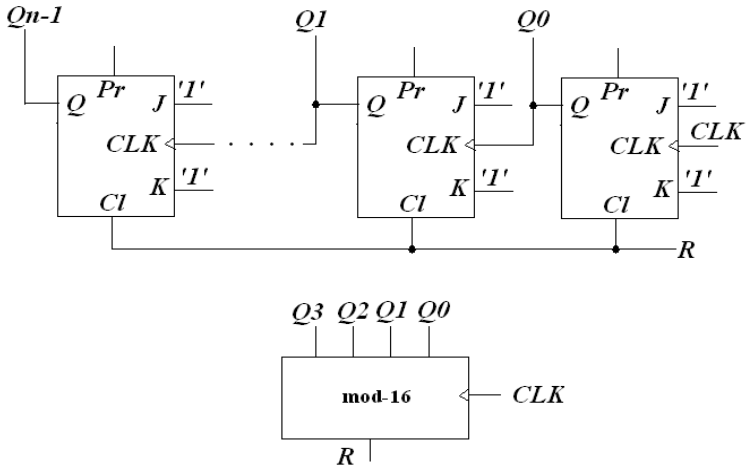
A számlálási sorrend a következő :

| Q6 | Q5 | Q4 | Q3 | CY | Q2 | Q1 | Q0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| . | . | . | . | . | . | . | . |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| . | . | . | . | . | . | . | . |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| | | | | | | | |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| . | . | . | . | . | . | . | . |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Vegyük észre, hogy az eredő modulus a komponens modulusok szorzata!

### 3.4.8. Aszinkron számlálók

Az aszinkron számlálók alapeleme a *kettes osztó. Modulo-$2^n$ ($m = 2^n$)* számlálót kapunk, ha *n*-számú kettes osztót a *3.31. ábrán* látható módon kapcsolunk össze :



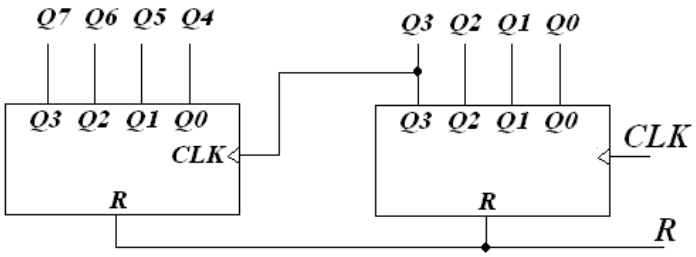

*3.31. ábra. Aszinkron számlánc kettes osztókból, és a szimbólum n = 4 esetén*

A számlálási sorrend (***Qn-1 . . .Q0***) ***R*** vezérlés (***RESET***) után:

| | |
|---|---|
| **000. .00** | **(0)** |
| **000. .01** | **(1)** |
| **000. .10** | **(2)** |
| **. . . . . .** | |
| **. . . . . .** | |
| **100. .00** | **($2^{(n-1)}$)** |
| **. . . . . .** | |
| **111. .11** | **($2^n$ -1)** |
| **000. .00** | **(0)** |

### 3.4.9. Aszinkron számlálók kaszkádosítása

Nem okoz időzítési (hazárd-működésből, adódó) gondokat, ha a kaszkád komponensek modulusai kettő hatványai. A *3.32. ábra* egy aszinkron *mod-255* számlálót mutat, *mod-16*-os modulokból.




*3.32. ábra. Mod-256 számláló 4-bites aszinkron számláncok kaszkádosításával*

## 3.5. FUNKCIÓS EGYSÉGEK

A funkciós egységek egy vagy két binárisan ábrázolt adattal valamilyen műveletet végeznek el. A funkcionális egységek kimerítő tárgyalása

meghaladná ennek a kurzusnak a kereteit, csupán felvillantjuk a legfontosabb alapelveket. A funkciós egységek opearandusai és eredményei *bináris kódban* ábrázolt számok. Kettő hatványaival súlyozott bináris kódolás elvét, az egész számok bináris alakjának felírását, a bináris számok decimálisokká való alakításának szabályait már ismerjük. Meg kell azonban ismernünk a negatív számok (egészek és törtek) ábrázolásának azt a módját is, amely könnyűvé teszi az előjeles bináris számokkal való aritmetikai műveleteket. Ezért ebben a bevezető részben megismerjük a *komplemens* kódok fogalmát és a velük való számolás fő szabályait.
Minden hatvány-kitevős súlyokkal ábrázolt pozitív számból, függetlenül a számrendszer *r* alapjától képezhető egy *inverz* szám. Az inverz szám minden egyes számjegye helyébe azt a számjegyet írjuk, amelyet az eredeti számjegyhez hozzáadva a rendszer maximális számjegyét kapjuk. Például 3-jegyű decimális számok esetén: Ha $N = 642$, akkor $N_I = 357$, hiszen a maximális értékű számjegy *9*. Belátható, hogy minden pozitív számra igaz, hogy

$$N + N_I = r^n - 1,$$

Ahol *n* számjegyek száma, azaz az előbbi példa esetében $n = 3$.
Az *3* jegyű pozitív decimális szám és inverzének összege *999*, azaz $10^3 - 1$. Ebből következik, hogy az *N* pozitív szám negatív párja a következőképpen írható fel:

$$-N = -r^n + N_I + 1$$

Tehát a -642 felírható, mint ( *-1000 + 357 +1* ).
Ha ezt a kettes hatványai szerint súlyozott bináris számokra alkalmazzuk, akkor az inverz előállítása azt jelenti, hogy minden egyes számjegyet, mint logikai értéket (*0* vagy *1*) negálni kell, majd ebből megkapjuk a szám negatív párját, ha a legkisebb helyi-értéken *1*-t hozzáadunk az inverzhez. Példa : legyen egy bináris tört a *0. 1 0 1*, ahol a bináris pontot a legnagyobb helyi-értékű pozíció után képzeljük. Ilyenkor a szám decimális tört alakban *1*(1/2) + 0*(1/4) + 1*(1/8) = + 5/8*
*Ennek a számnak az inverze : 1.0 1 0, komplemense pedig*

```
  1.0 1 0
+ 0.0 0 1
---------
  1. 0 1 1
```

Ez tehát a *-5/8* tört bináris, komplemens kódja.

A legfontosabb tulajdonság, hogy a *+5/8* és a *-5/8* összege bináris összeadással bináris zérust ad, azaz az összevonás műveleti eredményeit az algebra szabályainak megfelelően kapjuk meg.

```
  0. 1  0 1
 +1. 0 1 1
 ----------
1 0. 0 0 0
```

A ($2^1$) súlyú pozícióban keletkezett túlcsordulást nem vesszük figyelembe. Ha a negatív számokat abszolút értékük komplemensével jelenítjük meg, akkor a következő aritmetikai szabályok adódnak :

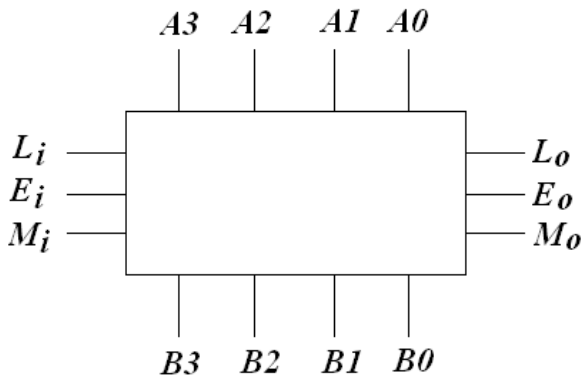
1. *A számok összeadása az ábrázolásnak megfelelő eredményt szolgáltat*
2. *Egy szám kivonása azonos eredményt ad a komplemensének hozzáadásával.*

Ebből az következik, hogy az aritmetikai egységünk a *kivonást is összeadással végezheti el.*

### 3.5.1. Komparátorok

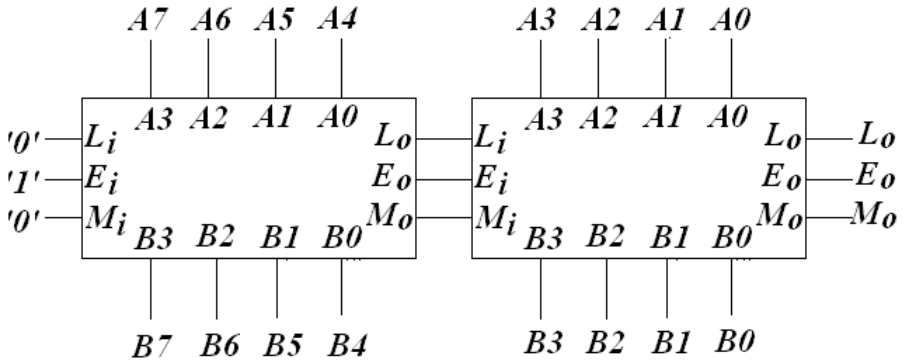
A komparátorok binárisan ábrázolt adatok összehasonlítását végzik el. A következőkben csak pozitív bináris számok összehasonlításával foglalkozunk, de megjegyezzük, hogy léteznek a komplemens kódra kiterjesztett komparátor változatok is. Az összehasonlításnak általában háromféle eredménye lehet : az egyik adat nagyobb a másiknál, (***Mo)***, azonos értékű a másikkal, (***Eo***) , vagy kisebb a másiknál. (***Lo***) A logikai áramkörcsaládok legtöbbjénél a komparátorok mindhárom lehetőséget egy-egy kimenettel reprezentálják, de a rugalmas felhasználáshoz az kell, hogy az adott méretű adatok összehasonlítását végző egységek összekapcsolhatók legyenek az operandus-méretek növelésének céljából. A bővítést szolgálja az összehasonlítás elve is, nevezetesen, hogy a komparátor az ***MSB***-től az ***LSB*** felé haladva mindaddig nem dönt, amíg valamelyik bit-pozícióban eltérést nem talál. Az eltérés döntést eredményez, és feleslegessé teszi a további alacsonyabb helyi értékű bitek összehasonlítását.
Ha egy adott méretű, például egy 4-bites komparátort három bemenettel is ellátunk, (***Mi, Ei, Li***) és az elé, magasabb helyi értékű 4-bitre elhelyezett komparátor azonos nevű kimeneteivel vezéreljük, valamint biztosítjuk, hogy a már feljebb meghozott ***Mo = 1*** vagy ***Lo = 1*** döntés egyenesen kijusson a

megfelelő kimenetre, ezzel szemben ha csak az ***E=1***, akkor az adott komparátor egység vizsgálja a neki kiosztott bit-párokat, akkor korlátlanul bővíthető (kaszkádosítható) komparátor egységeket kapunk. Egy ilyen komparátor egységet látunk a *3.33. ábrán.*




*3.33. ábra. 4-bites komparátor egység*

A *3.34. ábra* mutatja, hogyan kapcsoljuk össze a két *4-bites egységet 8-bitessé*. Ábránkon a baloldali modul balszélső, így bemeneteire a megfelelő logikai konstans értékeket kell kapcsolni.

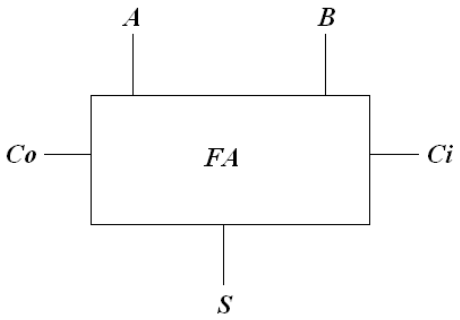



*3.34. ábra. 8-bites komparátor két 4-bites egységből*

### 3.5.2. Összeadók

#### 3.5.2.1 A teljes összeadó (1-bites összeadó)

A legtöbb aritmetikai logikai egység alapeleme az *1-bites* összeadó. Az egység igazság-táblázata könnyen megalkotható, ha elképzeljük a bináris összeadás műveletét egy adott helyi-értéken, ahová áthozhatunk értéket az megelőző, kisebb helyi-értékről, (***Ci***),hozzáadjuk az adott helyi-értéken megjelenő operandus bitek (***Ai, Bi***) összegét (***S***), és képezzük a nagyobb helyi-értékre való átvitel értékét, (***Co***). A teljes összeadó szimbóluma és igazság-táblája látható a *3.35. ábrán*.




| A | B | Ci | S | Co |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

*3.35. ábra A teljes összeadó és igazság-táblája*

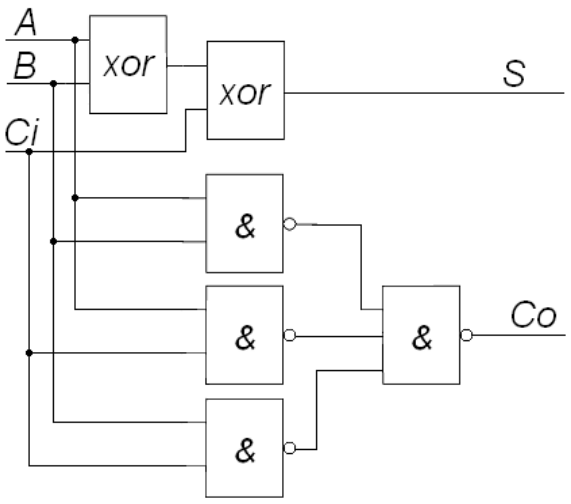
A tervezés K-táblán való elvégzése utáni logikai kifejezések:

$$S = (A \oplus B) \oplus Ci$$

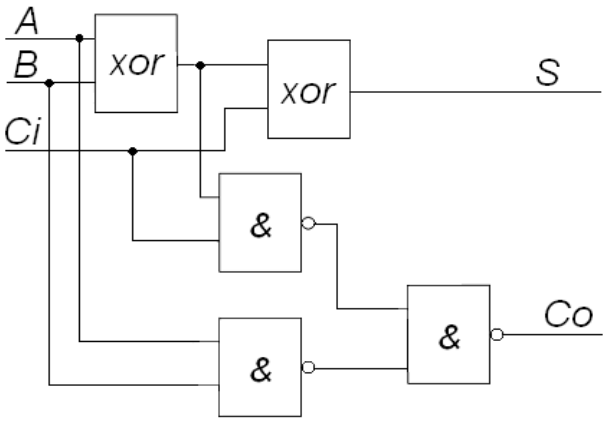
$$Co = A\,B + B\,Ci + A\,Ci \qquad (1)$$

$$Co = (A \oplus B) + A\,B \qquad (2)$$

A ’$\oplus$’ szimbólum a „kizáró-vagy” (***EXOR***) jele.
Láthatjuk, hogy az átvitel (***Co***) kifejezése kétféleképpen is felírható. A második kihasználja az összegben már szereplő **EXOR** eredményt.
Mindkét megoldás kapu-szintű struktúrája látható a *3.36. és 3.37 ábrákon*. Az első megoldás hátránya, hogy több kapuból áll, mint a második, ugyanakkor előnye, hogy mind az összeg, mind az átvitel két kapunyi úton halad át, szemben a második megoldással, ahol az átvitel három kapun keresztül alakul ki.



