A könyv a digitális technika klasszikus logikai tervezési eljárásait ismerteti. A kombinációs és sorrendi hálózatok leírási és logikai tervezési módszereinek tárgyalásán alapulva mutatva be a vezérlőegységek fázisregiszteres és mikroprogramozott felépítési módjait és ezen keresztül jut el a mikroprocesszorok lényegének egyfajta megközelítéséhez. A könyv anyaga nem lép fel azzal az igénnyel, hogy teljes szakirodalmi áttekintést adjon, vagyis az összes ismert módszert és megközelítést ismertesse, célja inkább az alapgondolatok bemutatása és a szemléletmód kialakítása többnyire induktív jelleggel, egyszerű példákon keresztül. A könyv a BME Villamosmérnöki és Informatikai Karon folyó digitális technika alapképzéshez készült, ezért a tárgyalásmód megválasztásakor fontos szempont volt, hogy az anyag megértése ne igényeljen egyetemi előképzettséget.

55013

55013

LOGIKAI RENDSZEREK TERVEZÉSE

Arató

BUDAPESTI MŰSZAKI ÉS
GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR

Arató Péter

LOGIKAI RENDSZEREK TERVEZÉSE

Műegyetemi Kiadó

Bírálók:

Frigyes Andor a műszaki tudományok doktora, tanszékvezető egyetemi tanár
Selényi Endre a műszaki tudományok kandidátusa

© Dr. Arató Péter, 1992

(Tizenharmadik utánnyomás)

Azonosító: **55013**

A Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Karának
megrendelése alapján kiadja a
Műegyetemi Kiadó
Felelős vezető: Wintermantel Zsolt
Terjedelem: 32,5 (A/5) ív
Nyomdai munkák:
PLANTIN Kiadó és Nyomda Kft
Felelős vezető: Gollob Józsefné

Tartalomjegyzék

Előszó 9

1. A logikai tervezési feladat megfogalmazása 12

1.1. A logikai feladat fogalma 12
1.2. A logikai hálozat fogalma 13
1.3. A logikai hálozatok csoportosítása a megoldandó logikai feladatok szerint . . . 17
1.4. A logikai hálozat és a logikai rendszer kapcsolata 18
1.5. A logikai tervezés célja 22

2. Kombinációs hálozatok tervezése 24

2.1. Kétérkű jelekkel végzett logikai műveletek algebrai leírása 24
2.1.1. A logikai érték fogalma 24
2.1.2. A logikai alapműveletek definiciója 25
2.1.3. A logikai változók és a logikai algebrai kifejezés fogalma 26
2.1.4. Adott számú bemenet és kimenet esetén megoldható logikai feladatok száma 28
2.2. Logikai függvények 29
2.2.1. A logikai függvény fogalma 29
2.2.2. A logikai függvények megadási módjai 29
2.2.3. Azonos változókon értelmezett logikai függvények közötti összefüggések 33
2.2.4. A logikai függvények kanonikus algebrai alakjai 35
2.2.5. Az elvi logikai rajz 41
2.3. A logikai függvények minimalizálása 46
2.3.1. A grafikus minimalizálás 48
2.3.2. A számjegyes minimalizálás (Quine-McCluskey-módszer) 60
2.3.3. Több kimentű kombinációs hálózatok kimeneti függvényeinek grafikus minimalizálása 71
2.3.4. Több kimentű kombinációs hálózatok kimeneti függvényeinek számjegyes minimalizálása (Quine-McCluskey-módszer) 79
2.3.5. Szimetrikus logikai függvények 87
2.4. A jelterjedési idők hatása a kombinációs hálózatok működésére 101
2.4.1. A jelterjedés kéleltetésének oka 102
2.4.2. A statikus hazárd 103
2.4.3. A dinamikus hazárd 110
2.4.4. A funkcionális hazárd 111

2.5. A több szintű kombinációs hálózatok tervezésének néhány alapgondolata . . . 114
2.5.1. Több szintű kombinációs hálózatok tervezése a logikai függvények dekompoziciójának elvén . . . 115
2.5.2. Több szintű kombinációs hálózatok NAND és NOR kapukból történő felépítésének egy módszere . . . 122
2.6. Kombinációs hálózatok megvalósítása memória elemek felhasználásával . . . 124
2.7. Kombinációs hálózatok megvalósítása PLA elemek felhasználásával . . . 133
3. Sorrendi hálózatok tervezése . . . 139
3.1. A sorrendi hálózatok fogalma és csoportosítása . . . 139
3.2. A sorrendi hálózatok működésének leírása . . . 151
3.2.1. Az állapottábla . . . 151
3.2.2. Az állapotgráf . . . 157
3.3. Elemi sorrendi hálózatok (flip-flopok) . . . 158
3.3.1. S-R flip-flop . . . 159
3.3.2. J-K flip-flop . . . 162
3.3.3. T flip-flop . . . 163
3.3.4. D-G flip-flop . . . 164
3.3.5. D flip-flop . . . 165
3.3.6. A flip-flopok működésének összefoglaló ábrázolása . . . 167
3.3.7. A különböző flip-flop típusok felhasználása egymás megvalósítására . . 168
3.4. Szinkron sorrendi hálózatok tervezési lépéseinek bemutatása egy egyszerű példán . . . 174
3.5. Aszinkron sorrendi hálózatok tervezési lépéseinek bemutatása egy egyszerű példán . . . 189
3.6. A szinkronizációs feltételek biztosítása az integrált áramköri építőelemek szinkron flip-flopjaiban . . . 203
3.7. Sorrendi hálózatok analízise . . . 213
3.8. Állapot-összevonási eljárások . . . 215
3.8.1. Állapot-összevonás teljesen specifikált hálózatok esetén . . . 215
3.8.2. Az állapotekvivalencia tulajdonságai . . . 223
3.8.3. Állapot-összevonás nem teljesen specifikált hálózatok esetén . . . 224
3.8.4. Az állapot kompatibilitás tulajdonságai . . . 235
3.9. Állapotkódolási eljárások . . . 236
3.9.1. Szinkron sorrendi hálózatok állapotkódolása . . . 239
3.9.1.1. Szomszédos kódolás . . . 242
3.9.1.2. Önfüggő szekunder változócsoportok szerinti kódolás . . . 245
3.9.2. Aszinkron sorrendi hálózatok állapotkódolása . . . 256
3.9.2.1. Instabil állapotok módosítása . . . 257
3.9.2.2. Átvezető állapotok felvétele . . . 259

3.9.2.3. Állapotkódolás megkülönböztető particiók alapján (Tracey-Unger -módszer) . . . 261
3.9.2.4. A bemeneti és a szekunder változások érzékelési sorrendjének hatása az állapotkódolásra . . . 269
3.9.2.5. A lényeges hazárd vizsgálata . . . 277
3.9.2.6. Összefoglaló következtetések az aszinkron sorrendi hálózatok állapotkódolásával kapcsolatban . . . 279
3.10. Sorrendi hálózatok megvalósítása memória- és PLA-elemekkel . . . 280
3.11. Sorrendi hálózatok kiindulási állapotának biztosítása . . . 281
4. Vezérlő egységek tervezése . . . 285
4.1. A vezérlő egységek működésének leírása folyamatábrával . . . 287
4.2. Fázisregiszteres vezérlőegység tervezése a folyamatábra alapján . . . 298
4.2.1. Szinkron fázisregiszteres vezérlőegység tervezése . . . 306
4.2.1.1. A szinkronizációs feltételek biztosítása a szinkron fázisregiszteres vezérlőegységben . . . 306
4.2.1.2. A szekunder változók számának csökkentése a folyamatábra alapján . . . 312
4.3. Mikroprogramozott vezérlőegység tervezése . . . 329
4.3.1. A folyamatábra elemi műveleteinek ábrázolása a memóritartalomban . . 331
4.3.2. A processzor működésének leírása . . . 337
4.3.3. A processzor tervezése . . . 338
4.3.3.1. Szinkron működésű belső vezérlőegység tervezése . . . 340
4.4. A fázisregiszteres és amikroprogramozott megoldás összehasonlítása a vezérlőegység működési sebessége szempontjából . . . 350
Függelék. Gyakorló feladatok . . . 355
F.1. Kombinációs hálózatok (2. fejezet) . . . 355
F.2. Sorrendi hálózatok (3. fejezet) . . . 364
F.3. Vezérlőegységek (4. fejezet) . . . 387
Irodalomjegyzék . . . 397

Előszó

A digitális technika mindenkori fejlettségi fokán logikai tervezésnek nevezhetjük azt a tevékenységet, amelynek során egy adott működési leírásból kiindulva meghatározzuk, hogy a készen kapható digitális építőelemeket miként kell összekapcsolni egy előírt működésű logikai rendszer létrehozása céljából. A logikai tervezés során általában számos peremfeltételt kell figyelembe venni (gazdaságos építőelem-felhasználás, előírt működési sebesség, tranziens viselkedés, megbízhatóság, bemérhetőség stb.). A digitális építőelemek előállítási technológiájának rohamos fejlődése következtében a logikai tervezés szempontjai és peremfeltételei szintén gyorsan változnak. Az ún. diszkrét építőelemek korában alakultak ki azok az ún. klasszikus tervezési eljárások, amelyek alapgondolata alkalmazható volt a kis integráltsági fokú integrált áramköri elemek megjelenése után is. Az integrált áramköri technika gyors fejlődése egyre nagyobb integráltsági fokú elemeket hozott létre, amelyek alkalmazása a logikai tervezési módszerekkel szemben is új és egyre változó követelményeket támaszt. Ezt a gyors, szakadatlan fejlődést természetesen követnie kell a digitális technika területén folyó egyetemi képzésnek is.

A mindenkori szükséges ismeretanyag és tervezői készség elsajátításához rendelkezésre álló képzési időt akkor lehet hatékonyan kihasználni, ha a gyorsan változó, legkorszerűbb ismeretek olyan alapképzésre épülnek, amely távlatilag is alkalmazható tervezési alapgondolatokat mutat be és kialakítja a további szakképzés által igényelt szemléletmódot is.

Ennek a tankönyvnek az a célja, hogy a Budapesti Műszaki Egyetem Villamosmérnöki Karán folyó digitális technikai alapképzést segítse a fenti értelemben. Ezért az anyag összeállításakor és a tárgyalásmód megválasztásakor fontos követelmény volt, hogy a tankönyv anyagának megértése ne igényeljen egyetemi előképzettséget. A tárgykör jellegéből következően ezért az alapgondolatok bemutatása és a szemléletmód kialakítása egyszerű példákon keresztül többnyire induktív jelleggel történik. Az olvasóra lehet bízni azokat az általánosításokat, amelyek csak későbbi tanulmányai során válnak időszerűvé.

A logikai tervezési módszerek alapgondolatának bemutatásakor már csak azért sem célszerű minden általánosítást az alapképzés szintjén megtenni, mert a digitális technika mindenkori fejlettségi fokán sokszor még az sem ítélhető meg egyértelműen, hogy a gyors fejlődés következtében egy adott tervezési módszer nem kerül-e át igen rövid időn belül az építőelemek előállításának területére (pl. aszinkron sorrendi hálózatok tervezése). Ilyen esetben az új, eltérő követelmények és peremfeltételek

miatt esetleg más irányú általánosítások válnak szükségessé, megőrizve azonban a tervezés alapgondolatát.

Fentiekből következően a könyv anyaga nem lép fel azzal az igénnyel, hogy teljes szakirodalmi áttekintést adjon, vagyis az összes ismert módszert és megközelítést bemutassa. Különösen igaz ez a vezérlőegységek tervezésével foglalkozó anyagrészre, amely — mivel a szakirodalomban nem ismeretesek a klasszikus tervezési eljárásokhoz hasonlóan általánosan elterjedt módszerek — a BME Folyamatszabályozási Tanszék egy munkacsoportjának kutatási eredményein alapul. A módszerek bemutatásához használt példák alapján levont következtetések végiggondolása, valamint a teljes anyag áttanulmányozása és megértése után az olvasó remélhetőleg megfelelő készséget szerez ahhoz, hogy kialakuljon a további digitális technikai tanulmányait hatékonyabbá tevő szakmai ítélőképessége és lényeglátása.

A tankönyv anyaga a kombinációs és sorrendi hálózatok klasszikus tervezési eljárásaitól kezdődően a vezérlőegységek fázisregiszteres és mikroprogramozott felépítési módjain keresztül jut el a mikroprocesszorok lényegének egyfajta megközelítéséhez. Az anyag ezen a ponton fejeződik be, mert az ismeretek további konkretizálását már erősen befolyásolja a rendelkezésre álló építőelem-készlet mindenkori fejlettsége. A további szakképzés feladata, hogy ezeket a gyorsan változó ismereteket tananyag-korszerűsítés révén megfelelően kövesse.

A tankönyv ezt a követést hivatott segíteni azáltal, hogy anyaga alapján olyan, viszonylag időtálló ismeretek jelölhetők ki, amelyek megkönnyítik a további szakképzés tananyagának a gyors fejlődés által megkövetelt szinte folyamatos korszerűsítését, az alapképzés anyagának nagy részét változatlanul hagyva.

A könyv anyagának alapvető elsajátítása utáni ismétléseket és a rendszerezést hivatottak segíteni az egyes fejezetek elején levő rövid célkitűzések és áttekintések, valamint az ábrafeliratok.

A függelékben szereplő gyakorló feladatok megoldásával az anyag önálló alkalmazási készsége ellenőrizhető.

Az elméleti háttér mélyebb megismerésének igénye esetén az egyes fogalomcsoportokat ajánlatos az irodalomjegyzékben közölt művek alapján feldolgozni az alábbi bontásban:

Absztrakt algebra: [18].
Véges automaták elmélete: [6], [12], [15], [16].
Kombinációs hálózatok: [5], [10], [13], [19].
Logikai függvények minimalizálása: [3], [10], [19].
Hazárdjelenségek formális leírása: [3], [5].
Sorrendi hálózatok: [4], [5], [10].
Állapotösszevonási eljárások: [6], [12], [15].
Állapotkódolási eljárások: [3], [4], [7], [10], [11].
Vezérlőegységek működésének leírása: [8], [9], [20], [21].
Mikroprogramozott vezérlőegységek: [9], [17].

Ezúton is kifejezem köszönetemet Dr. Frigyes Andor tanszékvezető egyetemi tanárnak, aki azonkívül, hogy a könyv megírását szorgalmazta és ehhez a megfelelő munkahelyi feltételeket biztosította, kezdetben szerkesztőként, később a kézirat egyik lektoraként nélkülözhetetlen segítséget nyújtott számomra.

Köszönet illeti dr. Selényi Endre docenst, aki lektori kötelezettségét is jóval meghaladó részletességgel tanulmányozta át a kéziratot. Számos értelemzavaró, sokszor rejtett hibára hívta fel a figyelmemet, és mint a könyv témakörének tapasztalt oktatója olyan hasznos szakmai tanácsokat adott, amelyek — megítélésem szerint — a kéziratot szakmailag és didaktikailag egyaránt igen kedvezően befolyásolták.

Köszönettel tartozom a BME Folyamatszabályozási Tanszék kollektívájának, ezen belül is közvetlen munkatársaimnak:
dr. Kalmár Péter docensnek, dr. Lantos Béla docensnek, dr. Kondorosi Károly adjunktusnak, dr. Risztics Péter adjunktusnak, dr. Terplán Sándor adjunktusnak, dr. Grantner János adjunktusnak, dr. Horváth István tudományos munkatársnak, és dr. László Zoltán tudományos munkatársnak.

Dr. Kalmár Péter, dr. Risztics Péter és dr. Kondorosi Károly szerzőtársaim voltak a jelen könyv előzményeinek is tekinthető egyetemi jegyzetek megírásakor. Így akkori munkájuk eredményei a könyv számos fejezetének kidolgozását jelentősen megkönnyítették számomra. A vezérlőegységek fázisregiszteres megvalósításával foglalkozó fejezetek dr. Kalmár Péter és dr. Terplán Sándor önálló kutató munkájának eredményein alapulnak. Így segítségük természetesen nélkülözhetetlen volt számomra az általuk kidolgozott tervezési módszerek didaktikai megközelítésében és leírásában. A könyv ábráinak gondos szerkesztéséért és rajzolásáért dr. Kalmár Péternét illeti köszönet. A kézirat gépeléséért Haris Klárának, Saáry Bélánénak és Parádi Lászlónénak tartozom köszönettel.

Budapest, 1982. július

A szerző

1. A logikai tervezési feladat megfogalmazása

Ebben a fejezetben először körülhatároljuk a logikai feladat és a logikai hálózat fogalmát, majd csoportosítjuk a logikai hálózatokat a megoldandó logikai feladatok szerint. Bevezetjük a logikai rendszer fogalmát és a rendszertechnikai felépítés bemutatása során szemléltetjük a logikai hálózatok és a logikai rendszer kapcsolatát. Ezután megfogalmazzuk a logikai tervezés végeredményének tekinthető logikai kapcsolási terv legfontosabb tulajdonságait.

1.1. A logikai feladat fogalma

A tervezési feladat megfogalmazásához először tisztáznunk kell, hogy milyen feladatokat nevezünk logikai feladatoknak. A könnyebb érthetőség céljából kezdjük egy ellenpéldával és fogalmazzuk meg, hogy milyen feladatot lát el pl. egy feszültségerősítő. A feszültségerősítő feladatát úgy is megfogalmazhatjuk, hogy a bemeneti feszültség minden egyes értékéhez hozzárendeli a kimeneti feszültség egy-egy értékét. Mivel erősítő, a kimeneti feszültség értéke nagyobb, mint a bemeneti. A bemeneti feszültségnek egy tartományban végtelen sok értéke lehet, és ezektől függően a kimeneti feszültség is végtelen sok értéket vehet fel egy adott tartományon belül. Pontosabban: a bemeneti és kimeneti jelek közötti kapcsolatot a bemeneti jel egy meghatározott értelmezési tartományán belül folytonos függvény írja le. E függvény megadásának sokféle módja lehet (analitikus, grafikus stb.). Az olyan jeleket, amelyek egy tartományon belül tetszőleges értékeket vehetnek fel, s a különböző értékekhez különböző információtartalmat rendelünk, *analóg jeleknek* nevezzük. Az ellenpéldánkban szereplő erősítő feladatát tehát azzal fogalmazzuk meg, hogy előírjuk az analóg bemeneti és kimeneti jelek között megkívánt folytonos függvény kapcsolatot.

Ha azonban a bemeneti és kimeneti jelek csak véges számú (diszkrét) értéket vehetnek fel, akkor az elvégzendő feladat azzal definiálható, hogy felsoroljuk a bemeneti jelek összes lehetséges értékét, és megadjuk, hogy ezek mindegyikéhez a kimeneti jelek mely diszkrét értékei tartoznak. Ebben a megfogalmazásban az egyes bemeneti jelértékek a hozzájuk rendelt kimeneti jelérték *keletkezésének* feltételeit jelentik. Az egyes kimeneti jelértékek tehát úgy tekinthetők, mint az őket létrehozó bemeneti jelértékek fennállásának *következményei*. Így a megfogalmazás során az egyes feltételeket (bemeneti diszkrét jelértékek) soroljuk fel. A tervezési feladat során általában a szöveges megfogalmazásból kell kiindulnunk. Példaként vizsgáljuk meg az alábbi feladatot.

A felvonó *csak* akkor induljon el, ha ajtaja csukva van *és* a fülkében levő emeletjelző gombok valamelyike be van nyomva.

E feladat a négyféle feltétel mindegyikéhez a lehetséges kétféle következmény egyikét rendeli hozzá.

Az 1.1. ábrán a feltételek és teljesülésük következményei táblázatos összefoglalásban láthatók.

Feltétel		Következmény
Ajtó	Emeletkiválasztó-gomb	A felvonó
nyitva	egyik sincs megnyomva	nem indul el
nyitva	valamelyik megnyomva	nem indul el
csukva	egyik sincs megnyomva	nem indul el
csukva	valamelyik megnyomva	elindul

1.1. ábra. Feltétel—következmény hozzárendelés táblázatos megadása

Ha a feltétel—következmény hozzárendelést ember végzi, akkor tevékenysége az alábbi részekre bontható:

1. A mindenkori teljesülő feltételek tudomásulvétele.
2. A feladat ismeretében és annak előírásai szerint az éppen teljesülő feltételekhez a megfelelő következmény kiválasztása azok közül a lehetséges következmények közül, amelyeket a feladat megfogalmazása tartalmaz.
3. A kiválasztott következmény előidézése.

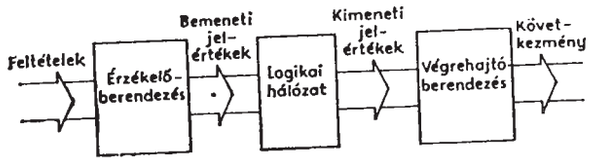
A tevékenység 2. fázisát *logikai döntésnek* is nevezik. Ez a döntés jelenti a *logikai feladat* megoldását. Általánosan fogalmazva tehát logikai feladatoknak azokat a feladatokat nevezzük, amelyeknek a megoldása során véges számú feltétel közül egyesek teljesüléséhez egyértelműen hozzá kell rendelni valamilyen előírás szerint egy-egy következményt, szintén véges számú lehetséges következmény közül válogatva.

1.2. A logikai hálózat fogalma

Miután a logikai feladat fogalmát tisztáztuk, vizsgáljuk meg, milyen eszközök szükségesek ahhoz, hogy az említett feltétel—következmény hozzárendelést emberi közreműködés nélkül el lehessen végezni. Az emberi tevékenység első része a mindenkori teljesülő feltételek tudomásulvétele. Az emberi közreműködést olyan

berendezéssel lehet pótolni, amely az egyes feltételek teljesülése esetén rendre további feldolgozásra alkalmas, egymástól különböző jelértékeket állít elő. Az emberi tevékenység második fázisát, a logikai döntést olyan berendezéssel lehet helyettesíteni, amely az egyes feltételek teljesülését jelző jelértékek hatására létrehozza azokat a jelértékeket, amelyek a feladat megfogalmazásban szereplő lehetséges következmények közül mindig annak felelnek meg, amelyet a megfogalmazás az éppen teljesülő feltételhez előír. Az ilyen berendezéseket nevezzük *logikai hálózatoknak*. Bár a továbbiakban csaknem kizárólag elektronikus elemekből felépített logikai hálózatokkal foglalkozunk, meg kell jegyeznünk, hogy a logikai feladatokat nemcsak villamos hálózatokkal lehet megoldani. Bizonyos speciális alkalmazásokban előfordulnak mechanikus, pneumatikus és hidraulikus elemekből álló logikai hálózatok is.

Az emberi tevékenység harmadik részét olyan berendezéseknek kell helyettesíteniük, amelyek a logikai hálózat által előállított — mindig valamilyen előírt következményt jelentő — jelértékek alapján az előírt következményt létrehozzák. Az 1.2. ábrán az eddig elmondottak alapján a feltétel—következmény hozzárendelést el-

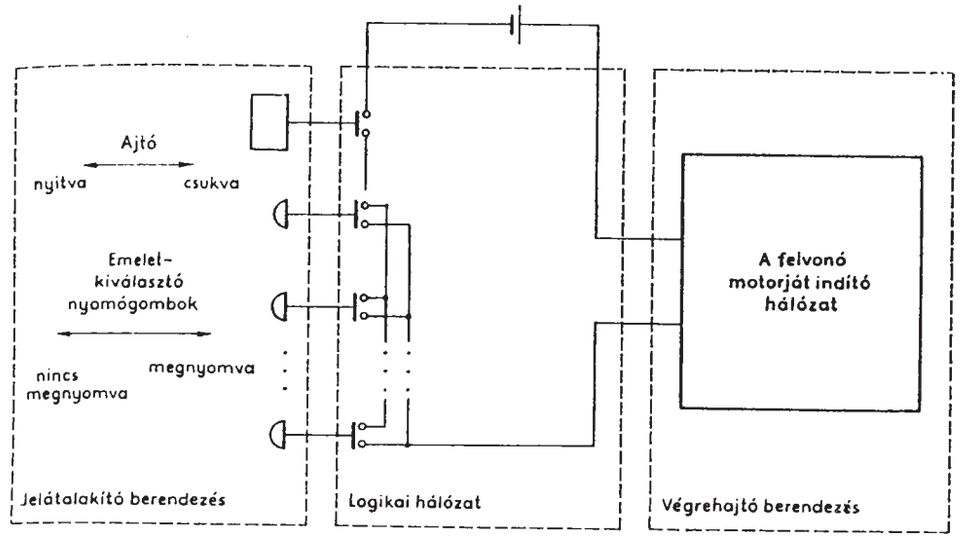


1.2. ábra. Feltétel—következmény hozzárendelés hatásvázlata

végző rendszer hatásvázlata látható. Vizsgáljuk meg, hogy a hatásvázlat szerinti felosztás miként képzelhető el a felvonó indítására vonatkozó példánkban. Az egyes feltételeket az ajtó- és az emeletkiválasztó nyomógombok együttes állapota jelenti. Az érzékelőberendezés mechanikus működtetésű érintkezőkből állhat, amelyek közül az egyik az ajtó becsukásának, a többi az emeletkiválasztó nyomógombok megnyomásának hatására zár egy-egy érintkezőt. A logikai hálózat bemeneti jelértékeit tehát az érintkezők nyitott, ill. zárt helyzetei alkotják. A logikai hálózat realizálása céljából a kétféle következményhez (motor áll, vagy megy), egy-egy kimeneti jelértéket kell hozzárendelni. A logikai hálózat a logikai feladat megfogalmazásának megfelelően a helyes kimeneti jelértéket előállítja. Ezután már csak arra van szükség, hogy a kimeneti jelérték létrehozza a hozzárendelt következményt. Példánkra vonatkozó kapcsolási vázlat az 1.3. ábrán látható. A logikai hálózatot jelen esetben azáltal alakíthattuk ki, hogy az ajtó helyzetét jelző érintkezőt sorba kapcsoltuk az egymással párhuzamosan kapcsolt emeletkiválasztó nyomógombok helyzetét jelző érintkezőkkel.

Mint említettük, a logikai hálózat bemenetén és kimenetén véges számú, egymástól különböző jelérték lehet jelen. Ezt általánosan úgy is fogalmazhatjuk, hogy a logikai hálózat véges számú bemeneti és kimeneti pontjainak mindegyikén véges számú, egymástól különböző értékű jel lehet.

Az érzékelőberendezés és a logikai hálózat a ma alkalmazott technológiákkal akkor valósítható meg a legegyszerűbben és legmegbízhatóbban, ha a bemeneti és kimeneti



1.3. ábra. Példa egy feltétel—következmény hozzárendelést megvalósító kapcsolásra

pontok mindegyikén csak két egymástól különböző jelértéket engedünk meg. A továbbiakban csak ilyen *kétértékű jelekkel* működő logikai hálózatokkal foglalkozunk. Ebben az esetben a bemeneti, ill. kimeneti értékeket azok a jelértékegyüttesek alkotják, amelyek egy adott helyzetben a bemeneti, ill. kimeneti pontokon fennállnak. A jelértékegyüttesek jellemzésére jól használható a kettes számrendszer, mivel minden bemeneti és kimeneti ponton csak két különböző jelérték léphet fel. Ha e két lehetséges érték megkülönböztetésére a 0 és 1 szimbólumokat használjuk, akkor egy jelértékegyüttes formálisan egy bináris számként adható meg. Ha pl. a bemeneti pontok száma n, akkor az egyes bemeneti pontok rendre egy n helyértékű bináris szám helyértékeinek felelnek meg, és így összesen 2^n számú, egymástól különböző bemeneti jelértékegyüttes képzelhető el, amelyek mindegyike a hálózat egy-egy bemeneti értékét alkotja. A 2^n számú bemeneti érték mindegyike kombinatorikailag ismétléses variáció útján származtatható, mert a két lehetséges jelértéket kell az n számú bemeneti ponton „elhelyezni" a sorrend figyelembevételével és ismétlődéseket megengedve. Így a bemeneti értéket bemeneti variációnak nevezhetnénk. A gyakorlatban azonban *bemeneti kombináció* elnvezés terjedt el, ezért a továbbiakban ezt használjuk, bár az elnevezés kombinatorikai szempontból helytelen. Az elmondottak természetesen értelemszerűen érvényesek a kimeneti pontokra, kimeneti értékekre és kimeneti variációkra, ill. kombinációkra.

Az így bevezetett elnevezésekkel tehát a logikai hálózatok működése úgy fogalmazható meg, hogy minden egyes bemeneti kombinációhoz előírt módon létrehoznak egy kimeneti kombinációt. Az előírást a logikai feladat valamilyen megfogalmazása

tartalmazza. Minden egyes bemeneti kombináció egy-egy feltétel teljesülését jelenti. A kimeneti kombinációk mindegyike pedig egy-egy következményt képvisel.

Természetesen léteznek olyan logikai feladatok is, amelyek nem mindegyik bemeneti kombinációhoz írják elő a kimeneti kombináció értékét. Ilyenkor tehát vannak olyan bemeneti kombinációk, amelyek fellépése esetén a logikai hálózat tetszőleges kimeneti kombinációt hozhat létre, mégis megoldja az előírt logikai feladatot. Ezek tehát olyan bemeneti kombinációk, amelyekhez előállított kimeneti kombináció értéke közömbös a logikai feladat megoldása szempontjából. Bár ilyenkor valójában a kimeneti kombináció értéke a közömbös, mégis az elterjedt szóhasználat szerint e közömbös kimeneti kombinációkat létrehozó bemeneti kombinációkat nevezik *közömbös* (don't care) *kombinációnak*. Bizonyos logikai feladatokban előfordulhatnak olyan feltételek is, amelyek a feladat jellegéből adódóan soha sem teljesülhetnek, ezek szintén közömbös kombinációk.

A közömbös kombinációkhoz tartozó kimeneti kombinációkat tervezés során tetszés szerint választhatjuk meg. Így véges számú olyan egymástól különböző logikai hálózatot állíthatunk elő, amelyek mindegyike megoldja az előírt logikai feladatot. A tervezés során fontos, hogy ezek közül a hálózatok közül mindig azt válasszuk, amely valamilyen adott szempontból a legkedvezőbb. A közömbös kombinációknak az ilyenfajta ún. helyes rögzítése tehát a tervezés során megoldandó feladat. A közömbös kombinációkat tartalmazó logikai feladatokat *nem teljesen határozott* vagy *nem teljesen specifikált* logikai feladatoknak nevezzük.

Meg kell jegyeznünk, hogy gyakran olyankor is logikai hálózatot alkalmaznak, amikor a bemeneti és kimeneti jelértékek száma egy tartományon belül végtelen, vagyis analóg jeleken végzett műveletekről van szó. Ebben az esetben a jelátalakító berendezés a bemeneti analóg jelből létrehozza a bemeneti kombinációt, vagyis az ún. *digitális jelet*, természetesen véges számú érték szerint kvantálva. Az érzékelőberendezés tehát ilyenkor az analóg—digitál átalakítást végzi. A végrehajtó-berendezés ebben az esetben a logikai hálózat mindenkori kimeneti kombinációja alapján, vagyis digitális jel alapján létrehozza az analóg kimeneti jelet, amely természetesen csak véges számú pontban pontos értékű. A végrehajtó-berendezés tehát ilyenkor a digitál—analóg átalakítást végzi. Ez a megoldás nyilvánvalóan akkor előnyös, ha az analóg jeleken elvégzendő műveletek annyira bonyolultak, hogy az analóg—digitál és a digitál—analóg átalakítás szükségessége ellenére a műveletek pontosabban, egyszerűbben és olcsóbban végezhetők el logikai feladat formájában, logikai hálózattal.

A továbbiakban kizárólag a logikai hálózatok tervezésével foglalkozunk, és csakis akkor térünk ki az érzékelő- és végrehajtó-berendezések felépítésére, ha az a tervezési lépéseket befolyásolja.

1.3. A logikai hálózatok csoportosítása a megoldandó logikai feladatok szerint

Az eddigiek alapján úgy tűnhet, hogy bármilyen logikai feladat megoldása céljából elegendő, ha a logikai hálózat csupán a mindenkor fennálló pillanatnyi bemeneti kombinációt veszi figyelembe a kimeneti kombináció előállításához. Hogy ez mennyire nincs így, az világosan látszik a következő logikai feladatból.

A felvonó csak akkor induljon el a harmadik emeletre, ha az ajtaja be van csukva, és a fülkében levő emeletkiválasztó nyomógombok közül a harmadik emeletre vonatkozó be van nyomva.

A feladat szövegében — burkoltan ugyan — három lehetséges következmény szerepel, feltéve, hogy az emeletek száma háromnál több:

— a felvonó nem indul el,
— a felvonó elindul a harmadik emeletre felfelé,
— a felvonó elindul a harmadik emeletre lefelé.

Az utóbbi két következményt nyilvánvalóan meg kell különböztetni. Ha a feltételeket és a teljesülésükhöz előírt következményeket az 1.1. ábrához hasonlóan ebben az esetben is táblázatosan kísérelnénk meg összefoglalni, akkor az ott felsorolt négy feltételt tudnánk ebben a példában is megkülönböztetni. Az emeletkiválasztó nyomógomb rovatba természetesen jelen esetben a harmadik emeletet kiválasztó nyomógomb helyzetei kerülnének. Nyilvánvaló, hogy a feladat megfogalmazásában megkülönböztethető feltételek alapján nem tudjuk egyértelműen kiválasztani valamelyiket a három következmény közül. Ehhez ugyanis szükség van a felvonó mindenkori helyzetének ismeretére, amelyre vonatkozóan a feladat megfogalmazása nem tartalmaz feltételt.

Az 1.3. ábrához hasonlóan jelen példánkban is a felvonó motorját indító hálózatot képzelhetjük el végrehajtóhálózatként. Ebben az esetben a három következményt rendre a motorindító hálózat bemenetére jutó feszültség hiánya, ill. jelenlétekor annak előjele jelképezheti. Ezek egyértelmű kiválasztásához tehát a logikai hálózatnak szüksége van a felvonó mindenkori helyzetére vonatkozó pótlólagos, ún. *másodlagos (szekunder) feltételekre*. Ezeket pl. az egyes emeleteken elhelyezett érzékelők szolgáltathatják, amelyek egy-egy érintkezőt zárnak, ha éppen az illető emeleten tartózkodik a felvonó. Ilyenkor tehát a logikai hálózat az előírt következményt, vagyis kimeneti kombinációt nem képes kizárólag a feladatban szereplő feltételeknek megfelelő, éppen jelenlevő bemeneti kombinációk alapján előállítani. Ehhez szüksége van a szekunder feltételeknek megfelelő *szekunder kombinációkra* is.

Mint példánkból is látszik, a logikai hálózat képes arra, hogy működése során megváltoztassa a szekunder feltételeket, vagyis a szekunder kombinációkat. Ha ugyanis a megindított felvonó egy másik emeletre ér, azonnal új szekunder feltétel teljesül, vagyis megváltozik a szekunder kombináció. Ezt úgy is fogalmazhatjuk, hogy ez az ilyen fajta logikai hálózat a működése során a szekunder kombinációt úgy változtat-

ja, hogy a bemeneti kombináció és a szekunder kombináció együttes pillanatnyi értéke alapján mindig elő tudja állítani a feladatban előírt kimeneti kombinációt. Az ilyen logikai hálózatok mindig zárt hatásláncot, jelterjedési visszacsatolást tartalmaznak. A szekunder kombinációk segítségével tehát az ilyen típusú logikai hálózatok képessé válnak arra, hogy ugyanahhoz a bemeneti kombinációhoz más-más kimeneti kombinációt szolgáltassanak attól függően, hogy a bemeneti kombináció fellépésekor milyen értékű a szekunder kombináció. A szekunder kombináció pillanatnyi értékét viszont a logikai hálózatra jutott korábbi bemeneti kombinációk és azok sorrendje is befolyásolja, mivel a szekunder kombinációkat a logikai hálózat a működés során változtatja. Ezért az ilyen típusú logikai hálózatok működését úgy is jellemezhetjük, hogy a mindenkori kimeneti kombinációt nemcsak a pillanatnyi bemeneti kombináció, hanem a korábban fennállt bemeneti kombinációk és azok sorrendje is befolyásolja. Emiatt nevezik ezeket a hálózatokat *sorrendi (szekvenciális) logikai hálózatoknak.*

Ha a logikai feladat megoldásához nem szükséges, hogy a logikai hálózatnak a fenti értelemben vett emlékezete legyen, akkor a mindenkori kimeneti kombináció létrehozásához elegendő a bemeneti kombináció pillanatnyi értéke. Ezeket a hálózatokat *kombinációs logikai hálózatoknak* nevezik. A kombinációs hálózatok természetesen nem képesek arra, hogy ugyanahhoz a bemeneti kombinációhoz más-más kimeneti kombinációt szolgáltassanak, hiszen minden bemeneti kombináció egyértelműen és kizárólagosan határozza meg a kimeneti kombinációt. A kimeneti kombinációból viszont kombinációs hálózatok esetén sem tudjuk általában egyértelműen meghatározni az azt előidéző bemeneti kombinációt, mert nem követelmény, hogy különböző bemeneti kombinációk minden esetben más-más kimeneti kombinációt hozzanak létre.

Összefoglalva megállapíthatjuk, hogy a megoldandó logikai feladattól függően kell kombinációs vagy sorrendi logikai hálózatot terveznünk.

1.4. A logikai hálózat és a logikai rendszer kapcsolata

Bonyolultabb logikai feladatok esetén nem célszerű egyetlen, ugyancsak bonyolult logikai hálózatot tervezni. Előnyösebb, ha ilyenkor a bonyolult logikai feladatot több, de önmagukban viszonylag egyszerű logikai hálózat létrehozásával, azok összehangolt működtetése révén oldjuk meg. A logikai hálózatoknak az így kialakuló rendszere alapján bevezethetjük a *logikai rendszer* fogalmát.

A logikai hálózatoknak egy adott feladat megoldása céljából együttműködő összességét nevezzük *logikai rendszernek.* A logikai hálózatokat készen kapható elemi logikai hálózatokból, ún. *építőelemekből* állítjuk össze. Ilyen értelemben egy logikai

rendszer logikai hálózatai maguk is logikai rendszereknek tekinthetők, amelyek logikai hálózatai az építőelemek. A megkülönböztetésnek azért van jelentősége, mert általában eltérő módszereket kell követnünk a logikai hálózatoknak az építőelemekből történő felépítésekor és a logikai rendszernek a logikai hálózatokból történő létrehozásakor. Ugyanakkor a technológiai fejlődés során egyre bonyolultabb logikai feladatok megoldására képes építőelemek jönnek lére, miáltal egy-egy építőelem önállóan is képessé válhat a logikai rendszer bizonyos feladatait megoldó logikai hálózat helyettesítésére.