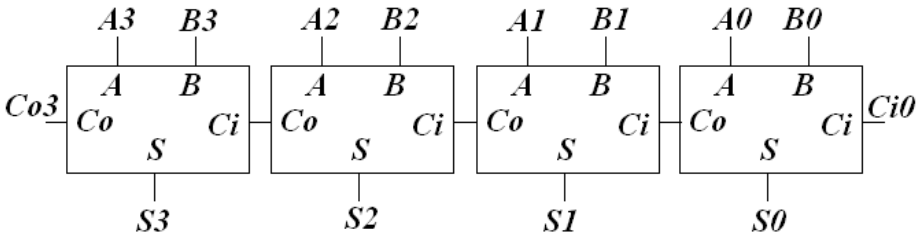

*3.36. ábra A teljes összeadó gyorsabb változata*




*3.37. ábra. A teljes összeadó egyszerűbb változat.*

### 3.5.2.2. Soros átvitel-képzésű bit-vektor összeadó

A teljes összeadók kaszkádosításával soros átvitelképzésű bit-vektor összeadót kapunk. (*3.38. ábra*). A soros átvitelképzés óriási hátránya, hogy a legnagyobb helyi-értéken az eredmény az összes fokozaton által késleltetve jelenik meg. Minél szélesebb az összeadó, annál nagyobb lesz a műveleti idő. A műveleti idő csökkentését szolgálják a párhuzamos átvitel-képzésű, illetve a vegyes, párhuzamos-soros átvitel-képzésű összeadók.



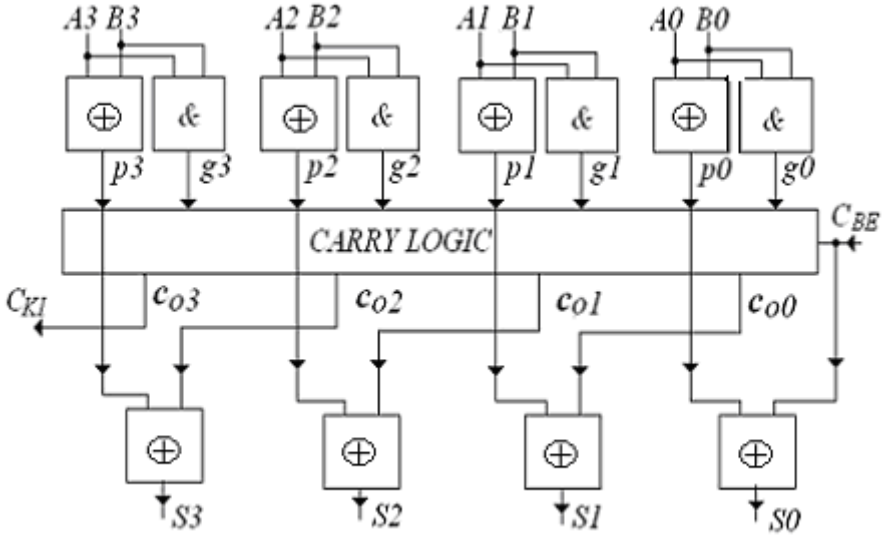

*3.38. ábra. Soros átvitelképzésű, 4-bites összeadó egység*

### 3.5.2.3. Párhuzamos átvitelképzésű bit-vektor összeadó

Minden helyi-értéken, így a *k*. helyi-értéken is, képezzük a következő logikai jeleket:

$$p_k = A_k \oplus B_k$$
$$g_k = A_k B_k$$
$$S_k = (A_k \oplus B_k) \oplus Ci_k = p_k \oplus Co_{(k-1)}$$
$$Co_k = (A_k \oplus B_k)\, Ci_k + A_k B_k = p_k\, Co_{(k-1)} + g_k$$

Elvégezve a legkisebb helyiértéktől kezdve a ***Cok*** értékek kiszámítását, és minden indexnél behelyettesítve a kisebb helyiértékek bemeneteit, olyan logikai kifejezéseket kapunk, amelynek alapján a *3.39. ábra* sémája rajzolható fel.

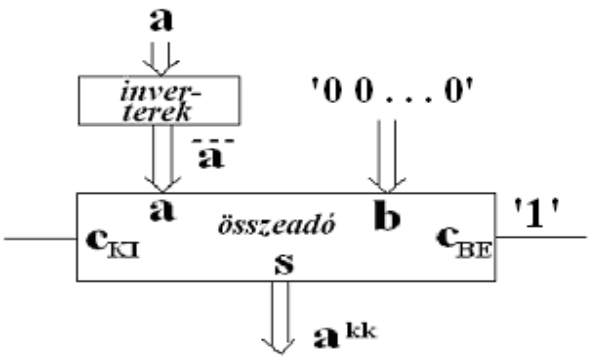



*3.39. ábra. 4-bites, párhuzamos átvitelképzésű összeadó*

### 3.5.3. Kivonás

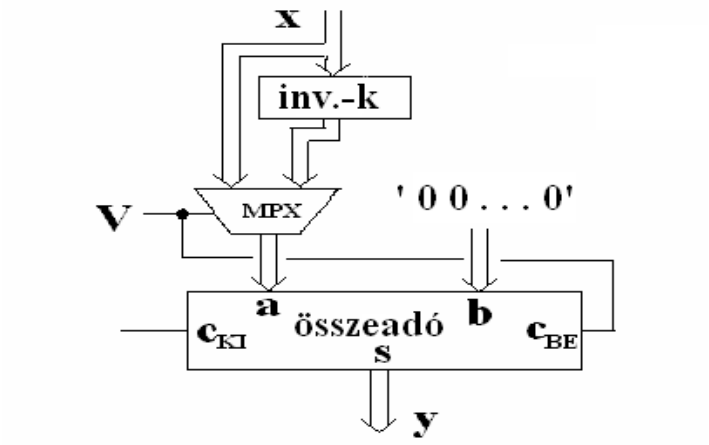
A kivonás műveletét a bináris számok kódolásának megfelelő megválasztásával összeadásra lehet visszavezetni. Azt a bináris kódot, amelynek alkalmazásakor az összeadás fenti módokon való elvégzése a kivonás műveletét is kiszolgálja, a már tárgyalt komplemens kód. (Ld. A *3.5.*

bevezetője). A *3.40. ábra* szerinti egység az a kettes-komplemens kódban ábrázolt szám kettes-komplemensét állítja elő. Azaz, minden kivonandót rajta átengedve, összeadással hajthatjuk végre a kivonást.



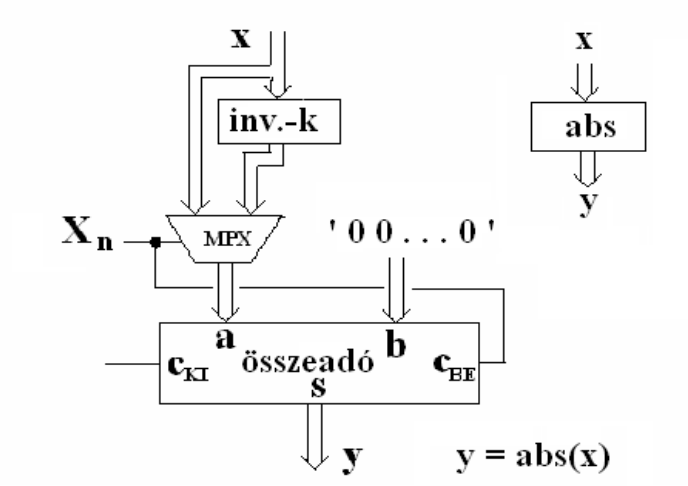

*3.40. ábra. Kettes-komplemens-képző egység.*

*A 3.41. ábra* szerinti egység ***V = 1*** esetén az ***x*** kettes komplemensét állítja elő, míg ***V = 0*** esetén ***y = x.***



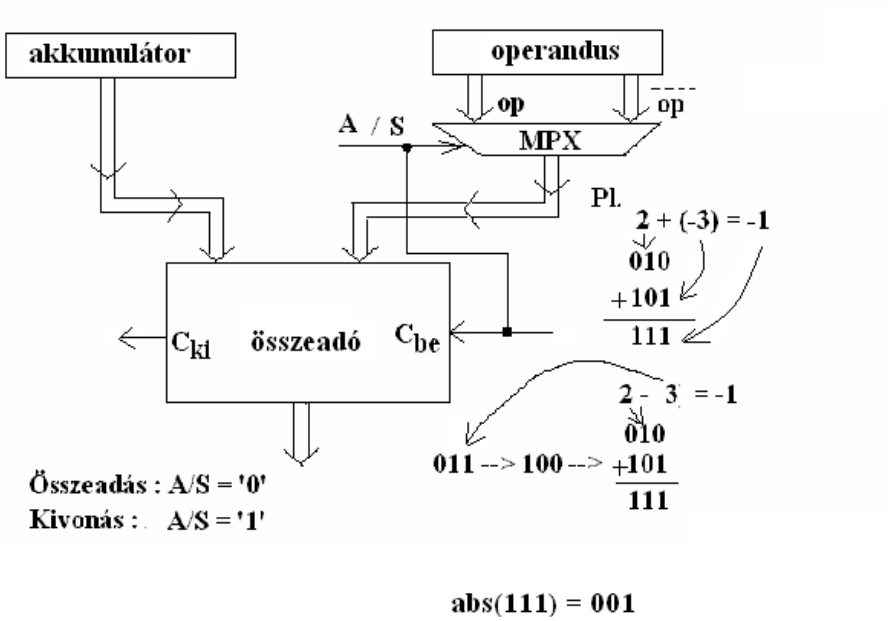

*3.41. ábra. Vezérelhető kettes-komplemens képző egység***.**

A *3.42. ábrán* abszolút-érték képző egységet láthatunk. Bármely kettes komlemens kódban érkező szám abszolút értéke kerül a kimenetre. Az ***Xn, az MSB,*** egyben a szám előjele, ha '***1***', akkor a szám kettes komplemense lesz az abszolút érték.




*3.42. ábra. Abszolút érték-képző egység*

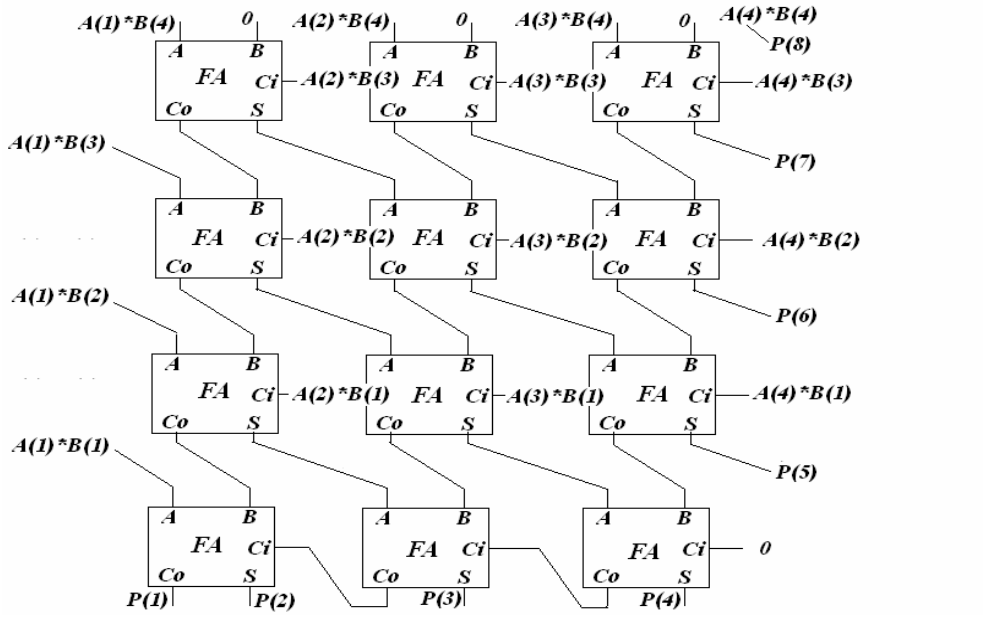
A *3.43. ábrán* azt mutatjuk be, hogyan vezetik vissza a mikroprocesszorokban a kivonás műveletét összeadásra. Ha összeadunk, ***A/S = '0'***, így az összeadó jobboldali bemenetére maga az operandus kerül az azt tároló regiszterből. Ezzel szemben, ha ***A/S = '1'***, azaz kivonni kell, a multiplexer az operandus bitenkénti negáltját kapcsolja az összeadóra, sőt, a ***Cbe*** bemenet is '***1***', azaz az operandus kettes-komplemense kerül az akkumulátor tartalmával való összeadásra.




*3.43. ábra. Kivonás mikroprocesszorok aritmetikai egységében*

### 3.5.4. Szorzók

A szorzókat, hasonlóan az összeadókhoz az jellemzi, hogy a bináris vektorok közötti bitenkénti szorzás-léptetés-összeadás műveletsorát végzik el. A szorzó megoldások igen nagy száma miatt ezeket részletezni nem fogjuk, inkább egyetlen megoldást mutatunk be, egy array-szorzót. (*3.44. ábra*) A tömb speciálisan összekapcsolt teljes-összeadókból áll. Az ***A(i)*B(j)*** jelölések ***ÉS*** kapukat szimbolizálnak. Az ábrán két ***4-bites*** bináris tört-vektor szorzását mutatjuk be, az indexek a bináris ponttól jobbfelé, azaz a kisebb helyi-értékek felé növekednek. Így a szorzat legnagyobb helyi-értékén ***P(1),*** a legkisebb helyi-értékén ***P(8)*** szorzat-bit szerepel. Hangsúlyozzuk, hogy az ábra szerinti tömb pozitív számok szorzására alkalmas, de hasonló tömb szorzó megoldásokat fejlesztettek ki kettes-komplemens kódban ábrázolt előjeles számok kettes-komplemens kódú eredményt szolgáltató szorzására is.

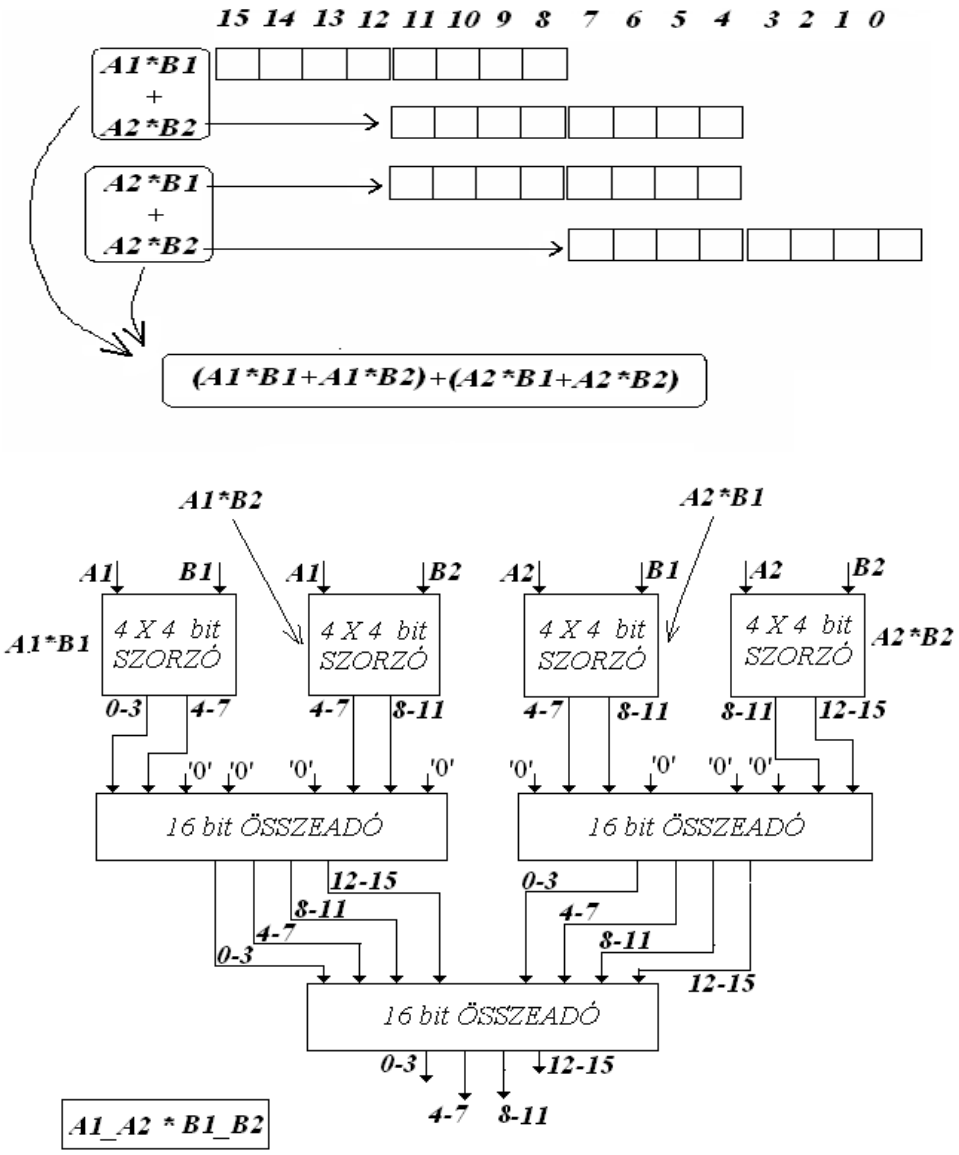



*3.44. ábra. 4 x 4 –es tömb-szorzó*

Megjegyezzük, hogy a tömb-szorzó késleltetése a bitvektorok dimenzió-számával közel lineárisan nő.
A *3.45. ábra* azt mutatja, hogyan lehet viszonylag kisméretű, például *4-bites* tömbökből nagyobb méretű, pl. *8 x 8* bites szorzót kialakítani. Ha az ***A1_A2*** két négybites vektor *lánca* az egyik operandus, a ***B1_B2*** , amely ugyancsak két négybites vektor lánca a másik operandus, akkor a szorzatokat *4*x*4* bites szorzókkal, részletekben is elő lehet állítani. Ezután minden részlet-szorzatot a maga helyi-értékének megfelelő helyen kell *16-bites* összeadókra vezetni.
Figyelem! A *3.45. ábra* szerinti séma minden egyes összeköttetése egy 4-bites vektort jelöl.

## 3.5. VEZÉRLŐ EGYSÉGEK

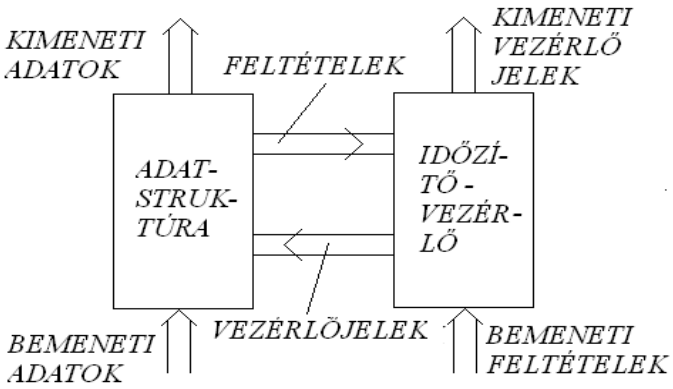
Ebben a fejezetben bemutatjuk a digitális berendezések tervezésének azt a módszerét, amelynek első lépése egy felbontás, azaz dekompozíció. Ez a gondolat elkülöníti az adat-utakat az azok kijelölését és időbeli mozgatását meghatározó időzítő-vezérlő egységtől.




*3.45. ábra. 8* x *8 bites szorzó, 4* x *4-s komponensekből (A szorzat MSB : 0, LSB: 15).*

### 3.6.1. Az adatstruktúrára (DATA-PATH) és vezérlőegységre (TIMING AND CONTROL) való felbontás (dekompzíció)

Az adat-struktúrára és időzítő-vezérlő egységre való felbontás szükségessé teszi a két egység közötti kapcsolódási pontok pontos meghatározását. Az adat-struktúra lényegében regiszterekből, multiplexerekből és funkciós egységekből áll, *bemeneti adatokat* fogad, és *kimeneti adatokat* szolgáltat. Ezek az egységek egymással összekapcsolódva különféle adat-pályákat tesznek lehetővé. Az, hogy ezek közül adott helyzetben mely pályák valósulnak meg, illetve e pályáknak mely szakaszai aktivizálódnak egy adott pillanatban, ezt az időzítő-vezérlő határozza meg. Az időzítő-vezérlő címzi meg a multiplexerek és forrásait illetve kimeneteit, állítja be funkciós egységek művelet-vezérlő kódjait, és felemeli, illetve lebocsátja a regiszterek beíró jeleit. Ezeket nevezzük *vezérlő-jeleknek*. Az időzítő vezérlő működését azonban az adat-struktúrából rávezetett jelek, *feltételek* befolyásolják. Ugyanakkor a környezet *bemeneti-feltételekkel* befolyásolja az időzítő-vezérlő működését, és maga az időzítő vezérlő is szolgáltat *kimeneti vezérlő jeleket* a környezet számára (*3.46. ábra*). A következőkben az időzítő-vezérlő egységet röviden vezérlő egységnek fogjuk nevezni.




*3.46. ábra. Digitális egység (rendszer) felbontása.*

### 3.6.2. Számláló-típusú vezérlők

A számláló típusú vezérlő három fő részből áll. Alapegysége egy *törölhető, engedélyezhető, betölthető* szinkron számláló. Ehhez három multiplexer

csatlakozik, amelyek egyike az engedélyező bemenetre rákapcsol egy kiválasztott feltételt, a másik a betöltést kiváltó bemenetre csatlakozva kiválaszt egy, az ugrást vezérlő feltételt, a harmadik a betöltendő szám bemeneteire csatlakozva kiválaszt több lehetséges betöltendő szám közül egyet. A multiplexerek kiválasztó bemeneteinek szintjeit a számláló kimenetei határozzák meg. Nézzük a *3.47. ábrát*. A számláló vezérlő-bemenetei között fennálló elsőbbségi viszonyokat figyelembe véve a működés a következőképpen írható le:

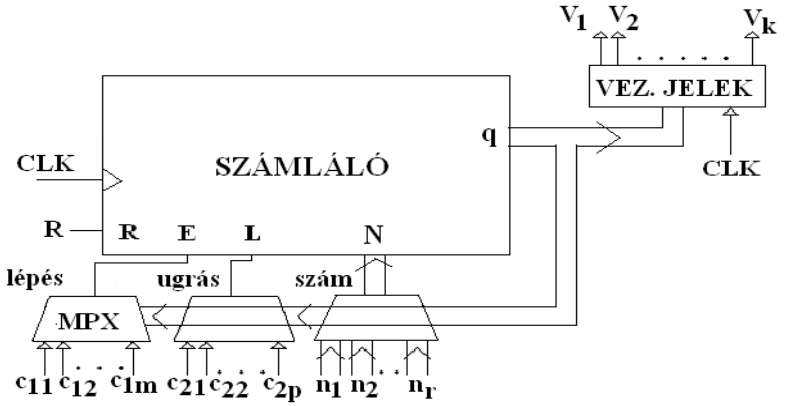
*- Ha az **R** bemenetet felemeljük, a számláló tartalma a fennálló helyzettől függetlenül nullázódik. Ezzel szemben, ha*

*- az **R** bemenet szintje alacsony, akkor két eset van:*

- *Egyik, ha a számláló kimenete által megcímzett, a betöltést vezérlő bemenetre kapcsolódó feltétel magas szintű, akkor az általa egyidejűleg megcímzett szám betöltődik a számlálóba, ezzel szemben a másik az,,*

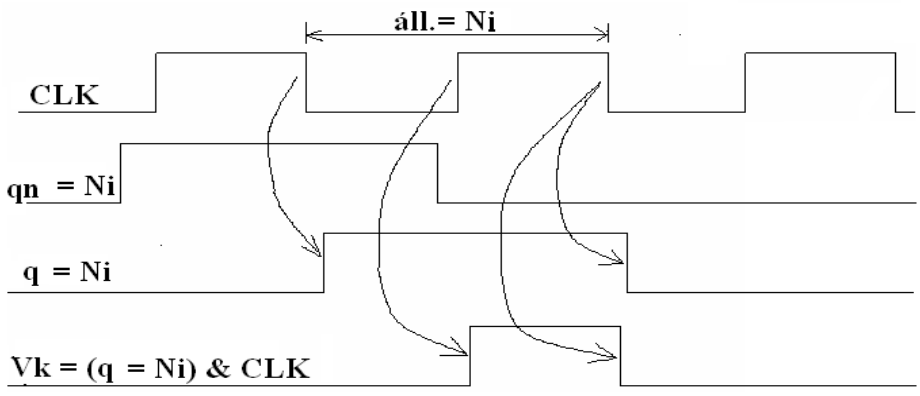
- *ha a betöltésre kapcsolódó feltétel szintje alacsony. Ekkor ismét két eset van:*

    - *Egyik, ha a kiválasztott, az engedélyező bemenetre kapcsolódó feltétel magas. Ilyenkor a számláló tartalma inkrementálódik,*
    - *ezzel szemben , ha az engedélyező bemenetre kapcsolódó feltétel szintje alacsony, akkor a számláló értéke nem változik.*




*3.47. ábra. Számláló típusú vezérlő*

A harmadik fő egység a környezetnek, és az adat-struktúrának szóló vezérlő-jeleket generálja. A vezérlő-jelek generálásának alapproblémája a hazárd-mentesség. Ez azt jelenti, hogy a vezérlőjelek csakis az tervező által meghatározott idő-intervallumokban lehetnek aktívak, azokon kívül stabilan passzív logikai szinten kell azokat tartani. Ha valamennyi vezérlő jel aktivitását magával az órajellel szinkronizáljuk, akkor a *3.48.* ábra szerinti idő-diagram igazolja a hazárdmentességet.

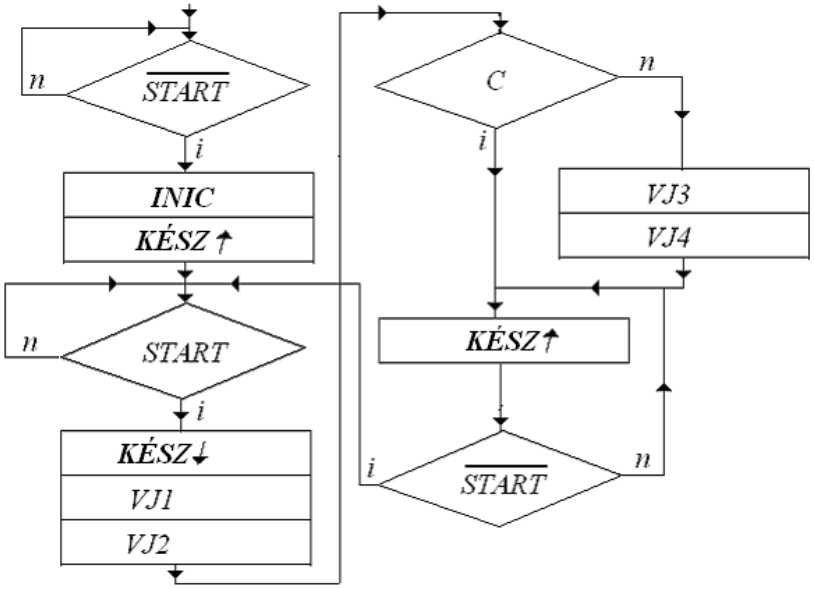



*3.48. ábra. A számláló típusú vezérlő időzítése.* ***qn*** *: a következő számláló állás,* ***q*** *az aktuális számláló állás*

Tegyük fel, hogy a ***q = Ni*** számláló-álláskor, és csakis akkor szeretnénk a ***Vk*** vezérlő-jel aktivitását kiváltani, az órajellel szinkronban. Belátható, hogy a vezérlőjel bekövetkezése után következő órajel már az időzítési feltétel biztos elmúlása után jelenik meg.