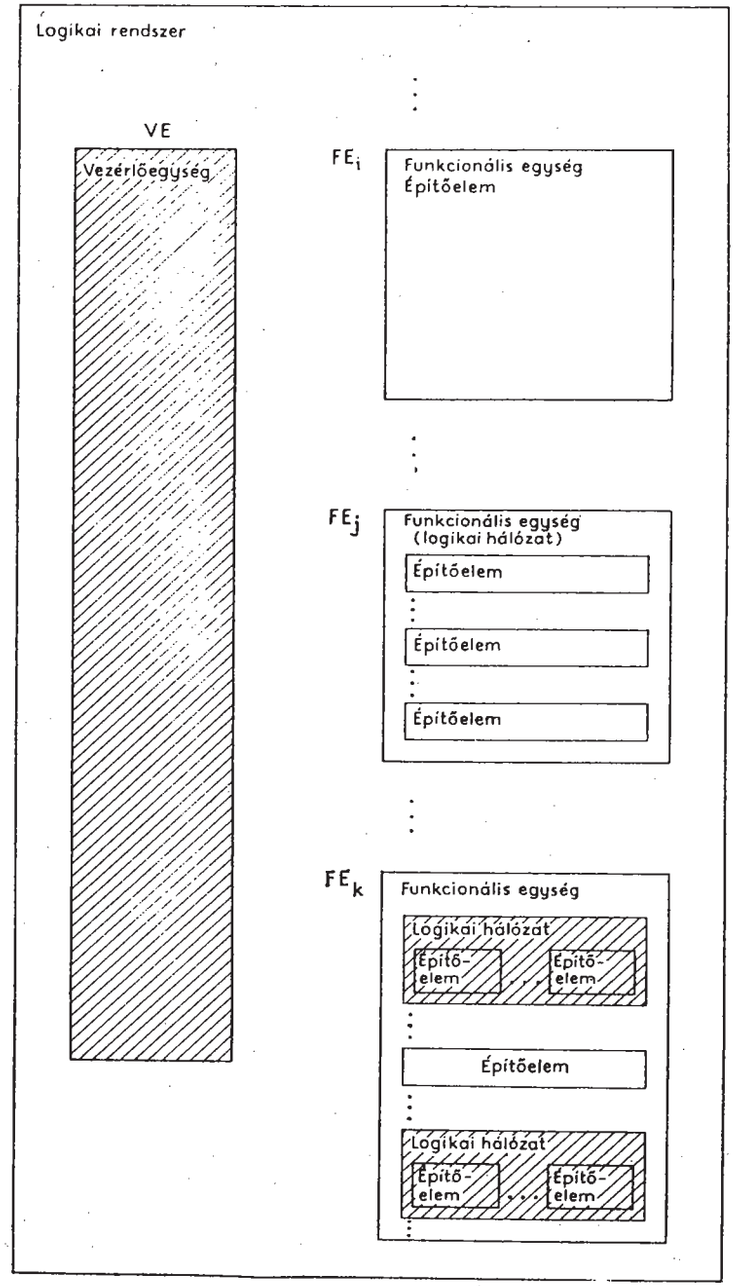
Megállapíthatjuk tehát, hogy a rendszer és a hálózat megkülönböztetését, valamint a tervezésükhöz alkalmazandó módszereket döntően befolyásolhatja a rendelkezésre álló építőelem-készlet fejlettsége.

A logikai rendszer által megoldandó feladat legtöbbször egyszerűen bontható fel részfeladatokra. Könnyen érthető az a törekvés, hogy először ezeket a részfeladatokat oldjuk meg külön-külön egy-egy részegységgel. Ezeket a részegységeket *funkcionális egységeknek* nevezik és nyilvánvalóan önmagukban egyszerűbb logikai rendszereknek tekinthetők, mint amilyen logikai rendszerhez jutnánk a részfeladatokra való felbontás nélkül. A funkcionális egységeket külön-külön lehet tervezni és megvalósítani, azonban együttműködésük megfelelő vezérléséhez létre kell hoznunk egy pótlólagos funkcionális egységet, az ún. *vezérlőegységet.*

A funkcionális egységek belső felépítése az általuk megoldandó logikai feladat bonyolultságától és a rendelkezésre álló építőelem-készlet fejlettségétől függ. Az 1.4. ábrán vázolt felépítésben az FE_i jelű funkcionális egységet például egyetlen építőelem alkotja. Az FE_i funkcionális egység által megoldandó logikai feladat tehát az építőelem-készlet fejlettségehez képest annyira egyszerű, hogy található olyan építőelem, amely önmagában megoldja ezt a feladatot.

Az FE_j jelű funkcionális egységről azt tételeztük fel, hogy több építőelem alkotja ugyan, de a szükséges építőelemek kiválasztását és azok összekapcsolását a funkcionális egység által megoldandó logikai feladat alapján az építőelem-készlet ismeretében mindenféle tervezési eljárás nélkül meg tudjuk adni. Ezt ezúttal is az építőelem-készletnek a megoldandó logikai feladathoz képesti viszonylagos fejlettsége teszi lehetővé. Az FE_j funkcionális egység tehát adott építőelemek összekapcsolása révén valósul meg, így logikai hálózatnak is nevezhetjük.

Az FE_k jelű funkcionális egység által megoldandó logikai feladatot az építőelem-készlet fejlettségéhez képest olyan bonyolultnak tételeztük fel, hogy a szükséges építőelemeket és azok összekapcsolását nem tudjuk meghatározni az FE_j funkcionális egység esetében lehetséges módon. Ilyenkor az FE_k funkcionális egység által megoldandó feladathoz képest viszonylag fejletlen építőelemekből először fel kell építeni olyan logikai hálózatokat, amelyeket a továbbiakban, mint ún. *látszólagos építőelemeket* használhatunk és segítségükkel már lehetővé válik az FE_j funkcionális egység esetében feltételezett megoldás. Természetesen a látszólagos építőelemeket alkotó logikai hálózatok mellett valóságos építőelemeket is felhasználhatunk a felépítés során.



1.4. ábra. Logikai rendszerek felépítése

Az FE$_k$ funkcionális egység esetében feltételezett helyzetben tehát az építőelemek viszonylagos fejletlenségéről beszélhetünk. Ilyenkor a látszólagos építőelemeknek tekintett logikai hálózatok felépítését általában nem célszerű a valóságos építőelemek intuitív összekapcsolásával végezni, hanem ajánlatos szisztematikus tervezési módszereket alkalmazni.

A funkcionális egység által megoldandó logikai feladat az adott építőelem-készlet fejlettségéhez képest lehet annyira bonyolult, hogy a fenti háromféle (FE$_i$, FE$_j$, FE$_k$) felépítési mód egyikét sem tudjuk alkalmazni. Ilyenkor még az FE$_k$ esetében említett látszólagos építőelemeknek tekinthető logikai hálózatokat sem tudjuk úgy definiálni, hogy lehetővé váljon az FE$_j$ funkcionális egységre feltételezett felépítési mód. Az ilyen funkcionális egységet ezért logikai rendszernek tekintve tovább kell bontani belső funkcionális egységekre és belső vezérlőegységre. Az így kialakuló belső funkcionális egységeket meg kell vizsgálni, abból a szempontból, hogy az 1.4. ábrán vázolt háromféle felépítés valamelyike alkalmazható-e. Ha valamelyik belső funkcionális egységre nem találunk így megoldást, akkor a felbontási folyamatot tovább kell folytatnunk mindaddig, amíg csak olyan funkcionális egységek nem keletkeznek, amelyek felépítése már nem igényel belső vezérlőegységet, vagyis alkalmazható rájuk az 1.4. ábrán szereplő megoldások valamelyike.

A logikai rendszer által megoldandó logikai feladat bonyolultságától és a rendelkezésre álló építőelem-készlet fejlettségétől függően tehát az 1.4. ábra szerinti felépítést több, ún. egymásba ágyazott *vezérlési szinten* is alkalmazhatjuk.

A vezérlőegységet mint speciális funkcionális egységet szintén az FE$_i$, FE$_j$, FE$_k$ egységek valamelyikéhez hasonló felépítésben képzelhetjük el. A vezérlőegység által megoldandó logikai feladat különleges sajátosságai lehetővé teszik szisztematikus tervezési eljárások alkalmazását. Az 1.4. ábrán vonalkázással jelöltük azokat az egységeket, amelyek szisztematikus tervezési eljárással építhetők fel. Belátható, hogy a logikai rendszer felbontása, valamint az FE$_j$ és az FE$_k$ funkcionális egységek összeállítása építőelemekből, ill. logikai hálózatokból alapvetően intuitív feladat. Tervezési eljárásokat tehát csak a látszólagos építőelemekként használható logikai hálózatok és a vezérlőegység megvalósítására fogunk megismerni.

Az 1.4. ábrán vázolt felépítés nyilvánvalóan elég általános ahhoz, hogy alkalmazható legyen például akár egyetlen nagybonyolultságú építőelem belső funkcionális működésének leírásához. Ilyenkor természetesen rögzítenünk kell, hogy mit tekintünk az építőelemen belül építőelemnek és ettől függően kell különválasztani a belső funkcionális egységeket és a belső vezérlőegységet.

A logikai rendszereknek az 1.4. ábrán vázolt felépítésével szemléltethetjük az építőelem-készlet fejlődését is. Minőségi változásnak tekinthetjük ugyanis az építőelem-készlet fejlődésében azoknak a fokozatoknak az elérését, amelyektől kezdve az FE$_k$-hoz hasonló funkcionális egységeket az FE$_j$ szerinti, vagy az FE$_j$-hez hasonlóakat az FE$_i$ szerinti megoldással építhetjük fel. Nyilvánvalóan további minőségi változás az, amikor a teljes logikai rendszert már egyetlen építőelemmel megvalósíthatjuk. Ilyen építőelem-készletből természetesen a korábbinál jóval bonyolultabb

logikai feladat megoldására alkalmas logikai rendszert építhetünk fel az 1.4. ábra vázlata szerint.

Azt mondhatjuk tehát, hogy az építőelem-készlet fejlődésében akkor következnek be minőségi változások, azaz ún. *generációváltozások*, amikor az addig látszólagos építőelemként felépített logikai hálózatok valóságos építőelemként beszerezhetőkké válnak, vagy az addig több építőelemből álló funkcionális egységek egyetlen építőelemmel megvalósíthatók lesznek, vagy az addig több funkcionális egységből felépülő logikai rendszerek helyettesíthetőkké válnak egyetlen építőelemmel.

Ha egy tervezendő logikai rendszer esetében meghatároztuk az építőelem-készletet, továbbá kijelöltük a logikai hálózatokat és a funkcionális egységeket, valamint definiáltuk működésüket, akkor kialakítottuk az ún. *rendszertechnikai felépítést*. Mint az eddigiekből is kiderült, a rendszertechnikai felépítést alapvetően befolyásolja az alkalmazandó építőelem-készlet fejlettsége.

1.5. A logikai tervezés célja

A logikai rendszerek tervezésében az előző fejezet alapján az alábbi lépéseket különböztethetjük meg:

1. A megoldandó feladat alapján az alkalmazható építőelem-készlet ismeretében a rendszertechnikai felépítés kialakítása.
2. A funkcionális egységek működésének valamilyen alkalmas leírása alapján azok belső rendszertechnikai felépítésének kialakítása.
3. A funkcionális egységeken belül kijelölt logikai hálózatok tervezése.

Az 1. lépésben természetesen meg kell adni egyértelműen az egyes funkcionális egységeknek egymással és a logikai rendszer környezetével való kapcsolatait, a kapcsolatokat megvalósító jeleket és jelfolyamatokat. Az 1.4. ábrán ezek a kapcsolatok a könnyebb áttekinthetőség érdekében nincsenek feltüntetve. Ugyancsak egyértelműen meg kell adni a 2. lépésben a funkcionális egységen belüli kapcsolatokat vagy a logikai hálózatok, vagy az építőelemek között attól függően, hogy az 1.4. ábrán vázoltak közül melyik felépítési mód vonatkozik a funkcionális egységre. A 3. lépésben elvégzendő feladatokról csak egy adott építőelem-készlet ismeretében érdemes beszélni.

A felsorolt három tervezési lépés végrehajtása után a logikai rendszer ún. *logikai kapcsolási tervéhez* jutunk, amely egyértelműen megadja a rendszerben felhasznált építőelemek közötti összeköttetéseket. A továbbiakban a logikai tervezés végeredményének a logikai kapcsolási tervet tekintjük. A logikai rendszer megvalósításának, megépítésének, gyártási dokumentációjának ez a logikai kapcsolási terv a kiindulási alapja. Ugyancsak a logikai kapcsolási terv adja a legfőbb támpontot a megépített logikai rendszer, az *ún. logikai berendezés* vagy más szóval *digitális berendezés* be-

méréséhez, vizsgálatához (teszteléséhez) és a hibakeresési algoritmusok kidolgozásához. A gyárthatóság és a könnyű bemérhetőség szempontjai sok esetben visszahatnak a logikai tervezés lépéseire és már a logikai tervezéskor tekintettel kell lennünk ezekre a követelményekre is.

Láttuk, hogy a logikai tervezési lépések mindegyikét alapvetően befolyásolja a rendelkezésre álló építőelem-készlet. Különösen érvényes ez a 3. lépésre, amely a funkcionális egységeken belül elkülöníthető logikai hálózatok létrehozására szolgál. Ezek a logikai hálózatok viszonylag egyszerű építőelem-készletből felépülve ugyanazt a rendszertechnikai szerepet tölthetik be, mint a bonyolultabb építőelemek, amelyek az adott készletben nem állnak rendelkezésre, ezért a logikai tervezés során kell azokat logikai hálózatként felépíteni meglevő egyszerűbb építőelemekből.

A továbbiakban feltételezzük, hogy a felhasználható építőelem-készlet valamilyen integráltsági fokú integrált áramköri elemekből áll. Ezen belül is először kis és közepes integráltsági fokú elemkészletből indulunk ki. Ebben az esetben az építőelemek önmagukban csupán viszonylag egyszerű logikai feladatok megoldására képesek. Ezért a 1. és 2. tervezési lépések végrehajtása során könnyebb, ha a rendszertechnikai felépítést nem építőelemekig, hanem csupán az azokból kialakítandó logikai hálózatokra bontjuk le. Így valójában az egyszerű építőelem-készletből a megfelelő logikai hálózatok tervezése útján egy látszólagos építőelem-készletet hozhatunk létre, amelynek elemei között a rendszertechnikai felépítéskor definiált logikai hálózatok is szerepelnek. Az 1. és 2. tervezési lépéseket tehát mindenképpen megkönnyíti, ha először azokkal a 3. lépésként említett tervezési módszerekkel ismerkedünk meg, amelyek segítségével kis és közepes integráltsági fokú integrált áramköri építőelemekből lehet logikai hálózatokat tervezni. A tervezés végeredményének ez esetben is a logikai kapcsolási tervet tekintjük, amely a felhasznált építőelemek összekapcsolásának egyértelmű megadása. A következő fejezetekben ismertetendő módszerekről előre bocsátva meg kell jegyezni, hogy alapelveiket abban az időben dolgozták ki, amikor még nem léteztek integrált áramköri építőelemek, és reléérintkezőket, ill. később elektroncsöves, diódás, tranzisztoros alapáramköröket kellett építőelemként alkalmazni. Az ekkor kidolgozott tervezési eljárások legfontosabb alapelvei és végrehajtási algoritmusaik azonban átültethetőknek bizonyultak a kis és közepes integráltsági fokú integrált áramköri építőelemek esetére is. Arra törekszünk, hogy olyan módszereket találjunk, amelyek lehetőleg kevés intuitív — tehát találékonyságot, próbálgatást igénylő — lépéssel vezetnek el a valamilyen szempontból legkedvezőbb logikai hálózathoz. Az ilyen módszerek szisztematikus jellegük következtében lehetővé teszik, hogy a tervezéskor előnyösen alkalmazzunk digitális számítógépet. Az alkalmazott építőelemektől függően a tervezési módszerek természetesen módosulnak, és nem minden építőelem-készlet esetén fogalmazhatók meg kezelhető módon a legkedvezőbb hálózat által kielégítendő feltételek.

2. Kombinációs hálózatok tervezése

Ebben a fejezetben először a kombinációs hálózatok működésének formális leírásával, a logikai függvények fogalmával és tulajdonságaival foglalkozunk. Ezután megismerkedünk a kombinációs hálózatok gazdaságos megvalósításának egyik legfontosabb eljárásával: a logikai függvények minimalizálásával. A kombinációs hálózatok működését befolyásolhatják a valóságos hálózatokban mindig meglevő jelkésleltető hatások. Ezek a hálózat tranziens viselkedését szabják meg és sok esetben hibás átmeneti kimeneti kombinációkat okoznak. A fejezetben bemutatjuk, hogy az ilyen okokra visszavezethető, többnyire véletlenszerű hibákat hogyan lehet csoportosítani és elkerülni, lehetőleg már a tervezés során. Bevezetjük a kétszintű, valamint a több szintű hálózatok fogalmát, és bemutatjuk a több szintű kombinációs hálózatok tervezésének néhány alapgondolatát. A fejezet végén vázoljuk azokat a legfontosabb szempontokat, amelyek a nagy integráltsági fokú építőelemek (memória, PLA) alkalmazásakor merülnek fel a kombinációs hálózatok felépítése során.

2.1. Kétértékű jelekkel végzett logikai műveletek algebrai leírása

2.1.1. A logikai érték fogalma

Az előzőekben a logikai hálózatok bemeneti és kimeneti pontjain lehetséges két különböző jelértéket 1-gyel és 0-val jelöltük. A gyakorlati megvalósításokban ezeket a jelértékeket a legkülönbözőbb fizikai paraméterértékek képviselhetik (pl. feszültségérték, áramérték stb.). Tulajdonképpen megállapodás kérdése, hogy a fizikai paramétereknek milyen értékét tekintjük 1-nek, ill. 0-nak és legtöbbször az egyszerű kivitelezhetőség a fő szempont. Lényeges, hogy két jól megkülönböztethető fizikai paraméterértéket vagy -tartományt válasszunk erre a célra. Könnyen belátható, hogy ezektől eltérő bármilyen paraméterértéknek semmiféle jelentése sincs a logikai feladat szempontjából. Ebből következik, hogy értelmetlenség ezekkel a fizikai paraméterértékekkel pl. bármiféle aritmetikai műveletet (pl. összeadást, kivonást stb.) végezni, hiszen olyan paraméterértéket kaphatunk eredményül, amelyeknek a logikai

feladat szempontjából nincs jelentésük. A két lehetséges jelértéket ezért elvonatkoztatva a fizikai paraméterértékektől *logikai értéknek* nevezzük. Pl. a logikai 0-nak 0 V és 0,8 V közötti, a logikai 1-nek 2 V és 5 V közötti feszültség felel meg. Más, az adott tűrési tartományon kívül eső feszültségnek logikai értéke *nincs*, az a logikai hálózatban üzemszerű, állandósult állapotban nem léphet fel.

A logikai műveletekben a logikai érték hasonló az aritmetikai műveletek számértékeihez. Míg azonban az utóbbiakban a számértékek végtelen sokasága létezik (pozitív, negatív, egész, tört, irracionális stb. számok), addig a logikai műveletekben csak véges számú, a kétértékű logikai műveletekben csupán kettő, melyeket a 0 és 1 szimbólumokkal szokás jelölni. Hangsúlyozzuk azonban, hogy ezeknek semmi közük az aritmetikai műveletek 0 és 1 számértékeihez, ezért vezettük be a logikai értékek elnevezést.

2.1.2. A logikai alapműveletek definíciója

Említettük, hogy a logikai értékek között nincsenek értelmezve az aritmetikai műveletek. Ezzel szemben a logikai értékek között definiálhatók ún. *logikai műveletek*. Tekintettel arra, hogy összesen csak két különböző logikai értéket értelmeztünk a logikai hálózatokban, az egyes logikai műveletek definícióit egyszerűen meg tudjuk adni. Nem kell ugyanis mást tennünk, mint az egyes műveletek eredményét megadni a hennük részt vevő összes lehetséges logikai értékre. A továbbiakban ilyen módon adjuk meg a logikai alapműveletek definícióját.

Logikai szorzás (konjunkció), ÉS kapcsolat, jele: $\cdot$

$$0 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$1 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

A művelet eredménye tehát csak akkor a logikai 1 érték, ha az első *és* a második tényező értéke egyaránt logikai 1.

A művelet felsorolt definíciós összefüggéseiből látszik, hogy a művelet kommutatív, vagyis az operandusok sorrendje felcserélhető.

Megfigyelhetjük, hogy a definíciós egyenletek formailag teljesen megegyeznek az aritmetikai szorzás szabályaival, jóllehet az 1 és 0 értékeknek csak logikai jelentésük van.

Logikai összeadás (diszjunkció), VAGY kapcsolat, jele: $+$

$0+0=0$

$0+1=1$

$1+0=1$

$1+1=1$

A művelet eredménye tehát csak akkor a logikai 1 érték, ha *vagy* az első, *vagy* a második tag (*vagy* mindkettő) értéke logikai 1.

Az utolsó definíciós összefüggés kivételével formailag az aritmetikai összeadás szabályai is alkalmazhatók a logikai értékekre.

A művelet felsorolt definíciós összefüggéseiből látszik, hogy a művelet szintén kommutatív.