### 3.6.3. Példa számláló típusú vezérlő-egység tervezésére

A megvalósítandó időzítő-vezérlő egységet a *3.49. ábra* szerinti folyamatábra definiálja. Eszerint az adat-struktúra számára szolgáltatni kell egy ***INIC*** nevű jelet, valamint a ***VJ1, VJ2, VJ3, VJ4*** vezérlő-jeleket, a környezetből fogadni kell egy ***START*** jelet, tegyük fel továbbá, hogy az adat-struktúrából ered a ***C*** nevű jel, és szolgáltatni kell a környezet számára a ***KÉSZ*** jelet.




*3.49. ábra. Egy vezérlő-egység működésének megadása folyamatábrával*

A folyamatábra szimbólumainak jelentése :

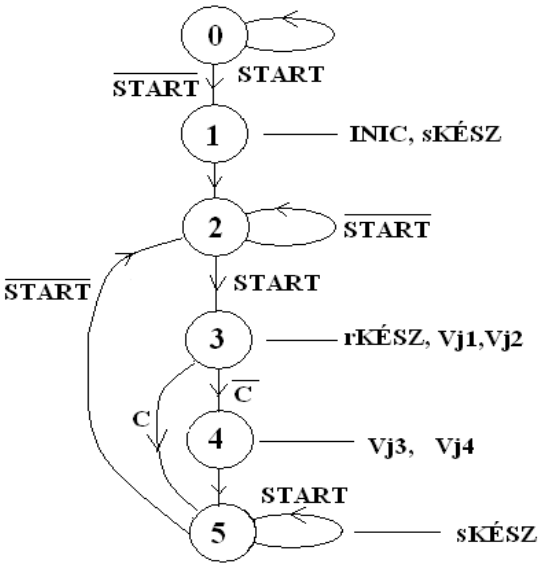
1. rombusz, logikai feltétellel *: Ha igaz, az 'i' kimeneten távozunk*
2. Négyszög, jelnév-bejegyzéssel *: órajellel szinkronizált pozitív impulzus a megnevezett jelen.*
3. Négyszög jelnév-bejegyzéssel és felfelé v lefelé mutató nyíllal *: Az órajel felfutó élére szinkronizált felfutás vagy lefutás a megnevezett jelen*

Eszerint a vezérlő-egység, amennyiben a ***START*** alacsony, produkál egy ***INIC*** impulzust, és a ***KÉSZ*** felemelésével jelzi a készenlétet. Ezután vár a ***START*** felfutó élére. Ha az megjelenik, visszaejti a ***KÉSZ*** jelet, és a ***VJ1, VJ2*** vezérlőjeleken párhuzamosan impulzust produkál. A következő akciók a ***C*** bemenettől függenek. Amennyiben ***C*** alacsony szintű, akkor a ***VJ3, VJ4*** impulzusai megjelennek, ha magas, akkor azok kimaradnak. Ezután megint fel kell emelni a ***KÉSZ*** jelet, és a ***START*** szintje szerint visszatérni.

### 3.6.3.1 A vezérlési folyamat ütemezése, azaz egy vezérlési szekvencia leírás elkészítése

A folyamatábra akcióit a számláló-állásokkal jelölt állapot-gráfban rendezzük. A számláló-állásokhoz rendelt *akciókat* a gráf jobboldalán soroljuk fel. Az akció rész lehet üres. Ilyen például az első ütem, amikor a ***START*** értékétől

függően várakozunk, vagy egy ütemmel tovább lépünk. Figyeljük meg, hogy itt a jel-emeléseket és ejtéseket '**s**' ill '**r**' előtagokkal jelöljük, érzékeltetve ezzel, hogy az ilyen típusú jeleket ***S-R*** bistabilokkal állítjuk elő. Így minden ilyen jelhez két impulzus-típusú jelet rendelünk. Ezzel azt is elérjük, hogy az ütemezett vezérlési szekvencia már csak impulzus-típusú jelekre hivatkozik. Lássuk tehát az ütemezett vezérlési-szekvencia leírást, a *3.50. ábrán*.




*3.50. ábra. A számláló állapotainak gráfja akciókkal*

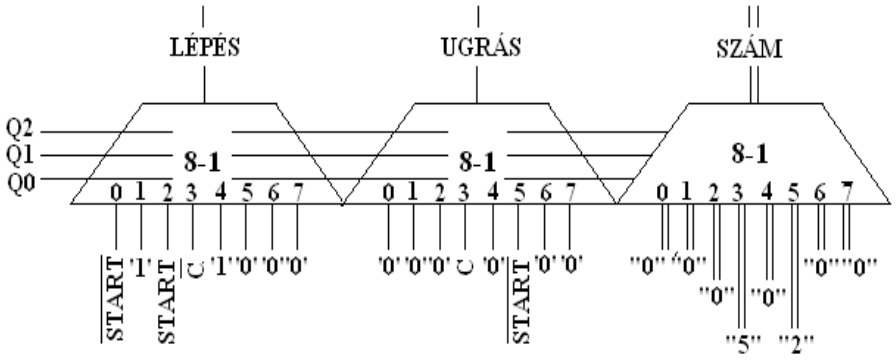
### 3.6.3.2 A multiplexerek megtervezése a vezérlési szekvencia-leírásból

A realizáláshoz *mod-8* számlálót fogunk használni, tehát *3-bites* számlálót.
A „***LÉPÉS***" multiplexer megtervezésekor azokat az ütemeket gyűjtjük össze, amelyekben inkrementálás található. Az ütemek az inkrementáláshoz tartozó feltételeket címzik. Feltétel nélküli inkrementálási ütemhez nyilvánvalóan konstans logikai ***1*** tartozik. Hasonlóan gyűjtjük össze az „***UGRÁS***" multiplexer címeit és bemeneteit is. Ugyanezekhez a címekhez rendeljük a „***SZÁM***" multiplexer bemeneteit is, amelyekhez az ugrásokhoz beírt ütemszámokat rendeljük.

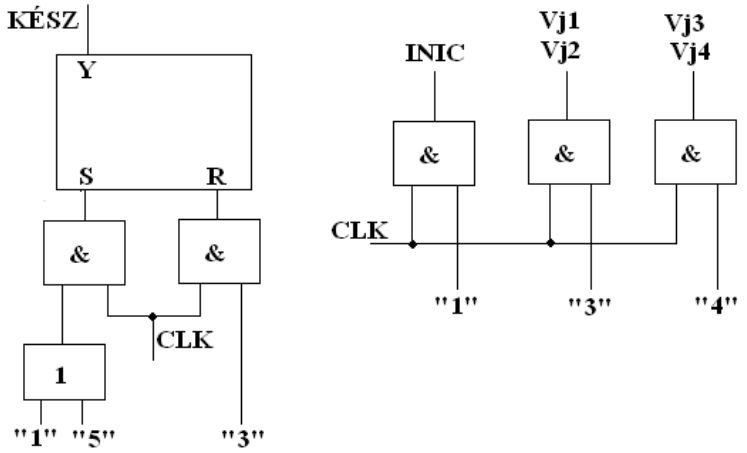
### 3.6.3.3 A vezérlőjel-generátorok tervezése

Az ***INIC*** valamint a ***VJ1 . . .VJ4*** vezérlőjelek előállítása ***NEM-VAGY*** kapukkal történhet, a számlálóról levett és dekódolt ütemszámok felhasználásával. Mivel a kapuk bemenetére a negáltakat kell kapcsolni, a dekódolt ütemszámok negáltját tüntettük fel a kapuk bemenetein.

A ***KÉSZ*** jel előállítása bistabil tárolón keresztül történik. Látható, hogy az alkalmazott ***NÉS-NÉS*** bistabilon is az órajellel szinkronban történik meg a változás.
A multiplexereket a *3.51. ábra*, a vezérlőjel-generátorokat a *3.52. ábra* mutatja be.



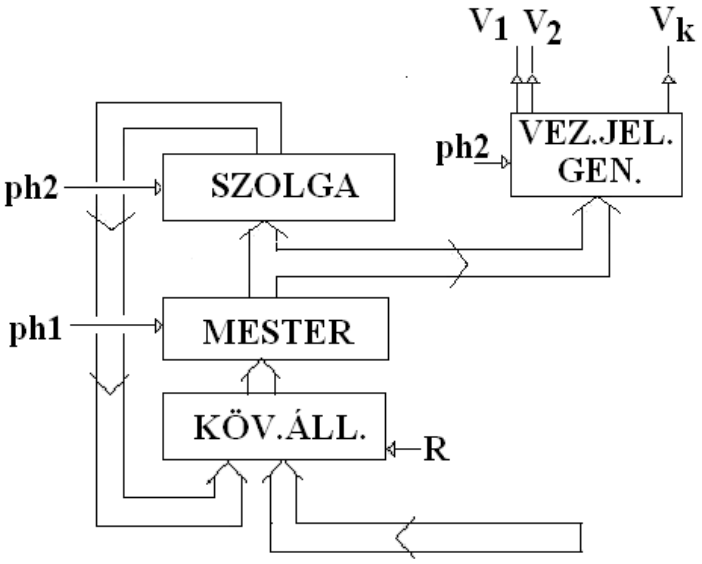

*3.51. ábra. A számlálót vezérlő multiplexerek. A (Q0 Q1 Q2) vektor elemei a számláló kimenetei*




*3.52. ábra A vezérlő-jel generátorok. Az állapotok dekódolt formában, egyetlen bittel szerepelnek az ábrán.*

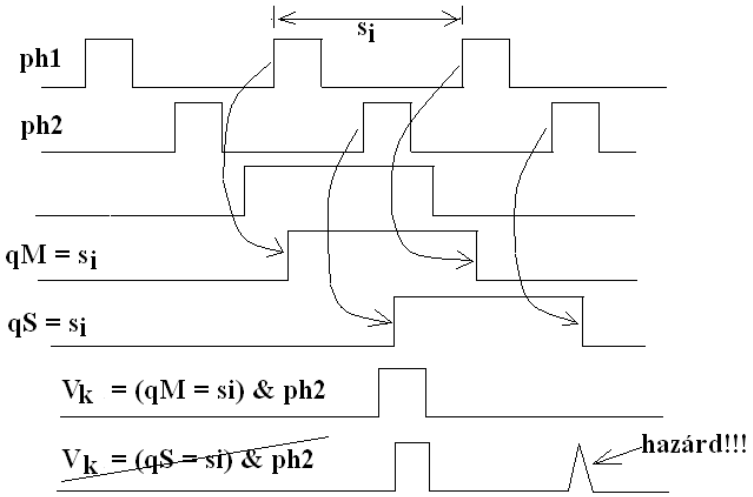
### 3.6.4. FSM típusú vezérlők

Az FSM ( Finite State Machine) típusú vezérlő, nevét arról kapta, hogy az időzítő lényegében véges állapotszámú szinkron sorrendi hálózat.




*3.53. ábra. Kétfázisú órajellel működő FSM típusú vezérlő*

Ábránkon egy kétfázisú órajellel működő egység látható (*353. ábra*). Felismerjük a klasszikus, kétfázisú szinkron-sorrendi hálózat struktúrát, ha a flip-flopok ***MESTER*** fokozatait is és a ***SZOLGA*** fokozatokat is összefogjuk egy-egy él-vezérelt regiszterben. Az idő-diagramon (*3.54. ábra*) azt mutatjuk be, hogyan kell kivitelezni egy ***ph2*** fázissal szinkronizált vezérlőjel generátorát. Látható, hogy csak a ***MESTER*** fokozatról levett időzítő-információ eredményez hazárdmentes megoldást. Figyeljünk fel arra, hogy az állapotokhoz tartozó idő-intervallumokat most a ***ph1*** felfutásai között definiáljuk. Megjegyezzük, hogy a klasszikus ***MSI***-szintű logikai áramkörök világában inkább a számláló típusú vezérlőt érdemes alkalmazni, hiszen a számláló készen van, és funkciói kihasználhatók, míg az ***LSI-VLSI*** világban leginkább a kétfázisú FSM-típusú vezérlő tervezése az ajánlott.

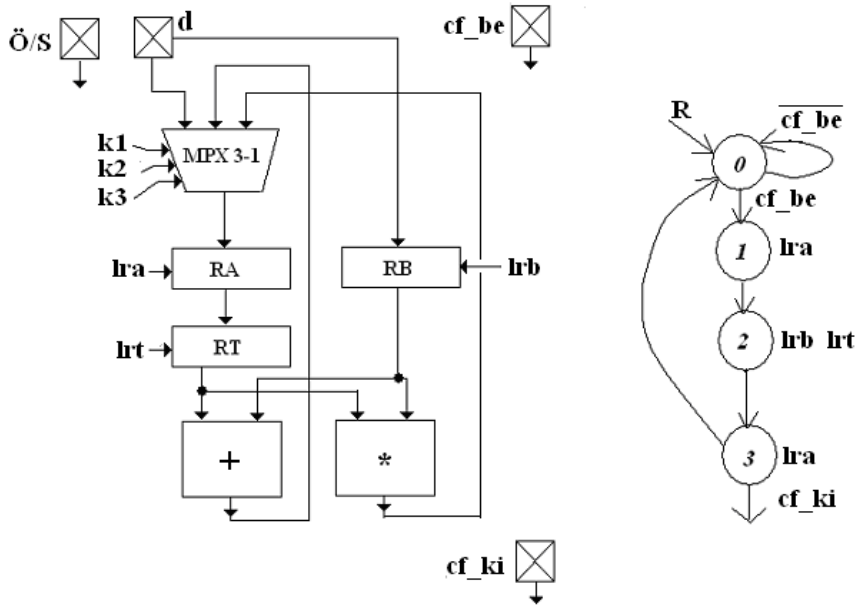



*3.54. ábra. Kétfázisú órajellel hajtott FSM típusú időzítő idő-diagramja*

### 3.6.5. Önálló makro-cella tervezése FSM típusú vezérlővel

Az önálló makro-cellák ***LSI-VLSI*** áramkörök építőelemei lehetnek. Sajátosságuk, hogy egy speciális feladatra tartalmazzák mind az adatstruktúrát, mind az időzítő-vezérlőt, és beilleszthetők egy nagyobb vezérlési folyamatba. Fontos, hogy több ilyen önálló egység osztozkodhat az adat-struktúra elemein, ha nem akadályozza ezt erőforrás igény ütközés. Valósítsuk meg a következő önálló (autonóm) makro-cellát:
Az egység várakozzon a vezérlési folyamatot átadó másik önálló egységtől vezérelt ***cf_be*** jel felfutására. Ha ez megérkezett, egymás után két adatot ugyanarról a ***d*** adat-sín bemenetről írjon be két regiszterbe (***RA, RB***), majd az ***Ö/S*** bemenet szintjétől függően adja össze, vagy szorozza össze azokat, és az eredményt irja vissza az ***RA*** regiszterbe. Ha ezt elvégezte, küldje tovább a vezérlést egy harmadik önálló egységnek a ***cf_ki*** jel felemelésével, ő maga pedig várakozzék újabb indításra, de a ***cf_ki*** jelet ejtse le alap-állapotába.
Ami az adatstruktúrát illeti, világos, hogy legalább két regiszterre és két funkciós egységre, egy összeadóra és egy szorzóra van szükségünk. Beláthatjuk azonban, hogy az egyik operandusnak az eredménnyel való felülírása megköveteli egy átmeneti regiszter (***RT***) bevezetését is. Nyilvánvaló ugyanis, hogy a funkciós egységek kombinációs egységek, és bemenetükön nem írhatjuk felül azt az adatot, amely az eredmény stabilan tartásához és tárolásához szükséges. Ezért az ***RA*** tartalmát átmentjük egy átmeneti tárolóba,

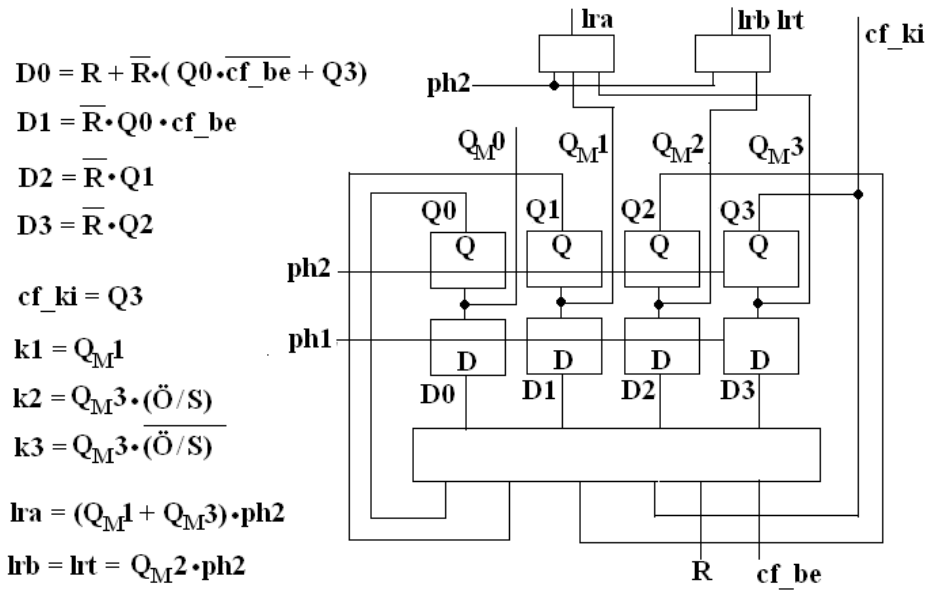
ezt kapcsoljuk a funkciós egységekre, és így az ***RA*** befogadhatja az eredményt. Azt is könnyen kitalálhatjuk, hogy az ***RA*** regiszterbe végül is három különböző forrásból kell tudni adatot betölteni. Ezek a ***d*** adatsín, az összeadó kimenete és a szorzó kimenete.
Ezek után felvázolható a *3.55. ábra* bal oldalán látható adat-struktúra. Mindhárom regiszterből „kilógnak" a betöltő-jelek, (***lra, lrb, lrt***) a ***3-1*** multiplexerből pedig a három kiválasztó bemenet, (***k1, k2, k3***). Természetesen a művelet-kiválasztó jel is szerepet kap a vezérlőben, csakúgy mint az FSM-t elindító ***cf_be*** külső vezérlő bemenet. Természetesen szolgáltatni kell a ***cf_ki*** vezérlés-továbbadó jelet is.




*3.55. ábra. FSM típusú időzítő-vezérlővel működő autonóm egység adat-struktúrája és időzítőjének állapot-gráfja*

Az ábra jobb oldalán az FSM állapot-gráffal megadott ütemezést láthatjuk. A ***0*** sorszámú. állapot a kezdeti állapot, ahová az ***R*** jel kényszeríti az időzítőt. Ebben az állapotban várakozni kell a vezérlési folyamat átadására, azaz a ***cf_be*** jel felemelkedésére. Innen kezdve sorrendben követik egymást az ***1 , 2*** és ***3*** sorszámú állapotok. Az ***1***-es állapotban az ***lra***-n beíró-impulzust kell kiváltani. Ezalatt a ***d*** sínre kell kapcsolódnia az RA regiszter bemeneteinek, azaz a multiplexer k1 kiválasztó jelét kell magasra emelni. A ***2***-es állapotban

egyszerre töltjük a második adatot a ***d***-ről az ***RB*** regiszterbe, illetve a ***RA***-ból az első adatot az átmeneti regiszterbe. Az ***lrb*** és ***lrt*** betöltő-impulzusok tehát egyszerre jelentkeznek. Innen kezdve a két funkciós egység kimenetén megjelenik az összeg, illetve a szorzat. A ***3***-as állapotban ismételten az ***lra***-ra adunk impulzust, de ezalatt a ***k2*** vagy a ***k3*** kiválasztó bemenetek egyikének kell magasan lennie.
A megvalósított időzítő-vezérlő a *3.56. ábrán* látható, sematikusan. A ***D*** bemenetekre csatlakozó fekete-doboz tartalmát a baloldali logikai kifejezések mutatják.




*3.56. ábra. A feladatban megfogalmazott időzítő vezérlő megvalósítása. Az időzítőt 1-es súlyú állapotkóddal realizáltuk.*

### 3.6.6. Vezérlés mikroprogramozással

A mikroprogramozott vezérlő-egység elve, hogy mind a vezérlő-akciókat, mind az elágazásra vonatkozó információkat egy címezhető memóriában, (lehet írható/olvasható, vagy csak olvasható is), az ütemezésnek megfelelő sorrendben helyezzük el, mikro-utasítások formájában. A *3.57. ábrán* látható egység egy adott mikroutasítás kiolvasása után vagy a következő mikroutasítás kiolvasásába kezd, vagy elugrik egy megadott címre. Azt, hogy ezek közül melyik eset valósul meg, feltételek teljesülésétől függ. Nyilvánvaló, hogy ez a két lehetőség kettős elágazás megvalósítására is

lehetőséget nyújt, hiszen a következő címen elhelyezhetünk egy feltétel nélküli ugrást.

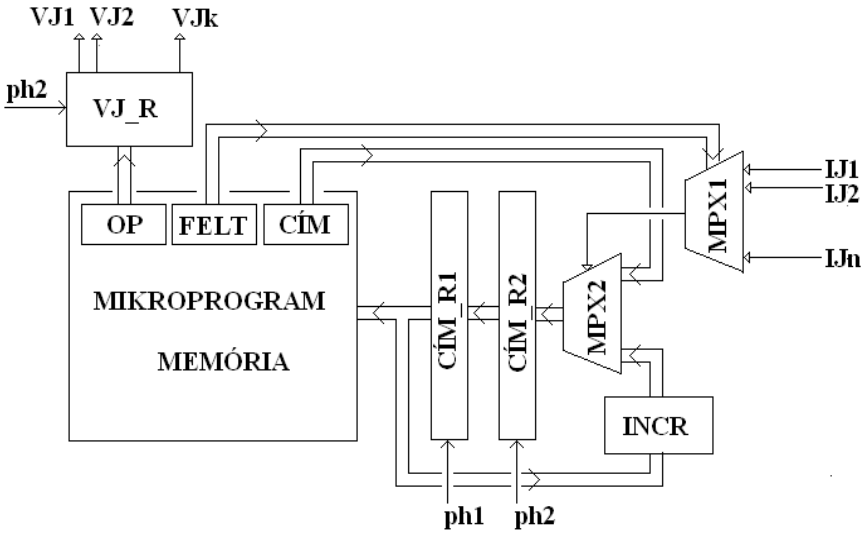
### 3.6.6.1. A mikroprogram utasítás-szó felépítése

Az utasítás-szó három részből áll. Az első az akciókat kiváltó, un. operációs rész, (***OP***) a második a bemeneti jelek (feltételek) címzésére alkalmas un. feltétel-rész (***FELT***), míg a harmadik pedig az a memória cím, ahová akkor ugrunk, ha a ***FELT*** rész által kiválasztott jel magas szintű.

Kritikus az ***OP*** rész méretének megválasztása. Ha az ***OP*** részben minden egyes vezérlőjelhez tartozik egy bit, akkor ezt horizontális mikrokódnak nevezik. Előnye, hogy egyetlen mikroutasítással akár valammennyi vezérlő-jel aktivizálhatóegyetlen mikroutasítással,, hátrány viszont, hogy a memória horizontális mérete maximális. Ha az ***OP*** rész csak a vezérlőjelek egyikét címzi meg, akkor a mikroutasítás szó a lehető legrövidebb, de egy mikroutasítással csak egyetlen egy vezérlőjel aktivizálható. Ez megnöveli a mikroprogram végrehajtásának idejét. Az ilyen mikroprogramot nevezik vertikális mikrokódnak.

Nyilvánvaló, hogy a két szélsőséges megoldás között számos közbenső változat létezik. Például párhuzamosan címezhető vezérlőjel csoportokon belül egyedi jeleket címezünk.

Nyilvánvaló tehát, hogy a teljes mikroprogram méretét nemcsak a folyamat ütemeinek a száma, hanem a mikroutasítás mérete is befolyásolja.




*3.57. ábra. A mikroprogramozott időzítő-vezérlő egység sémája*

### 3.6.6.2 A mikroprogramozott vezérlő egység időbeli működése

Az *3.57. ábra* alapján könnyen követhető a működés. A ***ph1*** órajel felfutására beíródik egy memória-cím a ***CÍM_R1*** regiszterbe, és ennek megfelelően a megcímzett utasítás szó részei megjelennek a memória kimenetein. Ez a mikroutasítás elővétele. Ennek hatására a ***VJ_R*** vezérlőjel regiszter bemenetén megjelenik az akciókat vezérlő ***OP*** rész, az ***MPX2*** kimenetén megjelenik a **FELT** rész által megcímzett feltétel ill bemenőjel logikai szintje, és az ***MPX2*** két bemenetén megjelenik egyrészt a potenciális ugrás címe, másrészt az ***INCR*** hálózaton keresztül az aktuális cím inkrementáltja. Az ***INCR*** tulajdonképpen egy összeadó, amelynek egyik operandusa mindig '***1***'.
A következő esemény a ***ph2*** órajel felfutása. Erre a kiolvasott ***OP*** rész megjelenik a vezérlő-jeleken, és az ***MPX1*** szintjétől függően vagy az inkrementált cím, vagy az ugrás címe töltődik be a ***CÍM_R2*** regiszterbe. Ezzel a következő ***ph1*** felfutásra megindul a következő mikroutasítás elővétele.

## 3.7. BEVEZETÉS A MIKROPROCESSZOROS RENDSZEREK TERVEZÉSÉBE

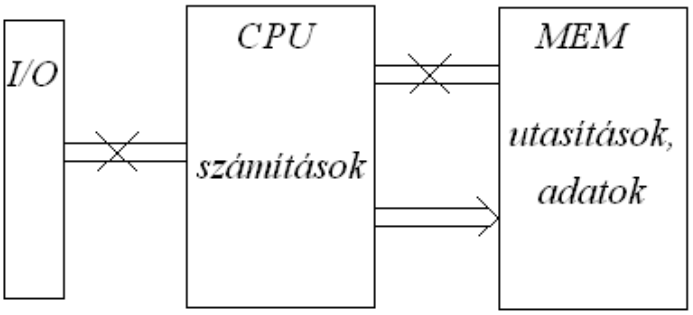
A fejezet célja, hogy az olvasó mikroprocesszoros rendszerek legalapvetőbb fogalmait megismerje, kellő alapismereteket szerezve az ilyen rendszerek tervezési módszereinek elsajátításához.

### 3.7.1. A Neumann-architektúra

A mikroprocesszorok technikája az elmúlt század hetvenes éveiben alakult ki. Néhány kivételtől eltekintve mind a klasszikus, mind a mai mikroprocesszorok a Neumann-féle architektúra hagyományos vonásait őrzik. A *mikroprocesszoros rendszer* fő részei a következők:

- A ***CPU***, azaz a központi egység, amely többnyire maga a mikroprocesszor,
- A ***MEMÓRIA***, amelyben a programot (utasításokat ) és az adatok nagy részét is tároljuk valamint az
- ***I/O***, azaz kommunikációs egységek, amelyek a környezettel való adatcserét biztosítják.

A rendszernek ezt a felbontását a *3.58. ábra* mutatja. A CPU a memóriával két sín-rendszeren keresztül kommunikál. Az egyikkel címzi a memóriát (ADDRESS-BUS, CÍM-SÍN), a másikon keresztül pedig utasítások érkeznek a CPU-ba, illetve adatok jönnek-mennek. Az I/O egységekkel való kapcsolat mindig kétirányú.
A CPU a rendszer legaktívabb eleme. Ha bekapcsoljuk, azonnal megcímzi a memória legelső celláját, és feltételezi, hogy amit onnan kiolvas, az egy utasítás. Innen kezdve, tehát az első utasítás végrehajtásától kezdve a processzor vagy a következő utasítást vesz elő, vagy elugrik arra a címre, amelyet az éppen végrehajtott utasítás diktál, és ott folytatja a program végrehajtását. Ezt a rendet persze néhány sajátos és rendkívüli esemény felboríthatja. Ezekkel később részletesen foglalkozunk. A Neumann architektúra sajátossága tehát az *utasítások sorrendben történő (szekvenciális)* végrehajtása.