Mindkét definiált logikai művelet értelemszerűen kiterjeszthető kettőnél több tényezőre, ill. tagra is. Ilyenkor természetesen a műveletek elvégzésének sorrendjét megfelelő zárójelek alkalmazásával jelölnünk kell, akárcsak aritmetikai műveletek esetén.

Tagadás (invertálás, negálás, komplementálás, ellentettképzés), jele: $^{-}$

$\bar{0}=1$

$\bar{1}=0$

Mivel csak két logikai értéket értelmeztünk:

$\bar{\bar{0}}=0$

$\bar{\bar{1}}=1$

A művelet tehát logikai értékekhez ellentettjüket rendeli hozzá. A műveletet páros számszor alkalmazva, eredményül a kiindulási logikai érték adódik. (Páratlan számú alkalmazás természetesen az ellentett, negált értéket eredményezi.)

2.1.3. A logikai változók és a logikai algebrai kifejezés fogalma

Miként az aritmetikai kifejezésekben a betűjelek *aritmetikai* változókat jelentenek, melyek általában tetszés szerinti számértékeket vehetnek fel, a logikai kifejezésekben használt betűjelek *logikai változók,* melyek logikai értékek helyett állnak. Kétértékű logikai műveletek esetén egy logikai változó vagy 0, vagy 1 értéket vehet fel.

Így a logikai hálózatok bemeneti és kimeneti pontjainak mindegyike egy-egy logikai változóval jelölhető. A logikai változókon és a konstansoknak tekinthető logikai

értékeken értelmezve az előzőekben definiált logikai műveleteket, az alábbi azonosságokhoz jutunk:

$A\cdot 0 \equiv 0 \qquad A+0 \equiv A$

$A\cdot 1 \equiv A \qquad A+1 \equiv 1$

$A\cdot \bar{A} \equiv 0 \qquad A+\bar{A} \equiv 1$

$\bar{\bar{A}} \equiv A$

$A+B \equiv B+A$

$A\cdot B \equiv B\cdot A$

$\overline{A+B} \equiv \bar{A}\cdot\bar{B}$

$\overline{A\cdot B} \equiv \bar{A}+\bar{B}$

A felsorolt azonosságok érvényességéről könnyen meggyőződhetünk, ha a logikai változók helyére az összes lehetséges módon behelyettesítjük a logikai értékeket. Pl. az $A+1=1$ igazságát az bizonyítja, hogy ha A értéke akár 0, akár 1, a logikai összeg 1:

$0+1=1,$ ill. $1+1=1,$

A két utolsó azonosságot *De Morgan*-azonosságnak nevezik. Általánosan is megfogalmazható, hiszen tetszőleges számú tagra és tényezőre is érvényes. Eszerint egy logikai összeg negáltja azonos a tagok negáltjainak logikai szorzatával, egy logikai szorzat negáltja pedig azonos a tényezők negáltjainak logikai összegével. Ez a kapcsolat a logikai műveletek közötti ún. *dualitást* tükrözi.

Könnyen belátható, hogy a definiált logikai műveletek — a korábban már említett kommutatív tulajdonságon kívül — asszociatív és disztributív tulajdonságúak. Ennek igazolásához elegendő ugyanis a logikai értékeket az összes lehetséges módon behelyettesíteni a logikai változók helyére az alábbi azonosságokban:

$A+(B+C) \equiv A+B+C$

$A\cdot(B\cdot C) \equiv A\cdot B\cdot C$

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

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

A zárójelek természetesen a műveletek elvégzésének sorrendjére utalnak.

Megállapíthatjuk, hogy az utolsó előtti azonosság kivételével formálisan az aritmetikai műveletek szabályai szerint is a felírt azonosságokhoz jutunk.

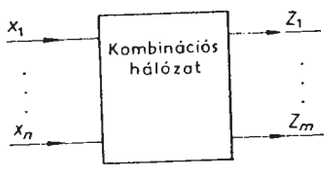
A logikai értékek — mint konstansok —, a definiált logikai alapműveletek, és a logikai változók összessége matematikai értelemben *logikai algebrának* vagy *Boole-algebrának* nevezhető. Így a felírt azonosságok *logikai algebrai kifejezések*

Megjegyezzük, hogy a továbbiakban a logikai szorzás jelét az egyszerűbb írásmód

érdekében nem minden esetben írjuk le, így a műveleti jel nélkül egymás után következő változókat mindig logikai ÉS kapcsolatban levőknek kell tekinteni. Az egyszerűbb fogalmazás érdekében azokat a változókat, amelyeken nincs kijelölve a tagadás (negálás) művelete, *ponált változóknak* nevezik. Tehát A ponált, $\bar{A}$ pedig negált változó.

2.1.4. Adott számú bemenet és kimenet esetén megoldható logikai feladatok száma

Vizsgáljuk meg, hogy n számú bemeneti pont és m számú kimeneti pont esetén (2 1. ábra) összesen hány egymástól különböző logikai feladat megoldására lehet képes egy kombinációs hálózat. Jelöljük e célból a lehetséges bemeneti kombiná-



2.1. ábra. Kombinációs hálózat összefoglaló ábrázolása

ciók számát N_b-vel, a lehetséges kimeneti kombinációk számát N_k-val. Az előzőek alapján:

$$N_b = 2^n; \qquad N_k = 2^m.$$

Az összes lehetséges, teljesen határozott logikai feladat számának meghatározása céljából ki kell számolnunk, hogy az összes lehetséges kimeneti kombináció közül hányféle — egymástól eltérő — módon lehet minden egyes bemeneti kombinációhoz egyet-egyet hozzárendelni. Más szóval azt kell meghatároznunk, hogy N_k számú elemből hányféleképpen tudunk N_b számú elemet kiválasztani ismétlődéseket is megengedve és a különböző sorrendű kiválasztásokat megkülönböztetve. Ez kombinatorikailag ismétléses variációt jelent, így a hálózattal megoldható összes lehetséges egymástól különböző, teljesen határozott logikai feladatok száma:

$$N_f = N_k^{N_b} = (2^m)^{2^n}.$$

Ezek után könnyen meghatározhatjuk a hálózattal megoldható, nem teljesen határozott logikai feladatok számát is. Jelöljük e célból d-vel a közömbös bemeneti kombinációk számát $(0 \leqq d < 2^n)$. Egy teljesen határozott logikai feladatból ekkor annyi nem teljesen határozott feladatot tudunk képezni, ahányféleképpen a 2^n számú bemeneti kombinációból d számút közömbössé tudunk tenni. Ez kombinatorikailag ismétlés nélküli kombináció, így az összes nem teljesen határozott logikai feladat száma $\binom{2^n}{d}$-szerese a teljesen határozott logikai feladatok számának.

2.2. Logikai függvények

2.2.1. A logikai függvény fogalma

Mint azt a korábbiakban megállapítottuk, a kombinációs logikai hálózatok kimeneti kombinációja egyértelműen meghatározható a mindenkori bemeneti kombináció ismeretében. Ez azt jelenti, hogy minden egyes kimeneti ponton levő jelérték attól függ, hogy egy adott időpontban a bemeneti pontokra milyen értékű jelek jutnak. Ha a bemeneti és kimeneti pontokat a korábbiak szerint logikai változóknak tekintjük, akkor ezt a következőképpen fogalmazhatjuk meg.

Minden egyes kimeneti pontnak megfelelő logikai változó (a továbbiakban kimeneti változó) értéke attól függ, hogy milyen értékűek a bemeneti pontoknak megfelelő logikai változók (a továbbiakban bemeneti változók).

Így tehát minden egyes kimeneti változó értéke egy függvénykapcsolat alapján határozható meg, amelyben a függő változó a kimeneti változó, a független változók pedig a bemeneti változók. Ha ismerjük a kombinációs hálózat minden egyes kimeneti változójára vonatkozólag ezt a függvénykapcsolatot, akkor ismerjük azt a logikai feladatot, amelyet a hálózat megold. Az egyes függvénykapcsolatok ismeretében ugyanis minden egyes bemeneti kombinációhoz meghatározható az összes kimeneti változó értéke, vagyis a kimeneti kombináció.

Ezeket a függvénykapcsolatokat nevezik *logikai függvényeknek*. A logikai függvényekre tehát az jellemző, hogy mind a függő változójuk, mind a független változóik logikai változók.

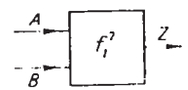
Ha egy kombinációs hálózat n számú bemeneti ponttal és m számú kimeneti ponttal rendelkezik, akkor az általa megoldott logikai feladat m számú n változós logikai függvénykapcsolattal adható meg.

A logikai függvények segítségével tehát egyértelműen leírható a kombinációs hálózatok működése. Ezért célszerű, ha a logikai függvények tulajdonságaival részletesebben is megismerkedünk.

2.2.2. A logikai függvények megadási módjai

Egy logikai függvényt egyértelműen megadhatunk, ha felsoroljuk a független változók értékeinek összes lehetséges kombinációját és mindegyikhez melléírjuk a függvénykapcsolat által előírt függvényértéket. Az ilyen táblázatot *igazságtáblának* nevezik. A 2.2. ábrán az összes lehetséges kétváltozós függvényt adtuk meg igazságtábla segítségével. A táblázat első két oszlopában minden egyes sorban egy-egy függetlenváltozó-kombinációt tüntettünk fel, vagyis az A és B logikai változók mindenkori logikai értékeit. A táblázat többi oszlopának mindegyikében egyfajta módon hozzárendeltünk logikai értékeket minden egyes függetlenváltozó-kombinációhoz, vagyis

Független változó kombinációk		Függvényértékek (Z)															
A	B	f_0^2	f_1^2	f_2^2	f_3^2	f_4^2	f_5^2	f_6^2	f_7^2	f_8^2	f_9^2	f_{10}^2	f_{11}^2	f_{12}^2	f_{13}^2	f_{14}^2	f_{15}^2
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



2.2. ábra. Az összes lehetséges kétváltozós logikai függvény igazságtáblája

megadtunk egy-egy logikai függvényt. Az egyes függvénykapcsolatokat f_i^2-vel jelöltük, ahol az i index annak a bináris számnak a decimális megfelelője, amelyet az oszlopba írt logikai értékek mint bináris számjegyek alkotnak, a legalsó sort tekintve legkisebb helyértéknek. A felső index arra utal, hogy a függvény kétváltozós.

Az előzőek értelmében a 2.2. ábrán tulajdonképpen felsoroltuk mindazokat a teljesen határozott logikai feladatokat, amelyeket kétbemenetű és egykimenetű kombinációs hálózat megoldhat. Ezek mindegyike egy-egy *logikai függvényt* definiál. Ebből is következik, hogy $n=2$ és $m=1$ esetén 16 különböző függvény létezik.

$$N_f = (2^1)^{2^2} = 16.$$

Mivel csak két logikai érték létezik, a függvények egyértelmű megadásához elegendő azokat a változókombinációkat felsorolni, amelyekhez 1 függvényérték tartozik. Ugyanígy egyértelmű lenne természetesen, ha csak azokat a változókombinációkat sorolnánk fel, amelyek esetén a függvényérték 0.

Vizsgáljuk meg ezek után rendre a 2.2. ábrán megadott függvényeket.

Az f_0^2 és f_{15}^2 függvények értéke a változók értékeitől függetlenül mindig 0, ill. 1, ezért ezek valójában nem jelentenek függvénykapcsolatot, hanem konstansok, s így írható, hogy

$$f_0^2 \equiv 0 \quad \text{és} \quad f_{15}^2 \equiv 1.$$

Az f_1^2 függvény csak akkor 1 értékű, ha mindkét változó értéke 1. A logikai szorzás definíciója ismeretében megállapíthatjuk, hogy az f_1^2 függvénykapcsolat logikai szorzást ír elő a két változó között. Ezek után az f_1^2 függvényt megadhatjuk logikai (Boole-) algebrai alakban is:

$$Z = f_1^2(A, B) = A \cdot B.$$

Az f_2^2 függvény értéke csak akkor 1, ha $A=1$ és $B=0$, azaz $A=1$ és $\bar{B}=1$. Így a függvény algebrai alakja:

$$Z = f_2^2(A, B) = A \cdot \bar{B}.$$

A sok lehetséges algebrai alak közül az egyiket az alábbi megfogalmazás formális leírásával nyerjük pl. f_3^2 esetén: Z akkor és csakis akkor 1, ha

$$A = 1 \quad \textit{és} \quad \bar{B} = 1,$$

vagy

$$A = 1 \quad \textit{és} \quad B = 1.$$

A formális leírásban az ÉS kapcsolatot logikai szorzással, a VAGY kapcsolatot logikai összeadással kell kifejezni:

$$Z = f_3^2(A, B) = A \cdot \bar{B} + A \cdot B.$$

A logikai algebra megismert szabályai szerint átalakításokat végezve:

$$Z = f_3^2(A, B) = A \cdot \bar{B} + A \cdot B \equiv A \cdot (\bar{B} + B) \equiv A.$$

A függvényérték tehát független B változótól, csak A-tól függ, mégpedig azzal azonos, ami egyébként az igazságtáblából közvetlenül látszik.

Hasonló gondolatmenettel írható fel a többi függvény algebrai alakja is. A továbbiakban csak az algebrai alakokat közöljük.

$$Z = f_4^2(A, B) = \bar{A} \cdot B,$$

$$Z = f_5^2(A, B) = \bar{A}B + AB \equiv (A + \bar{A}) \cdot B \equiv B,$$

ami az igazságtábla alapján is könnyen megállapítható.

$$Z = f_6^2(A, B) = \bar{A} \cdot B + A \cdot \bar{B},$$

vagyis csak akkor 1 értékű, ha a két változó közül valamelyik 1 értékű, de nem mindkettő. Ez úgy is fogalmazható, hogy f_6^2 értéke csak akkor 1, ha a két változó értéke különböző. Ezek miatt nevezik ezt a függvénykapcsolatot KIZÁRÓ VAGY kapcsolatnak, vagy ANTIVALENCIÁ-nak. Elterjedt a „modulo 2 összeg" elnevezés is, amely a függvény későbbiekben ismertetendő különleges felhasználhatóságára utal. Ezt a függvénykapcsolatot gyakran külön műveleti jellel jelölik: $\oplus$.

$$Z = f_6^2(A, B) = A \oplus B$$

$$Z = f_7^2(A, B) = \bar{A} \cdot B + A \cdot \bar{B} + A \cdot B = \bar{A} \cdot B + A \cdot (B + \bar{B}) = \bar{A} \cdot B + A.$$

A kiadódott algebrai kifejezésről könnyen belátható, hogy logikai VAGY kapcsolatot jelent a két változó között. Erről meggyőződhetünk úgy is, hogy a változók helyére az összes lehetséges módon behelyettesítjük a logikai értékeket, de természetesen algebrai úton is kiadódik az alábbi módon:

f_7^2 kifejezésében az AB szorzatot tetszőleges számúszor írhatjuk le VAGY kapcsolatban, hiszen az egyszer már szerepel benne. Így

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

Folytatva a függvények felírását:

$$Z = f_8^2(A, B) = \bar{A}\cdot\bar{B}.$$

Figyeljük meg, hogy az f_8^2 oszlopban és f_7^2 oszlopban szereplő függvényértékek egymás ellentettjei. Így

$$\overline{f_8^2(A, B)} = f_7^2(A, B),$$

ami a De Morgan-tételből is következik.

$$Z = f_9^2(A, B) = \bar{A}\cdot\bar{B}+A\cdot B,$$

vagyis a függvény értéke csak akkor 1, ha a két változó értéke azonos. Ezért ezt a függvénykapcsolatot EKVIVALENCIA-nak nevezik és gyakran külön műveleti jellel jelölik: $\odot$

$$Z = f_9^2(A, B) = A\odot B.$$

A 2.2. ábra alapján megállapítható, hogy

$$\overline{f_6^2(A, B)} = f_9^2(A, B), \qquad \text{vagyis} \qquad \overline{A\oplus B} = A\odot B,$$

ami a De Morgan-tételből is következik, hiszen

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

$$Z = f_{10}^2(A, B) = \bar{A}\cdot\bar{B}+A\cdot\bar{B} = \bar{B}(\bar{A}+A) = \bar{B},$$

vagyis

$$f_{10}^2(A, B) = \overline{f_5^2(A, B)},$$

$$f_{11}^2(A, B) = \overline{f_4^2(A, B)} = \overline{\bar{A}\cdot B} = A+\bar{B},$$

$$f_{12}^2(A, B) = \overline{f_3^2(A, B)} = \bar{A},$$

$$f_{13}^2(A, B) = \overline{f_2(A, B)} = \overline{A\cdot\bar{B}} = \bar{A}+B,$$

$$f_{14}^2(A, B) = \overline{f_1^2(A, B)} = \overline{A\cdot B} = \bar{A}+\bar{B}.$$

A 2.2. ábrán látható igazságtáblán megjelöltük azokat az oszlopokat, amelyekben megadott függvények egymás negáltjai.

Az egyes függvényeket úgy is felírhattuk volna algebrai alakban, hogy a 0 függvényértékeket, vagyis a függvény negáltjait előállító változókombinációkból képezzük az algebrai alakot. Sok esetben így egyszerűbb kifejezések adódtak volna. Például f_7^2 esetén

$$Z = \overline{f_7^2(A, B)} = \bar{A}\cdot\bar{B},$$

vagyis

$$Z = f_7^2(A, B) = \overline{\overline{f_7^2(A, B)}} = \overline{\bar{A}\cdot\bar{B}} = A+B.$$

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

Az algebrai alak felírásának gondolatmenetében ilyenkor láthatóan csupán annyi a változás, hogy a tagadott függvény 1 értékeihez tartozó kombinációkból indulunk ki, és a függvény algebrai alakjához az így felírt algebrai alak negálása útján juthatunk el.

Összefoglalva megállapíthatjuk, hogy a logikai függvények két megadási módját ismertük meg egy egyszerű példán:

— az igazságtáblát,

— a logikai algebrai alakot.

Láttuk, hogy az igazságtábla alapján könnyen felírhatjuk az algebrai alakot. Ezt formálisan úgy végezhetjük, hogy VAGY kapcsolatba hozzuk azoknak a változókombinációknak megfelelő logikai szorzatokat, amelyek esetén a függvényérték 1 (ill. 0). A szorzatokban a változók ponáltan szerepelnek, ha a kombinációban 1 az értékük, negáltan, ha a kombinációban 0 az értékük. Az így felírt algebrai kifejezést a megismert azonos átalakításokkal egyszerűsítettük.

A továbbiakban a logikai függvényekre az

$$F(A, B, C, \ldots)$$

valamint

$$F(x_1, \ldots, x_n)$$

jelöléseket egyaránt alkalmazzuk. A függvény értékét általában F-fel jelöljük.

2.2.3. Azonos változókon értelmezett logikai függvények közötti összefüggések

A kombinációs hálózatok bemeneti változói az összes kimeneti változó értékét befolyásolhatják, ezért a hálózat által megoldott logikai feladat ugyanazon független változókon értelmezett logikai függvényekkel adható meg.

További vizsgálataink érdekében tehát célszerű összefoglalni, milyen kapcsolatban lehetnek egymással az azonos független változókon értelmezett logikai függvények.

Jelöljünk F_1-gyel és F_2-vel két ugyanazon független változókon értelmezett logikai függvényt.

Ha F_1 függvény értéke 1 minden olyan függetlenváltozó-kombináció esetén, amelyhez $F_2=1$ tartozik, akkor ezt úgy fejezik ki szavakban, hogy F_1 tartalmazza F_2-t, vagy F_2 kisebb F_1-nél. Jelölésben:

$$F_1 \supseteq F_2.$$

Nyilvánvaló, hogy helyes az alábbi következtetés:

Ha $F_1 \supseteq F_2$ és $F_2 \supseteq F_1$ egyidejűleg fennáll, akkor

$$F_1 \equiv F_2,$$

hiszen ebben az esetben F_1 akkor és csak akkor 1, ha F_2 is 1.

A logikai függvényekre értelemszerűen kiterjeszthetjük a logikai műveleteket.