*3.58. ábra. Mikroprocesszoros rendszer fő elemei*

### 3.7.2. A CPU címzési módjai

A *CPU* a memóriában tárolt utasításokat és adatokat címeik segítségével éri el. A CÍM (ADDRESS) a címbuszon, (ADDRESS-BUS) megjelenő bináris vektor. Ha azt akarjuk, hogy ugyanazt a memóriát több *CPU* is használhassa, a címbuszt a *CPU*-nak „lebegtetnie", azaz szabadon hagynia is tudni kell. (Harmadik állapot) A címbusz tehát *egyirányú, háromállapotú sín. Feltételezzük, hogy minden utasítás egyetlen operandusra hivatkozik, mert a másik operandus helye adott. Ez általában az AKKUMULÁTOR regiszter. Így minden utasításnak egy memória címet kell meghatároznia. Ez lehet:*

- implicit
- közvetlen (*immediate*)
- direkt (*direct*)
- indirekt (*indirect*)
- indexelt

Az egyes címzési módok jelentése a következő:

*Implicit címzés :* az utasítás kódja utal az operandus helyére (pl. egy kitüntetett regiszter)

*Idézetes címzés :* az utasítás tartalmazza magát az operandust.

*Direkt címzés:* Az utasítás tartalmazza az operandus címét.

*Indirekt címzés :* Az utasítás tartalmazza azt a címet, ahol az operandus címe megtalálható.

*Indexelt címzés:* az utasítás egyrészt hivatkozik egy index-regiszter nevére, és megad egy címnövekményt (*displacement*), amit az index regiszter tartalmához adva megkapjuk az operandus címét.

### 3.7.3. Utasítások, adatok

Az utasításokat és adatokat a *CPU* a memóriából veszi, és oda is helyezi el az eredményeket.
Az *ADAT-SÍN (DATA-BUS)* tehát kétirányú. Ahhoz, hogy a memóriához más *CPU* is hozzáférjen, az *ADAT-SÍN*-t is lebegtetni kell. Ez tehát *kétirányú, háromállapotú* sín. Ha egy *CPU*-s rendszerben más, intelligens vezérlő-egységek is használják a *CPU*-hoz rendelt memóriát, akkor a *CPU* lekapcsolódása a másik egységnek *DMA*-t (*Direct Memory Access*) tesz lehetővé. Ha több *CPU* használja ugyanazt a memóriát, akkor *memóriában csatolt multiprocesszoros rendszerről* beszélünk.

### 3.7.4. Szekvenciális program

Az utasítások a memóriában növekvő címük sorrendjében helyezkednek el. A *CPU* ebben a sorrendben hajtja végre azokat. A címzés első közelítésben tehát számlálás (*PC,* program-counter). Eltérések ettől a szigorú monotonitástól:

- ***RESET***, azaz kezdeti állapotba állítás. Ilyenkor a *CPU* nullázza a program-számlálót, és a zérus címen újra kezdi a program végrehajtását.

- ***Ugró utasítás*** (JUMP), amely lehet
  -- feltétel nélküli
  -- feltételes
  Az utasítás tartalmazza a címet, ahová ugrani kell. Feltételes ugrás esetén az ugrás csak a feltétel teljesülése esetén következik be. A feltételt szintén az utasítás tartalmazza.

- ***Szubrutin hívás*** (CALL) és ***visszatérés*** (RETURN)
  Az utasítás tartalmazza a szubrutin (alprogram) kezdőcímét. A processzor ide ugrik, és elkezdi ennek az utasítás-sorozatnak, azaz az alprogramnak a szekvenciális végrehajtását. Az alprogram végén egy RETURN, azaz visszatérési utasítás található. A hívás helyére visszatérni a CPU csak akkor tud, ha a hívás előtti *következő utasítás címét* a hívás végrehajtásakor el kell tárolni a memória meghatározott részében (VEREM,

STACK). Mivel alprogramokat tetszőlegesen lehet egymásba-ágyazni, a VEREM speciális része a memóriának.

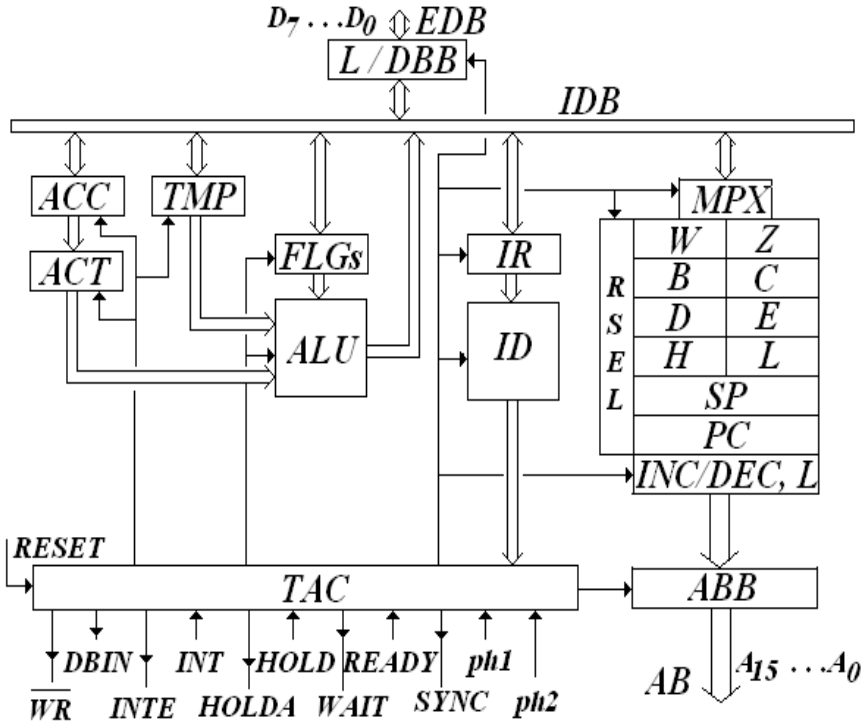
- ***Megszakítás*** (INTERRUPT)
  A program futását valamilyen esemény (*nem programozzuk, de számítunk rá típusú*) megszakítja. Ilyenkor le kell futtatni egy speciális, megszakítási alprogramot, amelynek a végén elhelyezett RETURN visszatérést eredményez ugyan a megszakított program következő utasításához, de a megszakítási alprogram más változásokat is okozhat, ezért a VEREM-ben való tárolás megszakítás kiszolgálásakor sokkal bonyolultabb, mint a CALL-RETURN esetén.

- ***Tartás*** : HOLD
  A HOLD állapotot egy azonos nevű *bemenő-jel* kezdeményezi, ha magas szintre áll. Hatására alkalmas pillanatban processzor megáll, és úgynevezett HOLD állapotba kerül. Ez azt jelenti, hogy a CPU mind az ADAT, mind a CÍM síneket elengedi (lebegteti), és ezzel DMA-ra ad lehetőséget. Ebből az állapotból kilépni a HOLD jel leejtésével lehet.

- ***Megállítás*** : HALT
  Ez olyan speciális utasítás, amelynek hatására a processzor előrehaladása megáll. Egy speciális várakozó állapotba kerül a processzor, amiből *megszakítással* lehet kikerülni.

### 3.7. 5. Egy jellegzetes egyszerű mikroprocesszor architektúra

A mikroprocesszor architektúrájának fő elemei egyrészt az utasítások elővételének és értelmezésének, másrészt végrehajtásának a feladatait látják el (*3.59. ábra*).




*3.59. ábra. Egy egyszerű mikroprocesszor regiszter-átviteli szintű architektúrája*

Vegyük sorra először az utasítás elővétel és értelmezés fő elemeit. Az utasítás-elővétel (FETCH) a programszámlálóból (***PC)*** indul. Maga a ***PC*** az ábra jobb oldalán látható regiszter-halmaz egyik eleme. Egy inkrementálást végrehajtó egység csatlakozik hozzá (***INC/DECR***). A ***PC*** tartalma az utasítás-elővétel első fázisában az ***ABB*** (*cím-sín meghajtó*) egységre kerül. Ezzel a cím megjelenik a memória címző bemenetén, és a memória a megcímzett memória-cella tartalmát a *külső* ***ADAT-SÍN***-re **(*EDB*)** helyezi. Ez az adat az ***L/DBB*** *adatsín tároló és meghajtó* fokozaton át a *belső* ***ADAT-SÍN***-re ***(IDB)*** kerül, ahonnan a processzor azt beírja az ***IR*** *utasítás-regiszterbe*. Az utasítás-regiszterre csatlakozó ***ID*** *utasítás-dekóder* kimenetein megjelennek azok a jel-szintek, amelyek az adott utasítás végrehajtásának logikai feltételeit biztosítják. Fontos, hogy a FETCH végrehajtásával egyidejűleg a ***PC*** inkrementálódik, tehát máris a következő utasítás címére mutat.
Az utasítások végrehajtásának legfontosabb eleme az *aritmetikai-logikai egység (**ALU**)* és a hozzá kapcsolódó speciális regiszterek. Vegyünk egy két

operandusú aritmetikai utasítást. Ennek egyik operandusa mindig az *akkumulátor* (***ACC***) regiszterben van, már az utasítás végrehajtásának megkezdése előtt. A másik operandust be kell hozni az *operandus-regiszterbe (**TMP**)*. Azt hogy azt honnan és hogyan kell ide hozni, azt a már az ***IR***-ben lévő utasítás-kód határozza meg. Tegyük fel, hogy a szóban forgó utasítás direkt-címzéssel határozza meg a másik operandus címét. A processzornak tehát be kell hoznia ezt a címet az utasítás kódot közvetlenül követő memória-cellákból, majd ezt a címet kell az ***ABB***-be juttatnia. Miután a memória kiadta ezt a megcímzett másik operandust, az a belső adat-sínen át bejut a ***TMP*** regiszterbe. Most kezdődhet a tulajdonképpeni aritmetikai számítás. Az ***ACC*** tartalma először az ***ACT*** átmeneti regiszterbe kerül, felszabadítva ezzel a helyet az eredmény számára az ***ACC*** regiszterben. Az aritmetikai művelet eredménye az ***ACT*** és a ***TMP*** regiszterekbe érkező operandusok megjelenésétől számított műveleti időn belül megjelenik az ***ALU*** egység kimenetén. Az utolsó mozzanat, hogy az ***ALU*** kimenetének értéke megjelenik az ***IDB***-n, és az ***ACC***-ba kerül. Az ***ALU***-hoz csatlakozó mutatók (***FLGs)*** bitek az ***ALU***-ban születő eredmények néhány fontos ismérvét mutatják.
Fontos szerepet tölt be a *regiszter-tár,* mind a címzésben, mind adat-tárolásban.
A processzor időzítő-vezérlőjének (***TAC***, *Timing and Control*) feladata, hogy az utasítás-elővételi és végrehajtási ciklusok állapotai alatt a tennivalóknak megfelelő vezérlő-jeleket generálja. Ezek a vezérlő-jelek lehetnek az órajel valamelyik fázisával szinkronizált impulzusok, például regiszterek beíró jelei, vagy valamely állapot valamelyik órajel-fázisára felfutó, valamely másik állapot valamelyik órajel fázisára lefutó jel, például ***ALU*** funkció beállítás. Az időzítő fogadja és generálja is azokat a vezérlő jeleket, amelyek a környezetből származnak, illetve a környezetnek szólnak. A ***TAC*** által generált vezérlő-jelek a rendszert értesítik a következőkről:

- a processzor írási memória vagy **I/O** műveletet hajt végre (***WR***),
- a processzor memóriából, vagy **I/O** egységből olvasását hajt végre (***DBIN***),
- a processzor engedélyezi a megszakítást (***INTE***),
- a processzor a tartó-állapotba való átmenetre vonatkozó kérést nyugtázza (***HOLDA***),
- a processzor várakozó állapotban van (***WAIT***)
- a processzor kihelyezte az adatbuszra a státusz-információt (***SYNC***)

Az időzítő-vezérlő fogadja a következő környezeti vezérlő-jeleket:

- megszakítás-kérés a processzorhoz (***INT***),

- kérés tartó-állapotba való átmenetre (***HOLD***)
- A processzor megszüntetheti a várakozó állapotot, a memória vagy **I/O** elkészült (***READY***)

### 3.7.6. A processzor időbeli működése (IDŐZÍTÉS, TIMING)

A processzorok időzítés-vezérlésének *állapotai gépi ciklusokba* vannak sorolva. A *gépi ciklus* azon állapotok halmaza, amelyek egy memóriával való kommunikációhoz kellenek Egy *gépi állapot* a kétfázisú, nem-átlapolt órajel első fázisának (***ph1***) két felfutása közötti időtartomány. Egy gépi ciklus változó számú állapotból állhat, míg utasítások végrehajtása pedig változó számú gépi ciklusból igényel. Kitüntetett gépi ciklus az első, azaz ***M1*** ciklus, ami a FETCH, azaz az utasítás-elővétel. A legrövidebb végrehajtású utasítások az ***M1*** alatt végre is hajtódnak A leghosszabb végrehajtású utasítások a fenti akár 4-6 gépi ciklust is igényelhetnek

### 3.7.7. Az utasítás-készlet

A mikroprocesszorok utasításkészletét alkotó utasításokat funkciójuk szerint csoportosítjuk, a következőképpen.

- **adatmozgatók**
- **aritmetikaiak**
- **elágazók**
- **verem, I/O és vezérlők**

Adatmozgató utasítások végrehajtásakor a memória-helyek, vagy a memória és a regisztertár közötti adatátvitel zajlik. Az aritmetikai utasítások végrehajtásakor operandusok közötti számítások vagy logikai műveletek mennek végbe a processzorban. Az elágazó utasítások során feltétel nélkül, vagy adott logikai feltételek fennállásakor megváltozik a memóriában levő utasítások cím szerinti sorrendben való elővétele.

### 3.7.8. Néhány kiragadott utasítás és végrehajtása

A processzor működésének megértése érdekében vizsgáljuk meg, milyen vezérlési szekvencia tartozik néhány utasítás elővételéhez és végrehajtásához.

### 3.7.8.1 A 'MOVr,M' ( Move from Memory) utasítás végrehajtása

| | |
|---|---|
| **Az utasítás bináris kódja :** | **0 1 D D D 1 1 0** |
| **Az utasítás definíciója :** | **r(DDD) ← MEM(H_L)** |
| **Ciklusok száma :** | **2** |
| **Állapotok száma :** | **7** |
| **Címzés :** | **regiszter-indirekt** |
| **A jelzőbitek :** | **nem változnak** |

Az utasítás definíciója szerint a processzor a **DDD** regiszter-címmel kijelölt gyors-regiszterbe hozza a **H_L** regiszter-pár aktuális tartalmával megcímzett memória-szót. Az elővétellel együtt két gépi ciklus kell a végrehajtáshoz, és összesen 7 gépi állapot, azaz órajel ciklus. A címzés az indirekt címzésnek egy meghatározott alfaja, amikor a memória-cím egy regiszter-párban található. Az utasítás végrehajtása nem változtatja meg a processzor jelzőbitjeit.