F_1+F_2 olyan logikai függvényt jelöl, amelynek függvényértéke kizárólag azon függetlenváltozó-kombinációk esetén 1, amelyekhez $F_1=1$ vagy $F_2=1$ tartozik.

$F_1 \cdot F_2$ olyan logikai függvényt jelöl, amelynek függvényértéke azon és csak azon függetlenváltozó-kombinációk esetén 1, amelyekhez $F_1=1$ és $F_2=1$ tartozik.

A tagadást, ill. ellentett képzést már a 2.2. ábra alapján is kiterjesztettük a logikai függvényekre.

$\bar{F}$ olyan logikai függvényt jelöl, amely ugyanazokon a független változókon értelmezett, mint F, és függvényértéke azon és csak azon függetlenváltozó-kombinációk esetén 1, amelyekhez $F=0$ tartozik. Az előzőek alapján nyilvánvaló, hogy

$$F+\bar{F} \equiv 1,$$

$$F \cdot \bar{F} \equiv 0.$$

Könnyen belátható, hogy a nem teljesen határozott logikai feladatok leírásakor olyan logikai függvények adódnak, amelyek nem minden függetlenváltozó-kombinációhoz írnak elő függvényértéket. Ezeket a logikai függvényeket értelemszerűen *nem teljesen határozott függvényeknek* nevezhetjük. Az ilyen függvények negáltjai természetesen szintén nem teljesen határozottak, vagyis a fenti két azonosság ilyenkor csak az előírt függvényértékekre vonatkozik. Meg kell azonban jegyeznünk, hogy egy megépített kombinációs hálózat alapján megállapított logikai függvény mindig teljesen határozottnak adódik. Ugyanis bár az előírt logikai feladat nem volt teljesen határozott, a megtervezett hálózat természetesen a közömbös kombinációk esetén is létrehoz valamilyen kimeneti kombinációt. A logikai feladat szempontjából közömbös kimeneti kombinációk egyfajta szabadságot jelentenek a tervezőnek, ezért azokat úgy kell megválasztanunk a tervezés során, ahogyan az a kiadódó logikai hálózat tulajdonságai szempontjából a legkedvezőbb. Jegyezzük tehát meg, hogy a kombinációs hálózat *vizsgálata (analízise)* alapján adódó logikai függvények mindig teljesen határozottak. Csak abban az esetben beszélünk nem teljesen határozott logikai függvényről, ha a függvénykapcsolatot az előírt, *nem teljesen határozott logikai feladat* alapján állapítjuk meg a kombinációs hálózat *tervezése (szintézise)* céljából.

Ha egy nem teljesen határozott logikai függvény mellett annak tagadottját külön is megvalósíthatjuk kombinációs hálózatként, akkor a fentiek alapján az is előfordulhat, hogy a két kombinációs hálózat analízise útján nyert két teljesen határozott logikai függvény az eredetileg közömbös kombinációkhoz nem rendel ellentétes függvényértékeket. Ezt nyilvánvalóan a közömbös függvényértékeknek a két hálózatban történt eltérő rögzítése okozza, ami a tervezés során előfordulhat.

2.2.4. A logikai függvények kanonikus algebrai alakjai

A kombinációs hálózatok tervezésének kiindulási lépése a logikai függvény felírása a megoldandó feladat alapján. Ezért célszerű részletesebben foglalkoznunk a logikai függvények algebrai alakjaival.

A 2.2. ábrán tulajdonképpen egyértelműen megadtunk 16 egymástól különböző, teljesen határozott logikai feladatot. Egyetlen kimeneti pont esetén ugyanis minden logikai feladat megadható egyetlen logikai függvénykapcsolattal. Az egyes függetlenváltozó-kombinációkhoz tartozó függvényérték a kombinációs hálózat kimeneti változójának logikai értékével azonos, ezért azt a továbbiakban F betűvel is jelölhetjük, ezzel is hangsúlyozva, hogy a kombinációs hálózat egy kimeneti pontjának értékét logikai függvény írja elő.

A 2.2. ábrához hasonlóan igazságtáblával adtunk meg egy háromváltozós logikai függvényt a 2.3. ábrán.

Az igazságtábla segítségével nem teljesen határozott logikai függvényeket is megadhatunk. A 2.4. ábrán egy nem teljesen határozott háromváltozós logikai függvény

Független változó kombinációk			Függvényérték
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

2.3. ábra. Példa háromváltozós, teljesen határozott logikai függvény igazságtáblájára

Független változó kombinációk			Függvényérték
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	--
1	0	1	0
1	1	0	--
1	1	1	1

2.4. ábra. Példa háromváltozós, nem teljesen határozott logikai függvény igazságtáblájára

igazságtáblája látható. Ebben az esetben tehát vannak F oszlopának olyan rovatai, amelyekben nincs előírt függvényérték. Ezekbe vízszintes vonalakat írtunk. Megállapíthatjuk, hogy négy olyan egymástól különböző teljesen határozott logikai függvény képezhető, amelyet megvalósító kombinációs hálózat megoldja a 2.4. ábra logikai függvényével megadott logikai feladatot. A két határozatlan (közömbös) függvényértéket ugyanis összesen négy egymástól eltérő módon tudjuk határozottá tenni. A tervezés során az említett négy teljesen határozott logikai függvényt megvalósító négy kombinációs hálózat közül kell kiválasztanunk a megvalósítás szempontjából a legkedvezőbbet.

A teljesen határozott logikai függvények, mint már említettük, egyértelműen megadhatók azoknak a függetlenváltozó-kombinációknak a felsorolásával, amelyekhez tartozó függvényérték 1, vagy azoknak a függetlenváltozó-kombinációknak a felsorolásával, amelyekhez tartozó függvényérték 0. A Boole-algebra segítségével ezt a felsorolást algebrai alakban is megtehetjük. A 2.2. ábra alapján végzett vizsgálataink ból megállapítottuk, hogy az egyes függetlenváltozó-kombinációknak megfelelő logikai szorzatok hogyan képezhetők. Mint láttuk, a kombinációs hálózat kimenetére — jelen esetben a függvényértékre — vonatkozó algebrai kifejezés logikai szorzatok logikai összegeként írható fel. A 2.3. ábrán megadott logikai függvény négy függetlenváltozó-kombinációhoz ír elő függvényértékként 1-et. Írjuk fel ezek alapján a függvény algebrai alakját:

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

Ha azokat a logikai szorzatokat hozzuk VAGY kapcsolatba, amelyeknek megfelelő függetlenváltozó-kombinációk esetén a függvényérték 0, akkor F negáltjára (tagadottjára) kapunk algebrai kifejezést:

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

Természetesen mindkét algebrai kifejezés egyértelműen jellemzi a logikai függvényt. Levonhatjuk tehát azt a következtetést, hogy bármely logikai függvény az ÉS kapcsolat, a VAGY kapcsolat és a tagadás (invertálás) segítségével megadható. A nem teljesen határozott logikai függvények algebrai felírásakor a közömbös kombinációknak megfelelő szorzatokat általában zárójelben szokás feltüntetni. Így a 2.4. ábra alapján

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

A Boole-algebrában megengedett átalakítások segítségével egy logikai függvény algebrai alakja alapján nagyon sok olyan algebrai alak írható fel, amelyek mindegyike természetesen ugyanazt a logikai függvényt adja meg egyértelműen, de csak sok átalakítás elvégzése után ismerhető fel róluk, hogy azonosak. A 2.3. ábra alapján felírt algebrai alakot pl. az alábbi módon alakíthatjuk át:

$$\begin{aligned} F(A, B, C) &= \bar{A}B\bar{C}+\bar{A}BC+A\bar{B}\bar{C}+AB\bar{C} = \bar{A}B(C+\bar{C})+A\bar{C}(B+\bar{B}) = \\ &= \bar{A}B+A\bar{C} = \bar{A}B+A\bar{C}+\bar{A}A = \\ &= \bar{A}(A+B)+A\bar{C} = \bar{A}(A+B)+A\bar{C}+\bar{A}A = \\ &= \bar{A}(A+B)+A(\bar{A}+C) = \dots \quad \text{stb.} \end{aligned}$$

A logikai függvények algebrai alakjainak azonos átalakításaira vonatkoznak az alábbi általános azonosságok is:

$$F(x_1, x_2, \dots, x_n) = x_1 F(1, x_2, \dots, x_n)+\bar{x}_1 F(0, x_2, \dots, x_n),$$

$$F(x_1, x_2, \dots, x_n) = [x_1+F(0, x_2, \dots, x_n)]\cdot[\bar{x}_1+F(1, x_2, \dots, x_n)].$$

Mindkét azonosság egyszerűen bizonyítható azáltal, hogy $x_1=1$-re és $x_1=0$-ra felírhatjuk a jobb oldal és bal oldal értékét. Az első azonosság pl.: $x_1=1$ esetén:

$$F(1, x_2, \dots, x_n) = 1\cdot F(1, x_2, \dots, x_n)+0\cdot F(0, x_2, \dots, x_n),$$

ami nyilvánvalóan fennáll.

$x_1=0$ esetén:

$$F(0, x_2, \dots, x_n) = 0\cdot F(1, x_2, \dots, x_n)+1\cdot F(0, x_2, \dots, x_n),$$

ami szintén érvényes.

Hasonló módon bizonyítható a második azonosság is.

Az ismertetett két azonosságot a logikai függvényekre vonatkozó *kifejtési tételnek* nevezik. Példaként alkalmazzuk a kifejtési tétel azonosságait a 2.3. ábrán szereplő logikai függvény algebrai alakjaira. Az azonosságokban x_1-gyel jelölt változó szerepét természetesen bármely változó betöltheti. Válasszuk x_1 szerepére az A változót. Ekkor

$$\begin{aligned} F(1, B, C) &= 0B\bar{C}+0BC+1\bar{B}\bar{C}+1B\bar{C} \equiv \\ &\equiv \bar{B}\bar{C}+B\bar{C} \equiv \bar{C}(\bar{B}+B) \equiv \bar{C} \\ F(0, B, C) &= 1B\bar{C}+1BC+0\bar{B}\bar{C}+0B\bar{C} \equiv \\ &\equiv B\bar{C}+BC \equiv B(C+\bar{C}) \equiv B, \end{aligned}$$

tehát az első azonosság alapján:

$$F(A, B, C) = A\cdot F(1, B, C)+\bar{A}\cdot F(0, B, C) = A\bar{C}+\bar{A}B.$$

A második azonosság alapján:

$$F(A, B, C) = [A+F(0, B, C)]\cdot[\bar{A}+F(1, B, C)] = (A+B)\cdot(\bar{A}+\bar{C}).$$

A kombinációs hálózatok tervezéséhez a megoldandó logikai feladatot leíró logikai függvény algebrai alakjából célszerű kiindulnunk. Mivel egy logikai függvénynek több algebrai alakja van, ezért szükséges olyan speciális tulajdonságú algebrai alakot

keresnünk, amely tulajdonságok kizárják, hogy egy logikai függvényhez több, ilyen tulajdonságú algebrai alak tartozzon. Az ilyen algebrai alak a logikai függvény *kanonikus (vagy normál) alakja*. A 2.3. ábra alapján felírt, logikai szorzatok logikai összegével képzett algebrai alakra olyan tulajdonságok érvényesek, amelyek miatt az kanonikusnak (normálnak) tekinthető. Ezt a függvényalakot úgy kaptuk, hogy logikai VAGY kapcsolatba hoztuk azoknak a függetlenváltozó-kombinációknak megfelelő logikai szorzatokat, amelyek esetén a függvényérték 1. Példánkban az így nyert

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

függvényalak tulajdonságai a következők:

— mindegyik szorzat egy olyan függetlenváltozó-kombinációt képvisel, amelyhez tartozó függvényérték 1,
— mindegyik szorzatban az összes független változó szerepel ponált vagy negált alakban.

Ebből a két tulajdonságból egyértelműen következik, hogy minden teljesen határozott logikai függvénynek csak egyetlen ilyen tulajdonságú algebrai alakja létezhet. (A szorzatok és a változók sorrendje természetesen tetszőleges lehet, a sorrendváltoztatás nem változtatja meg az algebrai alak fenti tulajdonságait.) Így ez kanonikus alaknak tekinthető. Képzési módjából következően nevezzük ezt a kanonikus alakot a logikai függvény *diszjunktív kanonikus alakjának*. A diszjunktív kanonikus alakban szereplő minden egyes logikai szorzat nyilvánvalóan egy-egy olyan logikai függvénynek tekinthető, amelynek függvényértéke csak egyetlen függetlenváltozó-kombináció esetén 1. (Mivel egy szorzat értéke csak akkor 1, ha minden tényezője 1, ezért ez a függetlenváltozó-kombináció éppen az, amelynek alapján magát a szorzatot felírtuk.) Így tehát a diszjunktív kanonikus alak nem más, mint speciális elemi logikai függvények logikai összege. Innen ered tulajdonképpen a kanonikus elnevezés. Ezeket a speciális elemi logikai függvényeket *mintermeknek* nevezik és általában

$$m_i^n$$

módon jelölik, ahol n jelöli a független változók számát, i pedig az illető mintermnek megfelelő változókombinációt jelölő bináris szám decimális értéke. Ezzel a jelöléssel a 2.3. ábrán adott függvény diszjunktív kanonikus alakja a következőképpen írható fel:

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

Az azonos független változókon értelmezett logikai függvényekre ismertetett összefüggések alapján nyilvánvaló, hogy

$$F \supseteq m_2^3; \quad F \supseteq m_3^3; \quad F \supseteq m_4^3; \quad F \supseteq m_6^3.$$

Innen ered tulajdonképpen a minterm elnevezés. Az is könnyen belátható, hogy n

számú független változó esetén összesen 2^n számú egymástól különböző minterm képezhető, vagyis annyi, ahány függetlenváltozó-kombináció létezik. A diszjunktív kanonikus alak megadása az előzőek szerint tehát azt jelenti, hogy logikai VAGY kapcsolatba hozzuk az összes képezhető mintermek közül azokat, amelyeket a logikai függvény tartalmaz.

A korábbiakban már megállapítottuk, hogy a teljesen határozott logikai függvényeket egyértelműen megadhatjuk azoknak a szorzatoknak a VAGY kapcsolatával is, amelyeknek megfelelő függetlenváltozó-kombináció esetén a függvényérték 0. Az eljárás azon a tényen alapszik, hogy $F=0$ esetén $\bar{F}=1$, ha tehát az $F=0$-hoz tartozó kombinációk logikai összegét képezzük, akkor az F függvény $\bar{F}$ negáltjának diszjunktív kanonikus alakjához jutunk. A 2.3. ábra alapján már fel is írtuk az

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

függvényalakot, amely tehát nem más, mint a függvény negáltjának diszjunktív kanonikus alakja. Alkalmazva a mintermekre bevezetett jelöléseket:

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

A függvény negáltja tehát azokat a mintermeket tartalmazza, amelyeket a függvény nem tartalmaz. (Ez természetesen csak akkor igaz, ha teljesen határozott logikai függvényről van szó.)

Kétértékű változók esetén képezve a függvény negáltjának a negáltját, természetesen a kiindulási függvényhez jutunk. Ezt a gondolatmenetet követve, és alkalmazva a De Morgan-tételt, egy újabb kanonikus alakhoz juthatunk.

Képezzük a 2.3. ábrán megadott logikai függvény negáltjának a negáltját.

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

A kiadódott algebrai alakról az alábbiakat állapíthatjuk meg:

— logikai összegek logikai ÉS kapcsolatban szerepelnek benne. A logikai összegek azokból a függetlenváltozó-kombinációkból képezhetők, amelyekhez a függvény 0 értéke tartozik.
— mindegyik összegben az összes független változó szerepel ponált vagy negált alakban. Az egyes tényezőket alkotó logikai összegekben a 0 értékű logikai változók szerepelnek ponált és az 1 értékűek negált alakjukban.

Ebből a két tulajdonságból egyértelműen következik, hogy minden teljesen határozott logikai függvénynek csak egyetlen ilyen tulajdonságú algebrai alakja létezhet. Így ez is kanonikus alak. Képzési módjából következően ezt a kanonikus alakot a logikai függvény *konjunktív kanonikus alakjának* nevezik. A konjunktív kanonikus alakban szereplő minden egyes logikai összeg egy-egy olyan logikai függvénynek tekinthető, amelynek függvényértéke csak egyetlen függetlenváltozó-kombináció esetén 0. (Mivel az összeg csak akkor 0, ha minden tagja 0, ezért ez a függet-

lenváltozó-kombináció éppen az, amelyben az egyes változók akkor szerepelnek ponált alakban, ha aktuális értékük 0 és negáltban, ha értékül: 1). Így tehát a konjunktív kanonikus alak speciális elemi logikai függvények logikai szorzatának tekinthető. Ezeket a speciális elemi logikai függvényeket *maxtermeknek* nevezik, és általában az alábbi módon jelölik:

$$M_i^n,$$

ahol n jelöli a független változók számát, i pedig egy decimális számot jelöl, amely az illető maxtermnek formailag megfelelő függetlenváltozó-kombinációt jelölő bináris szám decimális megfelelője.

Az i index képzésekor megállapodás szerint a maxtermben szereplő ponált változóknak 1 értéket, negált változóknak pedig 0 értéket tulajdonítunk. Ezzel a jelölésmódda! a 2.3. ábrán adott függvény konjunktív kanonikus alakja az alábbi módon írható fel:

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

Az azonos független változókon értelmezett logikai függvényekre ismertetett összefüggések alapján nyilvánvaló, hogy

$$M_7^3 \supseteqq F; \quad M_6^3 \supseteqq F; \quad M_2^3 \supseteqq F; \quad M_0^3 \supseteqq F.$$

Egy maxterm ugyanis a független változók aktuális értékeinek csak egyetlen kombinációja esetén 0; így $F=1$ esetén az adott maxterm értéke is biztosan 1. Innen ered tulajdonképpen a maxterm elnevezés.

Az is könnyen belátható, hogy n számú független változó esetén összesen 2^n számú egymástól különböző maxterm képezhető, vagyis annyi, amennyi kombinációja létezik a független változók aktuális értékeinek. A konjunktív kanonikus alak megadása az előzőek szerint azt jelenti, hogy logikai ÉS kapcsolatba hozzuk azokat a maxtermeket, amelyek a logikai függvényt tartalmazzák.

Vizsgáljuk meg, hogy a bevezetett általános jelölések felhasználásával hogyan lehet a két kanonikus alakot egymásba átalakítani. Induljunk ki példaképpen a 2.3. ábrán adott logikai függvény diszjunktív kanonikus alakjából:

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

A függvény tagadottjának diszjunktív kanonikus alakjában azok a mintermek szerepelnek, amelyeket a függvény nem tartalmaz:

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

Képezzük ennek a kifejezésnek a negáltját:

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

A mintermek negáltja természetesen maxtermeket eredményez, amelyekben szereplő függetlenváltozó-kombinációk ellentettjei a mintermekben szereplőknek.

A keletkező maxtermek alsó indexei tehát az egyes mintermek alsó indexeinek megfelelő bináris számok helyértékenkénti ellentett képzése útján adódó decimális megfelelők lesznek. Mint ismeretes, ezek úgy határozhatók meg, hogy az egyes minterm alsó indexeket 2^n-1-ből kivonjuk (n jelöli a független változók számát, vagyis a függetlenváltozó-kombinációknak megfelelő bináris számok helyértékeinek számát.)

Jelen esetben tehát:

$$\overline{m_0^3} = M_7^3; \quad \overline{m_1^3} = M_6^3; \quad \overline{m_5^3} = M_2^3; \quad \overline{m_7^3} = M_0^3,$$

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

ami természetesen megegyezik a korábban kapott eredménnyel.

Az előzőek alapján könnyen belátható, hogy egy teljesen határozott logikai függvény negáltjának konjunktív kanonikus alakjában azok a maxtermek szerepelnek, amelyek a függvény konjunktív kanonikus alakjában nem szerepelnek. Így az előbbi módszerhez hasonlóan a konjunktív kanonikus alakból kiindulva is könnyen felírhatjuk a diszjunktív kanonikus alakot az alsó indexekre vonatkozó formális szabályok segítségével.