A végrehajtás állapotonként:

**M1, T1 :** **AB ← PC, EDB ← státusz-információ**

**M1, T2 :** **PC ← PC + 1**

**M1, T3 :** **IR ← EDB**

**M1, T4 :**

**M2, T1 :** **ABB ← H_L, EDB ← státusz-információ**

**M2, T2 :** **IDB ← EDB**

**M2, T3 :** **r(DDD) ← IDB**

### 3.7.8.2. Az 'ADD M' ( Add Memory) utasítás végrehajtása

| | |
|---|---|
| **Az utasítás bináris kódja :** | **1 0 0 0 0 1 1 0** |
| **Az utasítás definíciója :** | **ACC ← MEM(H_L) + ACC** |
| **Ciklusok száma :** | **2** |
| **Állapotok száma :** | **7** |

**Címzés :** **regiszter-indirekt**
**A jelzőbitek :** **változnak**

A végrehajtás állapotonként:

**M1, T1 :** **AB ← PC, EDB ← státusz-információ**

**M1, T2 :** **PC ← PC + 1**

**M1, T3 :** **IR ← EDB**

**M1, T4 :** **ACT ← ACC**

**M2, T1 :** **AB ← H_L, EDB ← státusz-információ**

**M2, T2 :** **IDB ← EDB**

**M2, T3 :** **TEMP ← IDB**

**M1, T1 :** **IDB ← ACT + TMP**

**M1, T2 :** **ACC ← IDB**

Vegyük észre, hogy az **ADD M** utasítás végrehajtása átlapolódik a következő utasítás elővételével. Ezt a működés gyorsítására más utasítás végrehajtásánál is alkalmazzák.

### 3.7.8.3. A 'LDA' ( Load Accumulator ) utasítás végrehajtása

**Az utasítás kódja : 0 0 1 1 1 0 1 0**
**laddr**
**haddr**

**Az utasítás definíciója :** **ACC ← MEM(haddr_laddr)**
**Ciklusok száma :** **4**
**Állapotok száma :** **13**
**Címzés :** **direkt**
**A jelzőbitek :** **nem változnak**

**M1, T1 :** **AB ← PC, EDB ← státusz információ**

**M1, T2 :** **PC ← PC + 1**

**M1, T3 :** **IR ← EDB**

**M1, T4 :**

**M2, T1 :** **AB ← PC, EDB ← státusz-információ**

**M2, T2 :** **PC ← PC + 1**

**M2, T3 :** **Z ← MEM(AB)**

**M3, T1 :** **AB ← PC, EDB ← státusz-információ**

**M3, T2 :** **PC ← PC + 1**

**M3, T3 :** **W ← MEM(AB)**

**M4, T1:** **AB ← W_Z, EDB ← státusz-információ**

**M4, T2:** **IDB ← EDB**

**M4, T3:** **ACC ← IDB**

### 3.7.8.4. A 'CALL' ( Call, azaz alprogram hívás) utasítás végrehajtása

**Az utasítás kódja : 1 1 0 0 1 1 0 1**
**laddr**
**haddr**

**Az utasítás definíciója :** **MEM(SP-1) ← h_PC**
**MEM(SP-2) ← l_PC**
**SP ← SP - 2**
**PC ← haddr_laddr**

**Ciklusok száma :** **5**

**Állapotok száma :** **17**
**Címzés :** **közvetlen**
**A jelzőbitek :** **nem változnak**

Az utasítás végrehajtása állapotonként:

**M1, T1 :** **AB ← PC, EDB ← státusz-információ**

**M1, T2 :** **PC ← PC + 1**

**M1, T3 :** **IR ← EDB**

**M1, T4 :**

**M1, T5 :** **SP ←SP-1**

**M2, T1 :** **AB ← PC, EBD ← státusz-információ**

**M2, T2 :** **PC ← PC + 1**

**M2, T3 :** **Z ← MEM(AB)**

**M3, T1 :** **AB ← PC, EDB ← státusz-információ**

**M3, T2 :** **PC ← PC + 1**

**M3, T3 :** **W ← MEM(AB)**

**M4, T1:** **AB← SP, EDB ← státusz-információ**

**M4, T2 :** **SP ← SP -1**

**M4, T3 :** **EDB ← hPC**

**M5, T1:** **AB← SP, EDB ← státusz-információ**

**M5, T2 :** **SP ← SP -1**

**M5, T3 :** **EDB ← lPC**

**M1, T1 :** **AB ← W_Z**

**M1, T2 :** **PC ← W_Z + 1**

### 3.7.8.5. A 'RETURN' ( Return, azaz visszatérés az alprogramból ) utasítás végrehajtása

| | |
|---|---|
| **Az utasítás kódja :** | **1 1 0 0 1 0 0 1** |
| **Az utasítás definíciója :** | **lPC ← MEM(SP)** |
| | **hPC ← MEM(SP+1)** |
| | **SP ← SP + 2** |
| **Ciklusok száma :** | **3** |
| **Állapotok száma :** | **10** |
| **Címzés :** | **regiszter-indirekt** |
| **A jelzőbitek :** | **nem változnak** |

A végrehajtás, állapotonként:

**M1, T1 :** **AB ← PC, EDB ← státusz-információ**

**M1, T2 :** **PC ← PC + 1**

**M1, T3 :** **IR ← EDB**

**M1, T4 :**

**M2, T1 :** **AB ← SP, EDB ← státusz-információ**

**M2, T2 :** **SP ← SP + 1**

**M2, T3 :** **Z ← MEM(AB)**

**M3, T1 :** **AB ← SP, EDB ← státusz-információ**

**M3, T2 :** **SP ← SP + 1**

**M3, T3 :** **W ← MEM(AB)**

**M1, T1 :** **AB ← W_Z**

**M1, T2 :** **PC ← W_Z + 1**

### 3.7.9. A processzor működésének egyéb sajátosságai

#### 3.7.9.1. READY-WAIT

Az (***Mi,T2***) és az (***Mi,T3***) állapotok között általában memória írás vagy olvasás történik. Ha a memória nem olyan gyors, hogy a számára előírt műveletet el egy állapot alatt lehessen végezni, egy (***Mi,Tw***) várakozó állapotot kell beiktatni. Ez egy ***READY-WAIT*** jelpáros segítségével valósul meg. Amíg a memória nem jelzi a ***READY*** segítségével, hogy elkészült, a processzor fenntartja a várakozó állapotot, és a ***WAIT*** jelet.

#### 3.7.9.2. A státusz- információ

A processzor minden gépi ciklus ***T2*** állapotában kiadja a környezetének a ciklusra vonatkozó információt. Ezek mint kódolt bináris vektorok az ***EDB***-n jelennek meg, és a ***SYNC*** lefutó élével eltárolhatók.

Néhány jellemző státusz:

UTASÍTÁS ELŐVÉTEL,
MEMÓRIA OLVASÁS,
MEMÓRIA ÍRÁS,
VEREM OLVASÁS,
VEREM ÍRÁS

A státusz-információt rendszer-vezérlésre használjuk.

#### 3.7.9.3. A legfontosabb jelzőbitek

| | |
|---|---|
| **Z :** | ***1***, ha az ALU-ban képződött eredmény ***0***, |
| **CY**: | ***1***, ha van átvitel a legnagyobb bináris helyi-értékről, |
| **S** : | az ALU-ban képződött eredmény előjele |

### 3.7.9.4. SP értékének beállítása speciális utasítással

A verem-mutató beállítására speciális utasítást definiáltak. Ez az **SPHL** utasítás, amely a **H_L** regisztertár tartalmát betölti a verem-mutatóba.

**SP ← H_L**

### 3.7.9.5. Megszakítás-kezelés

Az ***EI*** utasítás engedélyezi a megszakítást. Végrehajtás : a belső ***INTFF*** (*Interrupt flip-flop*) ***1***-be áll, és ***INTE*** kimenő jel magasra emelkedik. Ha az ***INT*** a külső vezérlő-jelen megérkezik, a következő ***FETCH*** ciklus speciális utasítást fogad. Ez nem a memóriából jön, hanem a megszakítást kérő egységnek kell az ***EDB***-re tennie. Neve ***RST n*** Fontos, hogy a folyamatban lévő utasítást befejezzük, mielőtt az ***RST n***–t elindítjuk. Az ***RST n*** utasítás kódja: **1 1 N N N 1 1 1**

Az **RST n** végrehajtása :

**SP – 1 ← hPC**
**SP- 2 ← lPC**
**SP ← SP – 2**
**PC ← 8 * (N N N)**

A megszakítási szubrutin végén történő visszatérést a megszakítási rutin első utasítása, egy **PUSH PSW** ( tedd a verembe a *program-státusz-szót* ) készíti elő.

**A PUSH PSW definíciója :** **MEM(SP-1) ← ACC**
**MEM(SP-2) ← program-státusz-szó (jelzőbitek)**
**SP ← SP – 2**

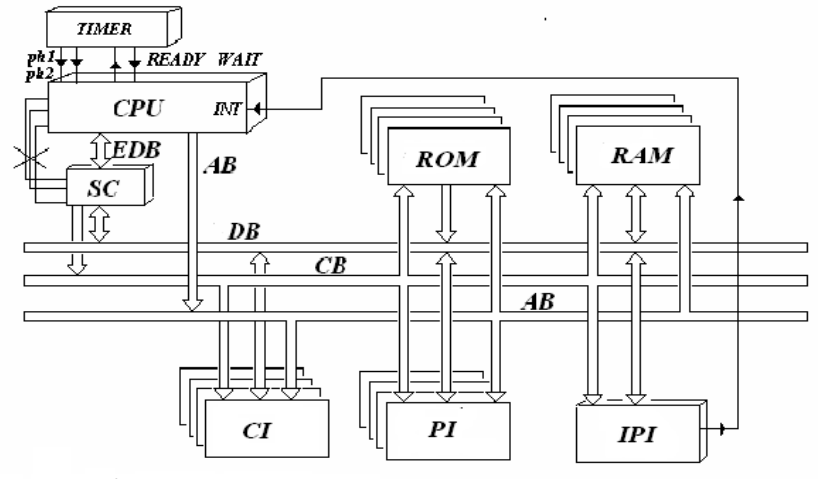
Tehát a **RESTART** elmenti a verembe a ***PC***-t, a megszakítási rutin pedig az akkumulátor, valamint a jelzőbitek tartalmát. Ezek után a megszakítási rutin futhat, de utolsó utasításai a következők egy **POP PSW** és egy **RETURN.** A **POP PSW** definíciója:

**FLAGs ← MEM(SP)**
**ACC ← MEM(SP + 1)**
**SP ← SP + 2)**

A **RETURN** definícióját már ismerjük.

### 3.7.10. A mikroprocesszoros rendszer

Egy mikroprocesszoros rendszer magán a processzoron kívül természetesen számos más rendszer-elemet is tartalmaz. A *3.60.* ábrán bemutatott rendszer a memória modulokat és a processzort kiszolgáló egyéb modulokból áll. A ***ROM*** (*Read Only Memory*), azaz csak olvasható memória) modulok olyan memória áramkörök, amelyekbe adatot írni nem lehet, az azokban rögzített adatok azonban ugyanolyan címzési és vezérlési eljárásokkal olvashatók ki, mint az írható-olvasható memóriák esetében. Az írható olvasható memória-modulok a ***RAM*** (Random Access Memory) egységben foglalnak helyet. Ez az elnevezés nem a funkcióra utal, hanem arra, hogy az adatok közül bármelyiket közel azonos idő alatt érhetjük el. A memóriákkal való kommunikációt a ***TIMER***-egység azzal segíti, hogy a lassú memóriától származó késlekedő ***READY*** jelet szinkronizálja a processzor számára, és fogadja a ***WAIT*** jelet. Az ***SC*** (*System Controller, rendszer-vezérlő*) a ***SYNC*** jellel szinkronban eltárolja a státusz információt, és ennek segítségével megfelelő módon vezérli a processzort, de a rendszer ***CB*** (*Control Bus*) vezérlő-sínjét is. A ***CB*** összeköti a processzor külső adat-sínjét a rendszer adat-sínnel (***DB***). A rendszert az I/O kommunikációt segítő egységek egészítik ki. A kommunikáció több, szabványos rendszerét segítő kommunikációs interfész (***CI***), a perifériákkal való kommunikációt vezérlő periféria interfész (***PI***), és a megszakítás-kéréseket prioritási sorrendbe állító ***IPI*** (*Interrupt Priority Inteface*) modul.




*3.60. ábra. Mikroprocesszoros rendszer*

### 3.7.11. A mikroprocesszoros rendszerek ASSEMBLY szintű programozása

Egy igen egyszerű példával illusztráljuk a gépi kódhoz igen közel álló ***ASSEMBLY*** nyelvű programozás sajátosságait. Tegyük fel, hogy egy címlista elemei a memóriában szétszórt adatok címeit tartalmazza. A címlista hossza nincs meghatározva, ezért a végét egy speciális cím jelzi, legyen ez az $FFFF_H$, azaz a maximális 4-jegyű hexadecimális sorszám. A program feladata, hogy a címlista elemeit egyenként beolvassa, megvizsgálja, hogy nem a lista-végét jelző cím-e, és ha nem, akkor hozza be a memóriából a listaelemmel megcímzett adatot, és adja hozzá a már kialakult részösszeghez. Az olvasó feladata, hogy a mellékelt utasítás-definíciók alapján értelmezze az ASSEMBLY nyelvű programot.

| *Címke* | *Mnem* | *Param* | *Megj* |
|---|---|---|---|
| | **LXI** | **H, LIST** | **A lista kezdőcíme a H_L be** |
| | **CALL** | **SUMMA** | **rutin hívása** |
| **SUMMA:** | **XRA** | **A** | **Az ACC törlése** |
| **LOOP:** | **MOV** | **C, A** | **Az ACC áttöltése a C regiszterbe** |
| | **MOV** | **E, M** | **Az első adat címének alsó része az E-be kerül (H_L címzés)** |
| | **INX** | **H** | **A H_L növelése** |
| | **MOV** | **A, M** | **Az első adat címének felső része ACC-ba kerül** |
| | **CPI** | **FF** | **Ha ACC tartalma FF,** |

```
                                        Z = 0 lesz
            JZ          BACK            Ha vége, ugrás
                                        a  BACK  címkére
            MOV         D, A            Ha nincs vége,
                                        cím alsó fele D-be
            LDAX        D               adatbyte a D_E ben
                                        megadott címről  az
                                        ACC-ba
            ADD         C               ACC hozzáadása C-
                                        hez
            JC          OVER            Ha CY=1,
                                        túlcsodulás,
                                        ugrás OVER-re
            INX         H               H_L növelése
            JMP         LOOP            Visszaugrás
BACK:       MOV         A, C            Összeg vissza az
                                        ACC-ba
            RET                         Visszatérés
OVER:       NOP                         A túlcsordulás
                                        lekezelésének kezdete
```

### 3.7.12 A még nem ismert utasítások definíciói

**LXI : h r ← byte3, l r ← byte2**
(az utasításkód utáni két byte az **r**-regiszter-párba kerül

**XRA: ACC ← A (XOR) r**
**(** az **ACC** és a megcímzett regiszter **XOR** eredménye
az **ACC**-ba kerül)

**INX: h r_l r ← h r_l r + 1 (** az r regiszter-pár tartalma inkrementálódik**)**

**CPI: ACC – byte2**
**(**az **ACC** tertalmából az **ALU** kivonja az utasítás utáni byte-t, és beállítja a jelzőbiteket**)**

**JZ: if Z = '1' then PC ← byte3_byte2** (ugrás, ha **Z = '1'**)

**LDAX: ACC ← MEM(h r_l r)**
**(** az ACC-t betöltjük a megcímzett regiszter-párban levő memória-címről)

**JC: if CY = '1' then PC ← byte3_byte2**
(ugrás, ha a CY jelzőbit = '1')

**JMP: PC ← byte3_byte2** **(**ugrás feltétel nélkül**)**

**NOP: (**No-operation,azaz üres utasítás**)**

*KÉRDÉSEK*

Sorolja fel az összetett építőelemek három fő csoportját!

Határozza meg a multiplexerek, a regiszterek és funkciós egységek fogalmát!

Hány **4-1** multiplexer kell négy **16** bites adatból egynek a kiválasztásához?

Milyen logikai szinteket kell kapcsolni egy hosszú, **4-bites** elemekből álló komparátor legjelentősebb 4 bitjét képviselő elem **Li, Ei, Mi** bemeneteire?

Egy **T** flip-flop órajelének frekvenciája ***f1***. A **T** bemenet állandó magas szinten van. Mekkora a **Q** kimeneten megfigyelt jel
frekvenciája?

Egy **mod–n** szinkron számláló **CY** logikája mit jelez?

Egy **4** dekádos számláló legfelső dekádjának **ENABLE** bemenetére hány bemenetű **ÉS** kapu kapcsolódik?

Milyen logikai függvényt teljesít egy bináris álvéletlen generátor visszacsatoló logikája?

Egy kétfázisú időzítő melyik regiszterének kimeneteit használjuk a ***ph1*** órajellel szinkronizált hazárdmentes vezérlésre?

Mi a megszakítás, és mi a szerepe a mikroprocesszoros rendszerekben?

Sorolja fel a regiszter-átviteli szint építőelemeinek három fő csoportját!

Hány **4-1** multiplexer kell **16** darab, **1** bites adatból egynek a kiválasztásához?

Hány **4**-bites komparátor kell két **1** byte hosszúságú bináris szám nagyság szerinti rendezéséhez?

Milyen flip-flop alkalmas kettes osztóvá való kiképzéshez?

Hány szóból áll az a memória, amelynek végigcímzéséhez éppen három **mod-16**-os számláló kaszkádja kell?

Egy **5** dekádos számláló legfelső dekádjának **ENABLE** bemenetére hány bemenetű **ÉS** kapu kapcsolódik?

Milyen logikai függvényt teljesít egy bináris álvéletlen generátor visszacsatoló logikája?

Egy kétfázisú időzítő melyik regiszterének kimeneteit használjuk a ***ph2*** órajellel szinkronizált hazárdmentes vezérlésre?

Mi az utasítás-dekóder?

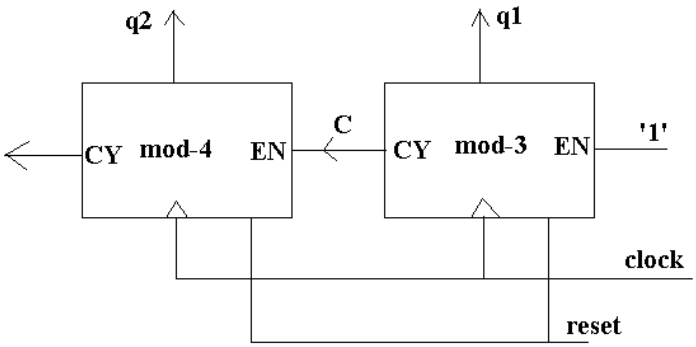
*TERVEZÉSI FELADATOK 5.*

1. Tervezzünk **4 – 1** Byte multiplexert, **4 – 1** bites egységekből!

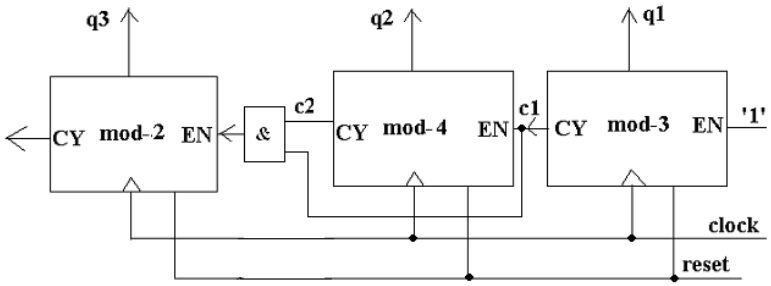
2. Tervezzünk **8 – 1** Byte multiplexert, **4 – 1** bites egységekből!

3. Tervezzünk **3** számjegyes szinkron dekadikus számlálót **mod-16** egységek felhasználásával!

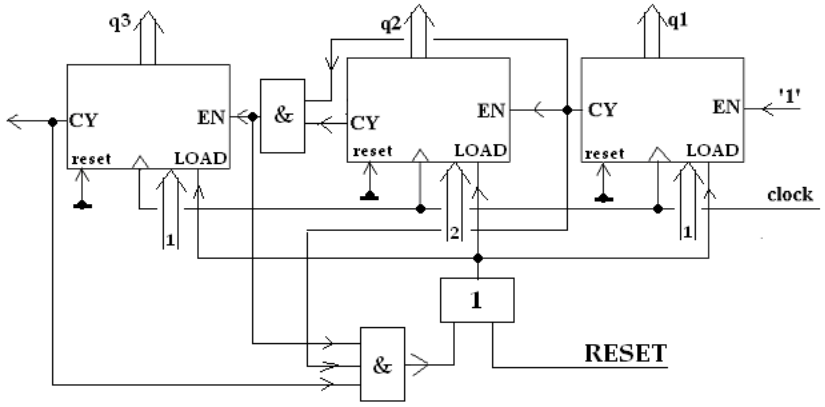
4. Analizáljuk az ábra szerinti szinkron két komponensből álló számlálót!




5. Analizáljuk az ábra szerinti szinkron, három komponensből álló számlálót!



6. Analizáljuk az ábra szerinti szinkron, három komponenből álló számlálót! A modulo-értékek balról jobbra : ***4, 3, 4***




7. Tervezzünk teljes összeadót **4-1** multiplexerekből!

8. Tervezzünk kizárólag **4-1** multiplexerek felhasználásával olyan áramkört, amely egy **S** vezetéken megadott logikai szinttel választ egy két-bemenetű **EXOR** és egy két-bemenetű **NAND (NÉS)** függvény realizálása között.

8. Tervezzünk **3** bemenetű **MAJORITÁS** egységet **4-1** multiplexerekkel! (A kimenet akkor magas, ha legalább két bemenet magas)

9. Tervezzünk kettes osztókkal **4**-csatornás impulzus-generátort, amely egy impulzussorozat minden *1., 5., 9, . . .* impulzusát a ***C1*** kimeneten*, 2., 6., 10.,. . .* impulzusát a ***C2*** kimeneten*, 3., 7., 11. . . .* impulzusát a ***C3****,* és *4., 8., 12. . . . .* impulzusát pedig a ***C4***-en jeleníti meg.

11. Tervezzünk olyan impulzus-generátort, amely egy órajel minden 4. impulzusát engedi át! Oldjuk meg a feladatot kettes osztóval!

12. Tervezzünk olyan áramkört, amely két vezérlő-bittel a következő funkciókat teljesíti:
00 → 4 bites, párhuzamos betöltésű párhuzamos – soros átalakító

01 → 4 bites párhuzamos betöltésű, visszacsatolt shift regiszter
10 → 4 bites párhuzamos betöltésű Johnson számláló
11 → 4 bites párhuzamos betöltésű véletlen szám generátor.
A párhuzamos betöltés egy **LOAD** nevű vezérlőjel magas szitjére, az órajellel szinkronban történjen. Élvezérelt, **PRESET-CLEAR** nélküli **D-MS** tárolókat használjunk.

13. Tervezzünk **8**-bites, előjeles abszolút-értékes bitvektorokból komplemens vektorokat előállító egységet !

14. Egy 5-bites **SHIFT**-regiszteres álvéletlen generátorba bekapcsoláskor a **21**-es szám bináris megfelelőjét töltjük. Írja fel az ezután következő **5** számot!

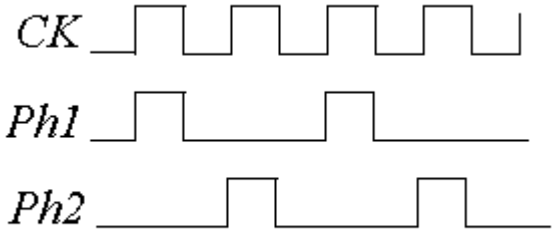
15. Tervezzünk **EXOR** áramkört **2-1 CMOS** átvivő-kapus multiplexer és inverter felhasználásával !

16. Tervezzünk **MOD-1234** törölhető szinkron számlálót dekadikus (**MOD 10**) számláló egységek felhasználásával! A számláló modulok rendelkeznek ***EN*** (*enable*) bemenettel és ***CY*** kimenettel, valamint szinkron **RESET** bemenettel.

17. Analizáljunk egy párhuzamos betöltésű, **4** bites *Johnson* számlálót, ha kezdeti állapota (**0101**).

18. Tervezzünk kizárólag **4-1** multiplexerek felhasználásával olyan áramkört, amely egy **S** vezetéken megadott logikai szinttel választ egy két-bemenetű **EXOR** és egy két-bemenetű **NAND (NÉS)** függvény realizálása között. A bemenetek : **X1, X2**.

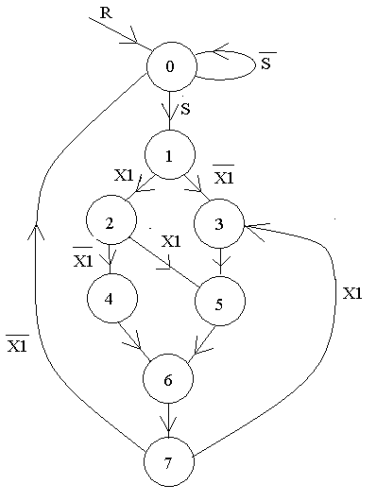
19. Tervezzünk nem átlapolt kétfázisú órajelet egyfázisú órajelből előállító óragenerátort kettes osztó és kapuk segítségével!




20. Tervezzünk **4**-bites inkrementer (1-el növelő) áramkört kizárólag **4-1** multiplexerek felhasználásával. (Például a **0110** bemenetre **0111** a kimenet)

21. Tervezzünk 2-bites párhuzamos átvitelképzésű teljes összeadót, **NÉS** kapukból!

22**.** Valósítsa meg engedélyezhető ***(E),*** betölthető ***(L),*** törölhető ***(R)*** ***mod-16*** szinkron számlálóval az alábbi állapotgráfot.
Az ***5***-ös állapotban egy ***lreg*** nevű beírójelet kell előállítani, az ***1***-es állapotban pedig egy foglaltságot jelző (***BUSY***) külső vezérlőjelet kell felemelnie, amelyet a *7*-s állapotban vissza kell venni.



23. Realizáljuk kétfázisú **FSM**-vel a következő autonom összeadó egységet! Az egység egyik operandusa az **OP1** regiszterben van, a másikat egy külső sínről kell betölteni az **OP2** regiszterbe, ha egy **START** jel elindítja a folyamatot. Ezután az egység elvégzi az összeadást, és az eredményt az **OP1** regiszterbe tölti. Ha a rendszer elkészült, tovább adja a vezérlést egy **CONT** jellel, maga pedig visszatér a **START**-.ra várakozó állapotba. A regiszterek betöltését a ***ph2*** órajellel szinkronizáljuk!

**Ajánlott irodalom :**


1. **Arató Péter: Logikai rendszerek tervezése**
   **Műegyetemi Kiadó. 2002**

2. **Benesóczky Zoltán: Digitális tervezés funkcionális elemekkel és mikroprocesszorokkal**
   **Műegyetemi Kiadó, 2002**

3. **Janovics-Tóth**
   **A logikai tervezés módszerei**
   **Műszaki Könyvkiadó, 1976**

4. **Daniel Gajsky, Nikil Dutt, Allen Wu, Steve Lin:**
   **High Level Synthesis**
   **Kluwer Academic Publishers**

5 **M. A Breuer : Design Automation of Digital Systems**

5. **J. Hartmanis, R.E Stearns : Algebraic Structure Theory of Sequential Machines.**
   **Prentice Hall, Englewood Cliff, N.J. 1966**

7. **D. Gajsky, N. Dutt, A Wu, S. Lin : High Level Synthesis**
   **Kluwer Academic Publishers1993**

8. **L. Lavagno, A Sangioanni-Vincentelli :Algorithms for Synthesis and Testing of Asynchronous Circuits**
   **Kluwer Academic Publishers**

9. **R. K. Brayton, G.D. Hachtel, C.T. McMullen, A.L. Sangiovanni-Vincentelli**
**Logic Minimization Algorithms for VLSI Syntehsis**
**Kluwer Academic Publishers**

10. **D. E. Thomas, E.D. Lagnese, R.A. Walker, J.A. Nestor, J.V. Rajan, R.L Blackburn**
**Algorithmic and Register-Transfer Level Synthesis : The System Architect Workbench**
**Kluwer Academic Publishers**