Megjegyezzük, hogy a mintermeket és a maxtermeket összefoglalóan gyakran nevezik *termeknek*.

2.2.5. Az elvi logikai rajz

A kis integráltsági fokú építőelem-készlet olyan integrált áramköri tokokból áll, amelyek néhány elemi kombinációs vagy sorrendi hálózatot tartalmaznak. A tokokban megvalósított elemi kombinációs hálózatok működését leíró logikai függvény legtöbbször olyan egyszerű, hogy valamelyik logikai alapművelettel adható meg. Az ilyen elemi kombinációs hálózatokat *kapuknak* nevezik. Ebben az értelmezésben definiálhatunk például

ÉS (AND), VAGY (OR), INVERTER, NEMÉS (NOT AND=NAND), NEM-VAGY (NOT OR=NOR)

kapukat, de könnyen belátható, hogy más logikai műveleteket is tekinthetünk alapműveleteknek és így más kapukat is definiálhatunk. Más kérdés az, hogy a sokfajta lehetséges kapudefiníció közül melyek azok, amelyeket egy adott építőelem-készletben ténylegesen megvalósítottak. A 2.5. ábrán összefoglaltuk néhány kapu definícióját és leggyakrabban alkalmazott rajzjelét. Megjegyezzük, hogy a rajzjelekre a felsoroltakon kívül számos más megoldás, sőt szabvány is létezik. A magyar szabvány az MSZ 9200/33—73 szám alatt rögzíti a kötelező előírásokat a kétállapotú (bináris) logikai elemek rajzjeleire vonatkozóan.

A kapuk és a rajzjeleik felhasználásával rajzzal megadhatjuk a logikai függvényeket és így a kombinációs hálózatokat. Például a 2.3. ábrán igazságtáblával meg-

Megnevezés	Rajzjel	Függvény
ÉS (AND) kapu		$F = x_1 x_2 \ldots x_n$
VAGY (OR) kapu		$F = x_1 + x_2 + \ldots + x_n$
INVERTER		$F = \bar{x}$
NEM ÉS (NAND) kapu		$F = \overline{x_1 x_2 \ldots x_n}$
NEM VAGY (NOR) kapu		$F = \overline{x_1 + x_2 + \ldots + x_n}$
KIZÁRÓ VAGY (EXOR) kapu		$F = x_1 \oplus x_2 \oplus \ldots \oplus x_n$

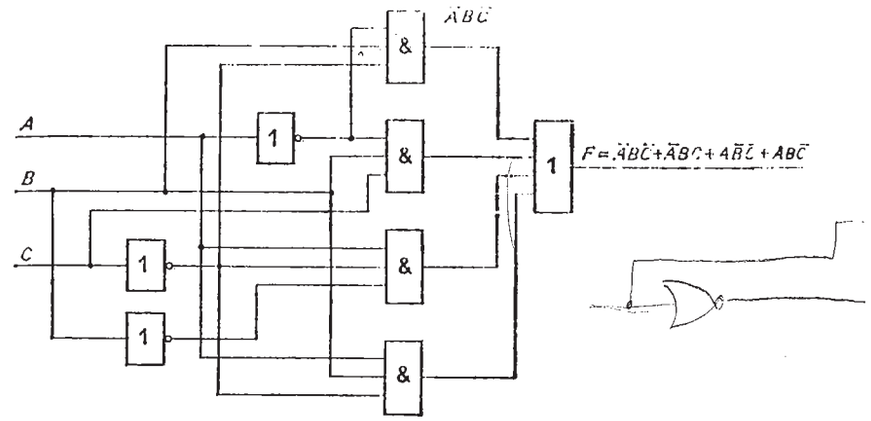
2.5. ábra. Néhány kapu definíciója és a leggyakrabban alkalmazott rajzjelek

adott logikai függvény ábrázolásához induljunk ki a diszjunktív normálalakból:

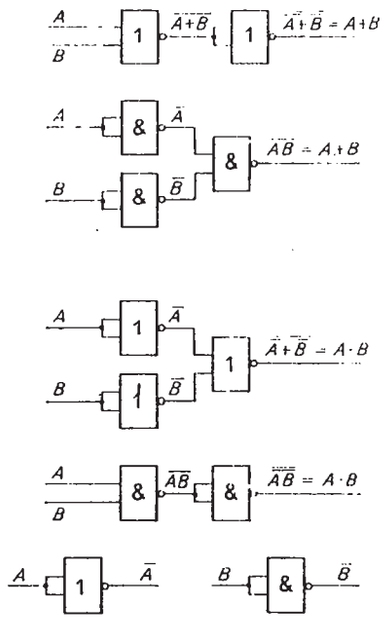
$$F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C}.$$

Látható, hogy a függvény (és természetesen minden logikai függvény) ÉS, VAGY és INVERTÁLÁS műveletekkel egyértelműen megadható. Ha az építőelem-készletben rendelkezésünkre állnak e műveleteknek megfelelő kapuk, akkor a függvényt a 2.6. ábra szerint ábrázolhatjuk. A diszjunktív normálalakból kiindulva ábrázolhatjuk a függvényt kizárólag NAND vagy NOR kapuk felhasználásával is. Könnyen belátható ugyanis, hogy az ÉS, VAGY és INVERTÁLÁS műveletek mindegyike külön-külön ábrázolható, vagy csak NAND, vagy csak NOR kapuk kizárólagos felhasználásával. Ezt mutatja be a 2.7. ábra. A fenti függvény NAND kapus ábrázolását követhetjük a 2.8. ábrán. Hasonló módon járhatunk el például NOR kapus ábrázolás esetén.

A bemutatott rajzokon a kapuk közötti összekötéseket tüntettük fel és nem jelöltük meg, hogy mely kapuk vannak közös integrált áramköri tokban, azaz közös építőelemben megvalósítva. Az ilyen ábrázolásmódot a továbbiakban *elvi logikai*



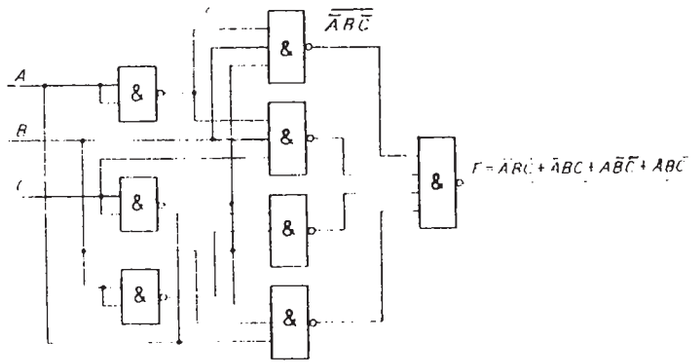
2.6. ábra. Logikai függvény ábrázolása ÉS kapukkal, VAGY kapukkal és INVERTEREKKEL



2.7. ábra. Az ÉS, VAGY és INVERTÁLÁS műveletek ábrázolása csak NAND, ill. NOR kapuk kizárólagos felhasználásával

rajznak nevezzük. Az elvi logikai rajzból tehát úgy kaphatjuk meg a logikai kapcsolási tervet, hogy a kapukat építőelemekhez rendeljük. A logikai kapcsolási tervben használható rajzjeleket Magyarországon az említett MSZ 9200/33—73 szabvány rögzíti.

A továbbiakban nem foglalkozunk az elvi logikai rajz átalakításával logikai kapcsolási tervvé, mert a bemutatandó tervezési eljárások lényegét ez nem érinti, az



2.8. ábra. Logikai függvény ábrázolása NAND kapukkal

építőelemekhez való legkedvezőbb hozzárendelés pedig az elvi logikai rajz alapján elvégezhető, természetesen általában próbálgatással.

A logikai kapcsolási terv egyszerűségét és így a megépítés gazdaságosságát nyilvánvalóan kedvezően befolyásolja:

a) a felhasznált kapuk számának csökkentése,

b) az összeköttetések számának csökkentése,

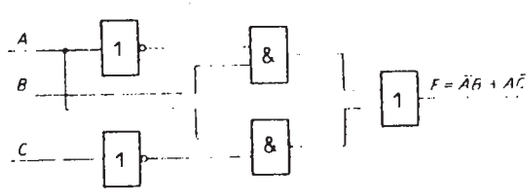
c) a kapukat megvalósító építőelem-fajták optimális megválasztása.

A *c)* pontban említett optimális megválasztásra általános módszer nem adható, hiszen az adott helyzetben rendelkezésre álló építőelem-készlet tulajdonságait kell figyelembe venni, és nem küszöbölhető ki az eljárás próbálgatásos jellege. Az optimalitás sem mindig azt jelenti egyértelműen, hogy a lehető legkevesebb építőelemet használjuk fel. Sokszor az építőelemek ára, a későbbi könnyű változtatási lehetőségek, valamint a modularitás biztosítása és a könnyű bemérhetőség szintén fontos szempontok.

Az *a)* és *b)* pontban említett egyszerűsítések szemléltetésére alakítsuk át az ábrázolt függvény algebrai alakját:

$$F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C} = \bar{A}B(\bar{C} + C) + A\bar{C}(\bar{B} + B) = \bar{A}B + A\bar{C}.$$

Ez utóbbi algebrai alak alapján szintén összeállíthatunk egy elvi logikai rajzot. Az ÉS, VAGY kapus és INVERTER-es változatot a 2.9. ábrán láthatjuk. Látszik, hogy mind a kapuk, mind az összeköttetések száma jelentősen csökkent a 2.6. ábra szerinti realizációhoz képest.



2.9. ábra. A 2.6. ábrán ábrázolt függvény elvi logikai rajza egyszerűsítő átalakítások után

Az eddig bemutatott elvi logikai rajzoknak közös jellemzőjük, hogy a bemeneti INVERTER-ektől eltekintve az egymás után kapcsolódó kapuk száma legfeljebb kettő. Az ilyen tulajdonságú kombinációs hálózatokat ***kétszintű kombinációs hálózatoknak*** nevezik. A két szint természetesen az elvi logikai rajz kiindulási alapját képező algebrai alakból adódik, hiszen akár a normál, akár az egyszerűsített alakból indultunk ki, mindig a logikai szorzatok eredményezték az egyik szintet, a logikai összegezés pedig a másikat.

A diódás, tranzisztoros építőelemek idejében a diódás kapuk szintrontó hatása miatt kettőnél több kaput áramkörileg sem volt megengedett egymás után kapcsolni tranzisztoros szinthelyreállító fokozat nélkül. Ezért akkor a kétszintű hálózatok kialakítása a tranzisztorokkal való takarékoskodást célozta. Az integrált áramköri építőelem-készletben megvalósított kapuk kimenete azonban eleve tranzisztoros fokozat, így emiatt nem feltétlenül szükséges kétszintű hálózatok kialakítására törekedni. Több szint megengedése esetén ugyanis várhatóan gazdaságosabb hálózathoz juthatunk a felhasznált építőelemek száma és az összeköttetések egyszerűsége szempontjából egyaránt. A több szintű kombinációs hálózatok azonban a nagyobb jelterjedési idő miatt lassúbbak a kétszintűeknél, és tervezési módszereik egyértelmű megfogalmazása is nehezebb. Sok esetben a végeredményként adódó több szintű hálózat gazdaságosságát csak úgy tudjuk megítélni, hogy összehasonlítjuk a kétszintű megoldással.

Fentiek alapján indokolt, ha először megismerkedünk azokkal a kétszintű kombinációs hálózatokat előállító tervezési módszerekkel, amelyek a kis integráltsági fokú építőelem-készlet esetében alkalmazhatók.

A kétszintű megvalósítás gazdaságossága szempontjából kedvező, ha az elvi logikai rajz létrehozásakor a lehető legkevesebb betűszimbólumot és műveletet tartalmazó diszjunktív vagy konjunktív algebrai alakból indulunk ki, amint ezt a 2.6. és 2.9. ábrák összehasonlítása is alátámasztja.

Figyeljük meg, hogy az elvi logikai rajzon a felhasznált kapuk száma és az összeköttetések száma egyaránt csökken, ha a kapubemenetek számának csökkentésére törekszünk. Ezért a továbbiakban arra keresünk választ, hogy milyen szisztematikus eljárásokkal találhatjuk meg egy tetszőleges logikai függvénynek azon algebrai alakjait, amelyek alapján a lehető legkevesebb kapubemenetet tartalmazó kétszintű elvi logikai rajzhoz jutunk. Kezdetben feltételezzük, hogy a tervezendő kombinációs hálózatnak csak egyetlen kimenete van, tehát működését egyetlen logikai függvénynyel írhatjuk le.

A keresett eljárásoknak alkalmasaknak kell lenniük arra, hogy célszerűen valamelyik kanonikus normálalakból kiindulva szisztematikus lépésekben szolgáltassák egy tetszőleges logikai függvény egyszerűsített diszjunktív vagy konjunktív algebrai alakjait. Az ilyen eljárásokat függvényminimalizáló eljárásoknak nevezik.

2.3. A logikai függvények minimalizálása

Egy adott feladatot megvalósító logikai függvény algebrai alakja akkor a legegyszerűbb, ha nem létezik olyan újabb algebrai alak, amelyben kevesebb betű és kevesebb művelet szerepel.

A továbbiakban nézzük meg a logikai függvények egyszerűsítésének alapját képező átalakításokat egy egyszerű példa alapján.

Induljunk ki egy függvény diszjunktív normálalakjából:

$$F_m = m_1^3+m_3^3+m_5^3+m_7^3.$$

Azonos átalakításokkal hozzuk a függvényt a lehető legegyszerűbb diszjunktív alakra.

A függvény mintermes alakja az alábbi:

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

Az algebrai alak jobb oldalán levő logikai összegzés első és második tagjából $\bar{A}C$, harmadik és negyedik tagjából AC logikai szorzat kiemelhető:

$$F(A, B, C) = \bar{A}C(\underbrace{\bar{B}+B}_{1})+AC(\underbrace{\bar{B}+B}_{1}) = \bar{A}C+AC.$$

Az egyszerűsített alakban már a B változó nem szerepel.

Ha az összevonásra került eredeti mintermeket jobban szemügyre vesszük, láthatjuk, hogy azok csak egy helyértéken térnek el egymástól. Ez tehát azt jelenti, hogy van egy logikai változó, amely az egyik mintermben ponált, a másikban negált értékével szerepel, a többi logikai változó pedig mindkét mintermnek megfelelő bináris kombinációban azonos értékű. Az ilyen tulajdonságokkal rendelkező mintermeket *szomszédos mintermeknek* nevezik. Például a négyváltozós mintermek közül az m_7^4 és m_{15}^4:

$$m_7^4 = \bar{A}BCD \quad \text{és} \quad m_{15}^4 = ABCD$$

valóban csak egy helyértéken térnek el egymástól, vagyis a két minterm szomszédos.

Az előbbi gondolatmenetből egyébként az is következik, hogy a logikai függvény egyszerűsítésének egyik fontos lépése a szomszédos mintermek megkeresése, párba válogatása, mert két szomszédos minterm mindig helyettesíthető egy olyan szorzattal (termmel), amelyben az egyik változó nem szerepel. Éppen az a változó marad el, amely a két szomszédos mintermet megkülönbözteti.

Az n változós logikai függvény egy mintermjének n szomszédos mintermje lehet, hiszen n helyértéken különbözhetnek egy változóban.

Előfordulhat, hogy két-két szomszédos minterm összevonásaként adódó termek ismét csak abban különböznek egymástól, hogy ugyanaz a változó az egyik szorzatban ponált a másikban negált értékével szerepel. Ilyen esetben ezt a két termet *szomszédos termnek* nevezik, és ezek is mindig helyettesíthetők egy olyan termmel, amelyből az eltérő módon szereplő változó kimarad.

Példánkban az egyszerűsítés folytatható, hiszen a kiadódott két termben ugyanazok a változók vannak és csak egy helyértéken különböznek egymástól:

$$F(A, B, C) = \bar{A}C+AC = C(\underbrace{\bar{A}+A}_{1}) = C.$$

A példából az is kiderült, hogy két betű elhagyásához négy mintermet kellett összevonni. Általános érvényű az a megállapítás, hogy k db betű elhagyásához 2^k db mintermet kell összevonni.

Az egyszerűsítés során egy mintermet a párba válogatáshoz az $A+A=A$ azonosság alapján többször is szerepeltetni lehet a logikai kifejezésben.

A logikai függvényt egyszerűsítő eljárások egyik legfontosabb lépése tehát a szomszédos mintermek megkeresése, párba válogatása. A lehetséges összevonások után a kiadódó termek közül szintén meg kell keresni azokat, amelyek szomszédosak. Az eljárást mindaddig folytatni kell, amíg a logikai függvény olyan szorzatok összege nem lesz, amelyekből már egyetlen betű sem hagyható el anélkül, hogy a logikai függvény meg ne változna. Az ilyen logikai összegben szereplő szorzatokat, termeket *prímimplikánsoknak* nevezik. Látható, hogy a logikai függvény legegyszerűbb diszjunktív alakja prímimplikánsok összege.

A logikai függvényt egyszerűsítő eljárások célja tehát, az előzőekben említett prímimplikánsok megkeresése. Látni fogjuk, hogy egy logikai függvénynek több ekvivalens legegyszerűbb alakja létezhet.

Az előzőekben a logikai függvény diszjunktív, vagyis mintermes alakjából indultunk ki, de gondolatmenetünk természetesen a függvények konjunktív, vagyis maxtermes alakjára is érvényes. Két szomszédos maxterm ugyanis szintén összevonható egyetlen összeggé, amely nem tartalmazza a két maxtermet megkülönböztető változót. Például az

$$M_0^3 = \bar{A}+\bar{B}+\bar{C} \quad \text{és} \quad M_4^3 = A+\bar{B}+\bar{C}$$

háromváltozós maxtermek szorzata azonos átalakítások után:

$$(\bar{A}+\bar{B}+\bar{C})(A+\bar{B}+\bar{C}) = (\bar{A}+(\bar{B}+\bar{C}))(A+(\bar{B}+\bar{C})) =$$
$$= A\bar{A}+(A+\bar{A})(\bar{B}+\bar{C})+(\bar{B}+\bar{C}) = \bar{B}+\bar{C}.$$

Így, ha az egyszerűsítő eljárás lépéseit mintermes alakból kiindulva fogalmazzuk meg, akkor azok a maxtermes alakból kiindulva is érvényesek maradnak és a végeredményt diszjunktív alak helyett konjunktív alakban kapjuk.

A továbbiakban a függvényminimalizálási eljárások két változatával foglalkozunk.

2.3.1. A grafikus minimalizálás

Tételezzük fel kezdetben, hogy az egyszerűsítendő függvény teljesen határozott és adott az igazságtáblája. A 2.10. ábrán példaként egy háromváltozós függvény igazságtábláját adtuk meg. Ugyanezen az ábrán ábrázoltunk egy olyan táblázatot, amely az igazságtábla fejléceinek átrendezése útján származtatható. Ez utóbbi táblázatból az egyes függvényértékekhez tartozó függetlenváltozó-kombinációk a táblázat peremeiről a 2.10. ábrán nyilakkal jelölt módon olvashatók ki. Az ilyen módon átrendezett igazságtáblát *Karnaugh-táblának* nevezik és az átrendezés olyan, hogy a szomszédos változókombinációk, azaz mintermek a táblának helyileg is szomszédos cellájába kerülnek. A 2.11. ábrán az egyes cellákba beírtuk, hogy háromváltozós esetben mely mintermek tartoznak az egyes cellákhoz. Látható, hogy a közös oldallal rendelkező cellákhoz tartozó mintermek valóban szomszédosak. (Pl.: $m_2^3 = \bar{A}B\bar{C}$ és $m_6^3 = AB\bar{C}$.) Azt is megfigyelhetjük, hogy szomszédosság szempontjából a Karnaugh-tábla a szélein összefüggőnek tekintendő. Így például $m_1^3 = \bar{A}\bar{B}C$ és $m_5^3 = A\bar{B}C$ szomszédosak. A háromváltozós Karnaugh-táblán úgy ábrázolhatunk egy tetszőleges háromváltozós függvényt, hogy 1-t írunk azokba a cellákba, amelyekhez tartozó mintermeket tartalmazza a függvény, és 0-t a többibe. Ha a függvény teljesen határozott, akkor természetesen elegendő az 1 értékek beírása a táblába. Mint láttuk a Karnaugh-tábla peremezése (fejléce) olyan, hogy a helyileg szomszédos 1-es bejegyzések szomszédos mintermeket jelölnek, ezért az egyszerűsítéshez szükséges párba válogatásuk könnyen elvégezhető. A 2.12. ábrán a kétváltozós és a négyváltozós Karnaugh-táblák peremezését, valamint a mintermek elhelyezkedését mutatjuk be. A négyváltozós táblán a szomszédosság vizsgálatakor szintén figyelembe kell

	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

$m_0^3 = ABC$ $\quad m_4^3 = \bar{A}BC$

$m_1^3 = \bar{A}\bar{B}C$ $\quad m_5^3 = \bar{A}B\bar{C}$

$m_2^3 = A\bar{B}\bar{C}$ $\quad m_6^3 = \bar{A}\bar{B}C$

$m_3^3 = A\bar{B}C$ $\quad m_7^3 = \bar{A}\bar{B}\bar{C}$

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

C

2.10. ábra. Az igazságtábla átrendezése a grafikus minimalizálás bevezetéséhez

AB \ C	0	1
00	m_0^3	m_1^3
01	m_2^3	m_3^3
11	m_6^3	m_7^3
10	m_4^3	m_5^3

2.11. ábra. A mintermek elhelyezkedése háromváltozós Karnaugh-táblán

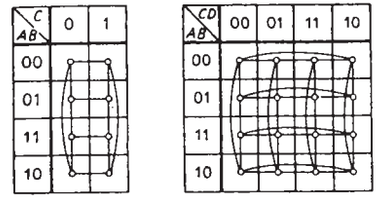
vennünk, hogy a tábla a szélein összefüggő a szomszédosság szempontjából. A 2.13. ábrán bejelöltük az összes páronkénti szomszédosságot mind a háromváltozós, mind a négyváltozós tábla esetén. Azok a cellák felelnek meg szomszédos mintermeknek, amelyekben levő köröket közvetlen vonalak kötik össze. Az ábrázolt szomszédossági viszonyokat könnyen ellenőrizhetjük az egyes cellákhoz tartó kombinációk kiolvasásával.

A Karnaugh-táblák peremezését egyértelműen jelezhetjük a változók bináris-érték-kombinációinak felírása nélkül is. A 2.14. ábrán a 2.12. ábra négyváltozós

A \ B	0	1
0	m_0^2	m_1^2
1	m_2^2	m_3^2

AB \ CD	00	01	11	10
00	m_0^4	m_1^4	m_3^4	m_2^4
01	m_4^4	m_5^4	m_7^4	m_6^4
11	m_{12}^4	m_{13}^4	m_{15}^4	m_{14}^4
10	m_8^4	m_9^4	m_{11}^4	m_{10}^4

2.12. ábra. A mintermek elhelyezkedése a kétváltozós és a négyváltozós Karnaugh-táblákon



2.13. ábra. Szomszédossági viszonyok a háromváltozós és a négyváltozós Karnaugh-táblán

m_0^4	m_1^4	m_3^4	m_2^4
m_4^4	m_5^4	m_7^4	m_6^4
m_{12}^4	m_{13}^4	m_{15}^4	m_{14}^4
m_8^4	m_9^4	m_{11}^4	m_{10}^4

C, B, A, D

2.14. ábra. A négyváltozós Karnaugh-tábla peremezésének megadása

táblájának peremezését adtuk meg az oldalakon elhelyezett vonalakkal. Az egyes változókhoz tartozó vonalak jelölik ki azt a táblarészt, amelynek cellái az illető változó ponált értékét tartalmazó mintermeknek felelnek meg. A táblarészeket kijelölő változókat természetesen tetszőlegesen felcserélhetjük, csupán a táblarészeket kijelölő vonalak egymáshoz képesti helyzete lényeges. Megengedett az olyan peremezés, ahol az egyik változó ponált értékéhez a középső két mezőt, egy másik változó ponált értékéhez pedig az egyik (bármelyik!) oldalon levő szélső két mezőt rendeljük. Ebben az esetben csupán az egyes cellákhoz tartozó mintermek cserélődnek fel, de a tábla változatlan módon ábrázolja a szomszédossági viszonyokat. Egy ilyen másfajta megengedett peremezést szemléltet a 2.15. ábra.

Ezek után vizsgáljuk meg, hogy a mintermeknek a Karnaugh-táblán könnyen elvégezhető szomszédosságvizsgálata és szomszédos párokba válogatása alapján miként fogalmazhatjuk meg a függvényminimalizáló eljárás lépéseit.

Tételezzük fel először, hogy az egyszerűsítendő logikai függvény teljesen határozott. Ez tehát azt jelenti, hogy a Karnaugh-tábla minden egyes cellájához előírt függvényérték tartozik.

m_8^4	m_{12}^4	m_4^4	m_0^4
m_{10}^4	m_{14}^4	m_6^4	m_2^4
m_{11}^4	m_{15}^4	m_7^4	m_3^4
m_9^4	m_{13}^4	m_5^4	m_1^4

2.15. ábra. A négyváltozós Karnaugh-tábla peremezésének egy megengedett megváltoztatása

Természetesen az egyszerűsítésben csak azok a mintermek vesznek részt, amelyeket a logikai függvény tartalmaz. Ennek megfelelően a Karnaugh-táblán történő egyszerűsítésben is csak azok a cellák vesznek részt, amelyekben 1-es szerepel, vagyis amelyekhez tartozó függetlenváltozó-kombinációra a logikai függvény értéke 1.

A Karnaugh-táblán könnyen felismerhetjük az előbbiek alapján a szomszédos mintermeket. Nem kell mást tennünk, mint minden 1-et tartalmazó cellához keresni egy olyan másik cellát, amelyben szintén 1 van és az előbbivel van közös oldala a szélein összefüggőnek tekintett táblán. Ha a szomszédos 1-eseket tartalmazó cellákat egy közös kerettel körülhatároljuk, akkor olyan hurkokat kapunk, amelyek mindegyike két cellát tartalmaz. Nevezzük ezeket *kettes hurkoknak*. Belátható, hogy ezek a kettes hurkok olyan szorzatoknak felelnek meg, amelyek két szomszédos minterm összevonásából keletkeztek, és amelyekben már nem szerepel a két szomszédos mintermet megkülönböztető változó, hiszen itt alkalmazhatjuk a megkülönböztető (pl.: A) változóra vonatkozó $A+\bar{A}=1$ alapösszefüggést. Ezek szerint a kettes hurkoknak megfelelő szorzatok úgy írhatók ki a Karnaugh-táblából algebrai alakban, hogy csak azokat a változókat írjuk le a logikai szorzat tényezőiként, amelyekre a peremezésből a kettes hurok mindkét cellájára nézve azonos logikai érték adódik.

Láttuk, hogy az egyszerűsítési eljárás során a mintermek összevonásából adódó szorzatok között is szomszédosakat kell keresnünk. Ez formailag úgy végezhető a Karnaugh-táblán, hogy a kettes hurkokat tekintjük a továbbiakban celláknak, és ezekből alakítunk ki a lehetőség szerint kettes hurkokat, amelyek az eredeti cellákra (mintermekre) nézve már *négyes hurkok*. Megállapítható, hogy két kettes hurok akkor jelent két szomszédos szorzatot (termet) ha — a Karnaugh-táblát a szélein is összefüggőnek tekintve — van közös oldaluk. Az algebrai alakot szintén úgy írhatjuk ki a Karnaugh-táblából, hogy csak azokat a változókat írjuk le szorzásjellel összekapcsolva, amelyekre a peremezésből a négyes hurokban szereplő mindkét kettes hurokra nézve azonos logikai érték adódik. Teljesen azonos módon folytatható az eljárás a négyes hurkokat tekintve cellának *nyolcas hurkok* kialakítása céljából stb.

Látható, hogy egy hurok, mindig annyi változó elhagyását teszi lehetővé, amennyi a benne szereplő cellák számának kettes alapú logaritmusa, mert k db változó elhagyásához olyan hurkot kell képezni, amely 2^k db cellát tartalmaz. (Az összevonás során csak olyan hurkok keletkeznek, amelyekben szereplő cellák száma 2 egész kitevőjű hatványa.)

A logikai függvény diszjunktív alakja akkor a legegyszerűbb, ha a lehető legkevesebb szorzatot tartalmazza és a szorzatok a lehető legkevesebb változót tartalmazzák. Az algebrai alak akkor teljesen egyenértékű a kiindulási függvénnyel, ha a diszjunktív kanonikus alak minden egyes mintermjét legalább egy szorzat helyettesíti, azaz lefedi.

A legegyszerűbb diszjunktív alakra vonatkozó fenti követelményeket, vagyis a minimalizálás célját a Karnaugh-tábla alapján négynél nem több változó esetén formálisan az alábbiak szerint fogalmazhatjuk meg:

A Karnaugh-táblán az 1-et tartalmazó cellákból a lehető legkevesebb hurkot kell kialakítani úgy, hogy minden 1-et tartalmazó cella legalább egy hurokban szerepeljen és az egyes hurkokban szereplő cellák száma 2 egész kitevőjű hatványa legyen.

Kellő gyakorlat után a kettőnél több cellát tartalmazó hurkok közvetlenül is felismerhetők, így az eljárás rendkívül gyorssá tehető.

Az előzőekben említett prímimplikánsok tehát a végeredményként adódó hurkoknak megfelelő szorzatok.

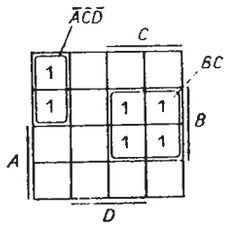
Előfordulhat, hogy a Karnaugh-táblán többféleképpen el lehet végezni a leírt összevonást és a kiadódó algebrai alakok egyszerűség szempontjából egyenértékűek. Az egyes cellák az $A+A=A$ azonosság értelmében többszörösen figyelembe vehetők a hurkok képzésében. Így általában több prímimplikáns adódik, mint amennyi az összes minterm legegyszerűbb lefedéséhez szükséges. Ilyenkor tehát vannak olyan prímimplikánsok, amelyek közül választási lehetőségünk van a legegyszerűbb diszjunktív alak felírásakor. Előfordulhat tehát, hogy több, egyszerűség szempontjából egyenértékű alak képezhető. Lehetnek a függvény Karnaugh-tábláján olyan 1-et tartalmazó cellák, amelyeket az összevonás során csak egyetlen hurokban vettünk figyelembe. Ezeknek olyan mintermek felelnek meg, amelyeket csak egyetlen prímimplikáns tud helyettesíteni azaz lefedni. Ezeket a mintermeket *megkülönböztetett mintermeknek* nevezzük.

Azt a prímimplikánst, amely legalább egy megkülönböztetett mintermet helyettesít, *lényeges prímimplikánsnak* nevezzük. Nyilvánvaló, hogy minden diszjunktív függvényalaknak tartalmaznia kell a lényeges prímimplikánsokat, hiszen ezek nélkül, a megkülönböztetett mintermeknek megfelelő változókombinációk fellépésekor nem lenne helyes a függvény értéke. Az előbbiekben említett, egyszerűség szempontjából egyenértékű, legegyszerűbb diszjunktív függvényalakoknak tehát a lényeges prímimplikánsokban feltétlenül meg kell egyezniük.

Példaként egyszerűsítsük Karnaugh-táblán az

$$F = m_0^4+m_4^4+m_6^4+m_7^4+m_{14}^4+m_{15}^4,$$

Első lépésként megrajzoljuk a négyváltozós Karnaugh-táblát, és egy adott változósorrendet feltételezve ábrázoljuk a függvényt (2.16. ábra).



2.16. ábra. Példa a grafikus minimalizálásra

Ezután kijelöljük a legtöbb egyest tartalmazó hurkokat. Mint az a 2.16. ábrán látható, egy kettes és egy négyes hurok képezhető. Ez a két hurok tartalmazza valamennyi egyest. Az ábrán szereplő Karnaugh-táblában valamennyi minterm megkülönböztetett, mert csak egyetlen hurokban szerepel és mindkét hurok lényeges prímimplikánst jelöl, mert legalább egy olyan minterm van bennük, melyet csak ez a hurok valósít meg.

Az egyszerűsített függvény felírását az egyes hurkokhoz tartozó függetlenváltozókombinációk alapján végezhetjük. Ez úgy történik, hogy megvizsgáljuk, melyek azok a változók, amelyek nem azonos értékkel szerepelnek a hurokba foglalt összes mintermben. Ezeket a változókat az algebrai alakból kihagyjuk, a többi változó pedig azzal az értékével (ponált vagy negált) szerepel az egyszerűsített szorzatban, amely a hurok összes mintermjében azonos.

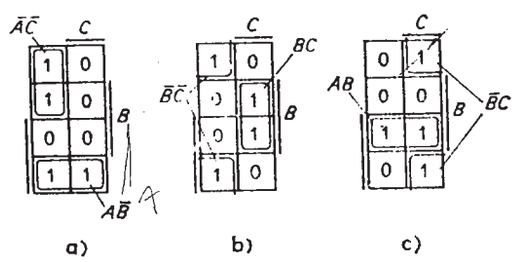
A példában szereplő kettes huroknál elhagyandó változó a B, a négyes huroknál pedig az A és D.

A kettes hurkot az $\bar{A}\bar{C}\bar{D}$, a négyes hurkot a BC szorzat határozza meg. Ennek megfelelően a függvény legegyszerűbb diszjunktív algebrai alakja:

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

A továbbiakban bemutatunk néhány jellegzetes kettes, négyes és nyolcas hurkot, valamint a megfelelő algebrai alakokat.

A háromváltozós logikai függvények esetén kettes és négyes hurkokat mutatunk be, hiszen nyolcas hurok esetén a logikai függvény értéke konstans. A 2.17. ábrán kettes hurkok szerepelnek.



2.17. ábra. Példák kettes hurkokra a háromváltozós Karnaugh-táblán

Az *a)* esetben a logikai függvény algebrai alakja:

$$F_a(A, B, C) = \bar{A}\bar{C}+A\bar{B},$$

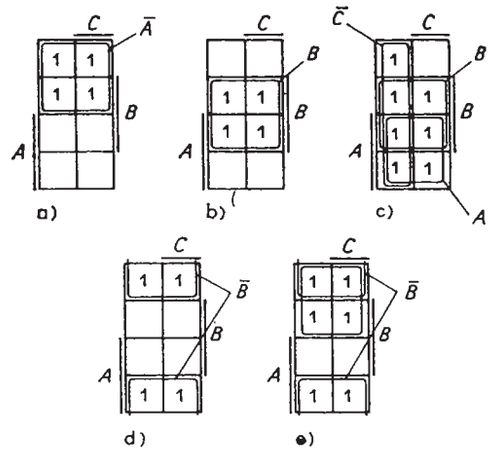
a *b)* esetben:

$$F_b(A, B, C) = \bar{B}\bar{C}+BC = B \odot C,$$

és a *c)* esetben:

$$F_c(A, B, C) = AB+\bar{B}C.$$

A 2.18. ábrán a háromváltozós Karnaugh-táblák közül olyanokat mutatunk be, amelyek négyes hurkot is tartalmaznak. Az *a)* esetben a logikai függvény algebrai



2.18. ábra. Példák négyes hurkokra a háromváltozós Karnaugh-táblán

alakja:

$$F_a(A, B, C) = \bar{A},$$

a *b)* esetben:

$$F_b(A, B, C) = B,$$

a *c)* esetben:

$$F_c(A, B, C) = A+B+\bar{C}.$$

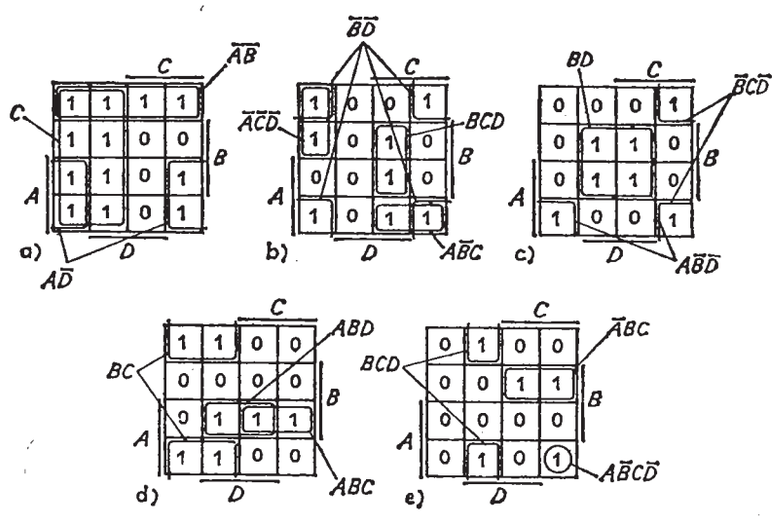
a *d)* esetben:

$$F_d(A, B, C) = \bar{B},$$

az *e)* esetben:

$$F_e(A, B, C) = \bar{A}+\bar{B}.$$

A 2.19. ábrán a négyváltozós Karnaugh-táblán képezhető jellegzetes hurkokat mutatunk be.



2.19. ábra. Példák a négyváltozós Karnaugh-táblán képezhető hurkokra

A logikai függvény egyszerűsített alakja az *a)* esetben:

$$F_a(A, B, C, D) = \bar{C}+\bar{A}\bar{B}+A\bar{D},$$

a *b)* esetben:

$$F_b(A, B, C, D) = \bar{B}\bar{D}+\bar{A}\bar{C}\bar{D}+BCD+A\bar{B}C,$$

a *c)* esetben:

$$F_c(A, B, C, D) = BD+\bar{B}C\bar{D}+A\bar{B}\bar{D},$$

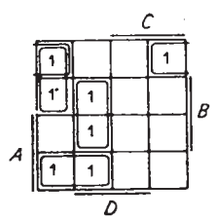
a *d)* esetben:

$$F_d(A, B, C, D) = \bar{B}\bar{C}+ABC+ABD,$$

végül az *e)* esetben:

$$F_e(A, B, C, D) = \bar{B}\bar{C}D+\bar{A}BC+A\bar{B}C\bar{D}.$$

Ezek után példaként írjuk fel a 2.20. ábrán Karnaugh-táblával adott négyváltozós logikai függvény legegyszerűbb diszjunktív algebrai alakját. Az ábrán egyfajta módon



2.20. ábra. Példa négyváltozós függvény egyszerűsítésére

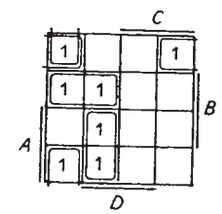
bejelöltük az összevonásokat. Ez alapján az algebrai alak:

$$F = \bar{A}\bar{C}\bar{D}+\bar{A}\bar{B}\bar{D}+B\bar{C}D+A\bar{B}\bar{C}.$$

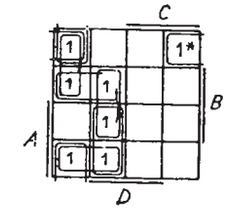
A 2.21. ábrán megfigyelhetjük, hogy egyszerűség szempontjából a fentivel egyenértékű algebrai alakhoz juthatunk másfajta hurokelrendeződés kialakításával is. A 2.21. ábrán bejelölt hurkok alapján:

$$F = \bar{B}\bar{C}\bar{D}+\bar{A}\bar{B}\bar{D}+\bar{A}B\bar{C}+A\bar{C}D.$$

Az egyenértékű megoldások létezését nyilvánvalóan az magyarázza, hogy az egyszerűsítendő függvény jellegéből következően a képezhető összes prímimplikáns közül különböző módokon választhatjuk ki azokat, amelyeket a legegyszerűbb diszjunktív alakban figyelembe veszünk. Ennek szemléltetése céljából a 2.22. ábrán



2.21. ábra. Példa a 2.20. ábrával egyenértékű összevonásra

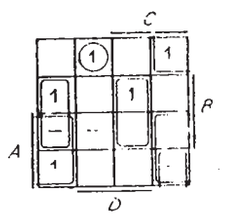


2.22. ábra. Az összes képezhető prímimplikáns szemléltetése

egy táblán láthatók az összes prímimplikánsnak megfelelő hurkok. A függvény egyetlen megkülönböztetett mintermjét *-gal jelöltük. Az ezt tartalmazó $\bar{A}\bar{B}\bar{D}$ prímimplikáns lényeges, emiatt szerepelnie kell az összes diszjunktív algebrai alakban. Az egyenértékű alakok csupán a többi — nem lényeges — prímimplikánsban különbözhetnek egymástól aszerint, hogy a mintermek lefedéséhez hogyan válogattunk a prímimplikánsok közül. A figyelembe veendő nem lényeges prímimplikánsok kiválasztása nem mindig olyan egyszerű és szemléletes, mint a bemutatott példában. Sok esetben igen nehéz próbálgatással eljutni a legegyszerűbb diszjunktív alakhoz. A későbbiekben ezért szisztematikus eljárást mutatunk be az összes mintermnek prímimplikánsokkal történő ún. *optimális lefedésére,* ami a példában bemutatott helyzetekben kiküszöböli a próbálgatásos jelleget és az összes egyenértékű megoldást szolgáltatja. A 2.22. ábra alapján megállapítható, hogy a bemutatott egyenértékű megoldásokon kívül próbálgatással még további egyenértékű megoldásokhoz juthatunk. Bonyolultabb esetben a sok választási lehetőség miatt próbálgatással nemcsak az összes egyenértékű megoldás képzése nehéz, hanem még az is nehezen ítél-

hető meg, hogy egy megoldás valóban a legegyszerűbb diszjunktív algebrai alakot eredményezi-e, azaz a lefedés optimális-e.

Vizsgáljuk meg ezek után, hogy a bemutatott grafikus minimalizálási eljárás milyen módon alkalmas a nem teljesen határozott logikai függvények egyszerűsítésére. A 2.23. ábrán Karnaugh-táblájával megadott nem teljesen határozott (specifikált) logikai függvénynek a közömbös függvényértékhez tartozó mintermjeit a táblán vízszintes vonallal jelöltük. Az összevonás során, azaz a lefedő hurkok kialakításakor a nem rögzített függvényértékeket tetszés szerint választhatjuk 1-nek vagy 0-nak attól függően, hogy melyik választás adja a legkedvezőbb megoldást.



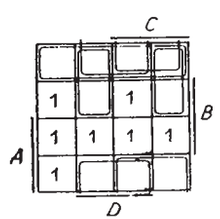
2.23. ábra. Példa nem teljesen határozott logikai függvény egyszerűsítésére

Így a 2.23. ábrán bemutatott megoldásban azokat a határozatlan függvényértékeket, amelyek a kialakított hurkokon belül vannak 1 értékűre, azokat pedig, amelyek a hurkokon kívül maradtak 0 értékűre rögzítettük. Az egyszerűsített függvény tehát:

$$F = A\bar{D} + B\bar{C}\bar{D} + BCD + \bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}D.$$

A Karnaugh-tábla segítségével tehát szemléletes módon végezhetjük el a közömbös függvényértékek legkedvezőbb rögzítését, hiszen a legelőnyösebb hurokelrendeződés kialakításával kell csupán törődnünk, a közömbös bejegyzések helyén egyaránt megengedhetjük akár a logikai 0, akár a logikai 1 értéket.

Az eddigiekben mindig a legegyszerűbb diszjunktív alakot írtuk fel a Karnaugh-tábla alapján. Az alábbiakban bemutatjuk, hogy legegyszerűbb konjunktív algebrai alakot is könnyen képezhetjük a Karnaugh-táblán. Induljunk ki a 2.24. ábrán meg-



2.24. ábra. Példa a legegyszerűbb konjunktív függvényalak képzésére

adott függvényből, és írjuk fel a függvény tagadottjának legegyszerűbb diszjunktív alakját. Mivel az adott függvény teljesen határozott, a tagadott függvény azokat és csak azokat a mintermeket tartalmazza, amelyeket a függvény nem tartalmaz. A tagadott függvényre vonatkozó összevonást a Karnaugh-táblán tehát azokra a cellákra kell elvégezni, amelyekben nincs 1-es bejegyzés. A 2.24. ábra alapján

így felírhatjuk a függvény negáltjának legegyszerűbb algebrai alakját:

$$\bar{F} = \bar{A}\bar{B} + \bar{B}D + \bar{A}\bar{C}D + \bar{A}C\bar{D} + \bar{B}C.$$

Ismételt tagadással visszajuthatunk a kiindulási függvényhez:

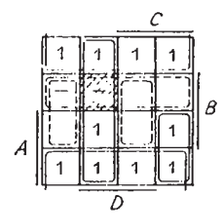
$$F = \overline{\bar{A}\bar{B} + \bar{B}D + \bar{A}\bar{C}D + \bar{A}C\bar{D} + \bar{B}C}.$$

Alkalmazva a De Morgan-azonosságot:

$$F = (A+B)(B+\bar{D})(A+C+\bar{D})(A+\bar{C}+D)(B+\bar{C}).$$

A kapott függvényalak összegek szorzata, és nem más, mint a legegyszerűbb konjunktív alakja a kiindulási függvénynek. Ha megfigyeljük az egyes összegekben szereplő változókat, kiderül, hogy a legegyszerűbb konjunktív alakot a Karnaugh-táblából közvetlenül is kiolvashatjuk. A 2.24. ábrán képzett hurkoknak megfelelő változókombinációk ugyanis éppen a konjunktív alak összegeiben szereplő változókombinációkkal egyeznek meg, ha a Karnaugh-tábla peremezését az eddigiekkel ellentétesen értelmezzük, azaz akkor írunk ponált változót, amikor a peremezés negáltat írna elő és akkor negáltat, amikor a peremezés alapján ponált adódna.

Ha a függvény nem teljesen határozott, akkor természetesen előfordulhat, hogy a közömbös bejegyzéseket máshogy célszerű rögzíteni a legegyszerűbb konjunktív alak képzésekor, mint ahogyan azt a legegyszerűbb diszjunktív alak képzésekor tennénk. Ezt szemlélteti a 2.25. ábra. A folytonos vonallal berajzolt hurkok a függ-



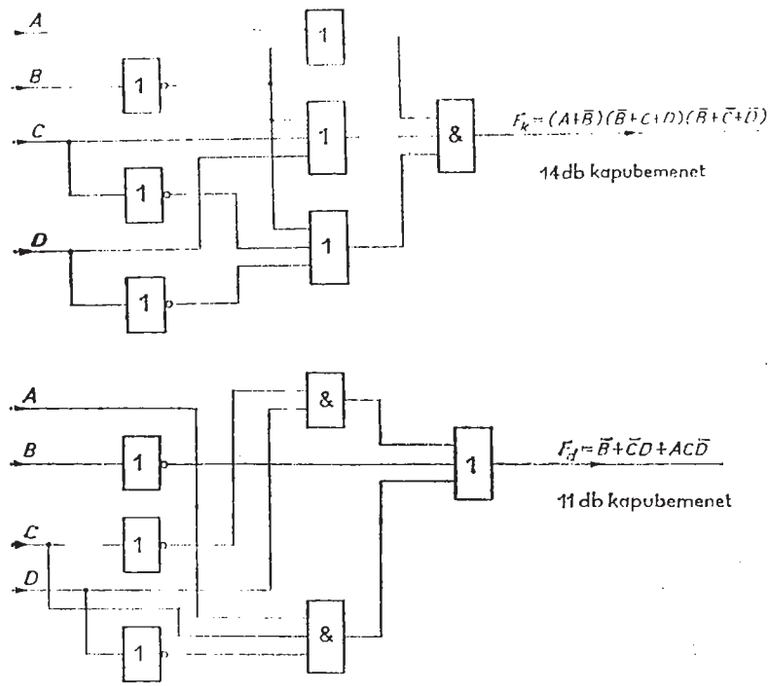
2.25. ábra. Példa a közömbös függvényértékek eltérő rögzítésére a legegyszerűbb diszjunktív és a legegyszerűbb konjunktív algebrai alakok képzésekor

vény, a szaggatott vonallal berajzoltak pedig a tagadott függvény legegyszerűbb diszjunktív alakját ábrázolják. Nyilvánvaló, hogy a tagadott függvény elnevezés itt már magyarázatra szorul. Látható ugyanis, hogy a 2.25. ábrán a vonalkázott cellában levő határozatlan függvényértéket a függvény képzésekor 1-nek, a tagadott függvény képzésekor pedig 0-nak választottuk a lehető legegyszerűbb hurokelrendeződés kialakítása céljából. Így a tagadott függvény értéke nem minden bemeneti kombináció mellett lesz ellentétes értékű a függvény értékével. Ilyen okok miatt általánosságban is kimondhatjuk, hogy a nem teljesen határozott logikai függvények esetében a legegyszerűbb diszjunktív és a legegyszerűbb konjunktív függvényalakból eltérő függvényértéket kaphatunk azoknak a változókombinációknak a fellépésekor, amelyekhez közömbös függvényérték tartozik. Példánkban a két függvényalak:

$$F_d = \bar{B} + \bar{C}D + AC\bar{D},$$

$$F_k = (A+B)(\bar{B}+C+D)(B+C+\bar{D}).$$

Könnyen ellenőrizhető, hogy a vonalkázott cellának megfelelő $\bar{A}B\bar{C}D$ változókombináció esetén $F_d=1$ és $F_k=0$. A példa alapján az is megfigyelhető, hogy F_d ezúttal egyszerűbb kétszintű hálózatot eredményez, mint F_k. A 2.26. ábrán ezt szemléltetik az elvi logikai rajzok.



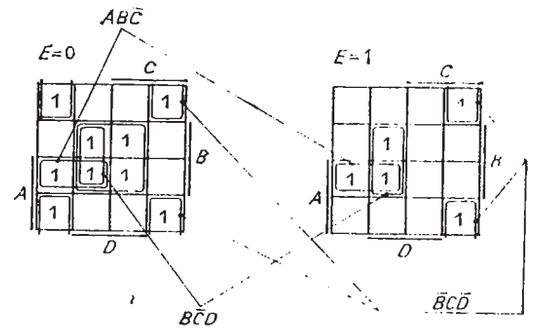
2.26. ábra. A 2.25. ábra alapján képzett függvényalakoknak megfelelő elvi logikai rajzok

A diszjunktív és a konjunktív legegyszerűbb alakok által kapott elvi logikai rajzok nemcsak akkor különbözhetnek egymástól a kapubemenetek számában, ha az egyszerűsítendő függvény nem teljesen határozott. Például a 2.24. ábrán szereplő teljesen határozott függvény esetében is meggyőződhetnénk erről, ha képeznénk a függvény legegyszerűbb diszjunktív alakját is. Így, ha a megvalósításkor szabadon választhatunk a diszjunktív és konjunktív alak között, azaz nincs megkötés az ÉS és a VAGY szintek sorrendjére, akkor a legkevesebb kapubemenetet igénylő megoldást csak a kétfajta függvényalak próbálgatásos jellegű elemzése révén kaphatjuk meg.

A grafikus minimalizálás során az eddigiekben feltételeztük, hogy a változók száma négynél nem nagyobb. Ötváltozós esetben az eljárást két négyváltozós Karnaugh-tábla segítségével értelmezhetjük. Bármely ötváltozós függvényt ugyanis egyértelműen ábrázolhatunk két négyváltozós Karnaugh-táblán. Az egyik tábla cellái azoknak az ötváltozós mintermeknek felelnek meg, amelyekben az ötödik változó tagadott alakban, a másik tábla cellái pedig azokat az ötváltozós mintermeket képviselik, amelyekben az ötödik változó tagadás nélküli alakban fordul elő. A 2.27. ábrán az

$$F=\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}+\bar{A}\bar{B}CD\bar{E}+\bar{A}B\bar{C}D\bar{E}+\bar{A}BCD\bar{E}+AB\bar{C}D\bar{E}+ABCD\bar{E}+$$
$$+AB\bar{C}\bar{D}\bar{E}+A\bar{B}\bar{C}\bar{D}\bar{E}+ABC\bar{D}\bar{E}+AB\bar{C}\bar{D}E+AB\bar{C}DE+\bar{A}B\bar{C}DE+$$
$$+\bar{A}\bar{B}C\bar{D}E+A\bar{B}C\bar{D}E$$

függvény ábrázolását és egyszerűsítését követhetjük ilyen módon. Az ábrázolásmódból következően az eddigi szomszédossági helyzeteken kívül azokat a mintermeket, ill. szorzatokat is szomszédosoknak kell tekintenünk, amelyek a két azonos peremezésű



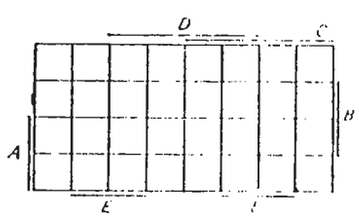
2.27. ábra. Példa ötváltozós függvény ábrázolására és egyszerűsítésére

tábla azonos $ABCD$ kombinációjú cellájához tartoznak, hiszen ezek csak az E változó értékében különböznek egymástól. Ennek figyelembevételével az egyik lehetséges megoldást a 2.27. ábrán követhetjük. Az ábrán összekötő vonalak jelölik azokat a hurkokat, amelyek a másik táblán levő hurkokkal vonhatók össze és így olyan prímimplikánsokat hoznak létre, amelyekben az E változó nem szerepel. Az így elvégzett összevonás eredménye:

$$F=AB\bar{C}+B\bar{C}D+\bar{B}C\bar{D}+BD\bar{E}+\bar{B}\bar{D}\bar{E}.$$

Ötváltozós esetben a két négyváltozós táblát a peremezés megváltoztatásával egybefüggőnek is tekinthetjük a szomszédossági viszonyok még könnyebb felismerése céljából. Egy ilyen megoldást szemléltet a 2.28. ábra.

Hat változó esetében a függvény ábrázolásához négy négyváltozós Karnaugh-tábla szükséges. A hat változó közül kettőnek az értékét kell ugyanis rögzítettnek tekintenünk egy-egy táblán. Mivel két változónak négy különböző értékkombinációja

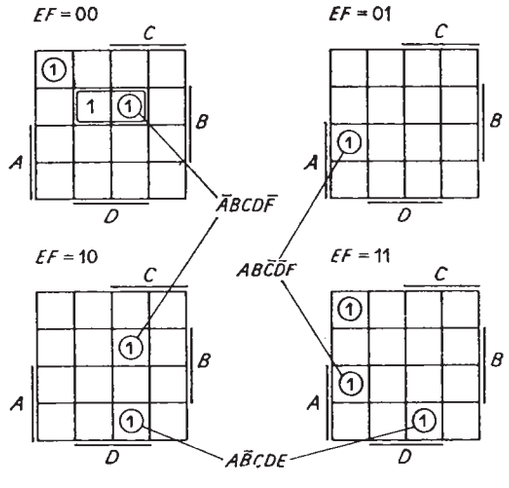


2.28. ábra. Példa ötváltozós Karnaugh-tábla peremezésére

lehet, ezért szükséges négy tábla. A 2.29. ábrán példaként az

$$F = \bar{A}B\bar{C}D\bar{E}\bar{F}+\bar{A}BCD\bar{E}\bar{F}+AB\bar{C}\bar{D}\bar{E}F+\bar{A}BCDE\bar{F}+A\bar{B}CDE\bar{F}+$$
$$+AB\bar{C}\bar{D}EF+\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}+\bar{A}\bar{B}\bar{C}\bar{D}EF+A\bar{B}CDEF$$

függvény egyszerűsítését követhetjük.



2.29. ábra. Példa hatváltozós függvény ábrázolására és összevonására négy Karnaugh-tábla segítségével

A hurkok képzésekor azokat a mintermeket, ill. szorzatokat kell szomszédosaknak tekintenünk, amelyek két szomszédos EF kombinációjú tábla azonos $ABCD$ kombinációjú cellájához tartoznak. A bejelölt összevonások alapján a megoldás:

$$F = \bar{A}BD\bar{E}\bar{F}+\bar{A}BCD\bar{F}+AB\bar{C}\bar{D}F+A\bar{B}CDE+\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}+\bar{A}\bar{B}\bar{C}\bar{D}EF.$$

Bonyolultabb esetekben a négy tábla páronkénti áttekintése már nem könnyű, különösen a figyelembe veendő nem lényeges prímimplikánsok kiválasztása okoz nehézséget. Ezért hat vagy annál több változó esetén az ismertetett grafikus minimalizálási módszer nem előnyös. Öt változóig azonban gyors és szemléletes, kis gyakorlattal az egyenértékű legegyszerűbb megoldások közvetlenül kiolvashatók a Karnaugh-táblából.

2.3.2. A számjegyes minimalizálás (Quine—McCluskey-módszer)

Ha az egyszerűsítés során a mintermeket a Karnaugh-táblán történő ábrázolás helyett az alsó indexeikkel helyettesítjük és minden következtetést ezekből a számokból próbálunk levonni, akkor olyan minimalizáló eljáráshoz juthatunk, amelynek végrehajthatósága nem függ a változók számától.

Mint láttuk, a mintermek alsó indexei decimális számok, amelyek a minterm által képviselt változókombinációknak megfelelő bináris számok decimális alakra történő

átírása révén adódtak. Vizsgáljuk meg, hogy miként lehet a mintermek, ill. szorzatok szomszédosságvizsgálatát és a prímimplikánsok képzését elvégezni kizárólag ezeknek a decimális számoknak az ismeretében. Ehhez először azt kell megállapítanunk, hogy milyen feltételek teljesülése esetén képvisel két decimális szám szomszédos mintermet.

a) Könnyen beláthatjuk, hogyha két minterm szomszédos, akkor alsó indexeik különbsége 2 egész kitevőjű hatványa. A szomszédosság ugyanis úgy is fogalmazható, hogy a két mintermnek megfelelő bináris számok minden számjegye egy kivételével valamennyi helyértéken azonos a két számban. Ha a nagyobbik bináris számból kivonjuk a kisebbiket, akkor a különbség bináris alakjában csak egyetlen helyértéken van 1, a többin 0. Például az m_2^4 és m_6^4 szomszédos mintermek esetén az alsó indexek bináris megfelelőinek különbsége:

$$\begin{array}{rr} 6 & 0110 \\ -2 & -0010 \\ \hline 4 = 2^2 & 0100 \end{array} \qquad \left.\begin{array}{l}\bar{A}BC\bar{D}\\ \bar{A}\bar{B}C\bar{D}\end{array}\right\}\rightarrow \bar{A}C\bar{D}.$$

A fenti feltétel azonban csak szükséges, de nem elégséges a szomszédosság megállapítására. Például az m_2^4 és m_4^4 mintermekre teljesül a feltétel, mégsem szomszédosak:

$$\begin{array}{rr} 4 & 0100 \\ -2 & -0010 \\ \hline 2 = 2^1 & 0010 \end{array} \qquad \begin{array}{l}\bar{A}B\bar{C}\bar{D}\\ \bar{A}\bar{B}C\bar{D}\end{array}$$

b) Ha két minterm szomszédos, akkor az egyiknek megfelelő bináris szám eggyel és csakis eggyel több 1-et tartalmaz, mint a másiknak megfelelő. Ezt a magától értetődő kijelentést is szemlélteti a korábbi példa az m_2^4 és m_6^4 mintermekre. A mintermeknek megfelelő bináris számokban szereplő 1-ek számát a továbbiakban a mintermek *bináris súlyának* nevezzük. Így tehát, ha a mintermek szomszédosak, akkor bináris súlyaik különbsége 1. Az m_2^4 és m_4^4 mintermek esetében éppen az a feltétel nem teljesült.

Az *a)* és *b)* feltétel egyidejű teljesülése esetén is még csak szükséges feltételét kaptuk két minterm szomszédosságának. Például az m_9^4 és m_7^4 mintermekre az *a)* és *b)* feltétel egyaránt teljesül, a két minterm mégsem szomszédos:

$$\begin{array}{rr} 9 & 1001 \\ -7 & -0111 \\ \hline 2 = 2^1 & 0010 \end{array} \qquad \begin{array}{l}A\bar{B}\bar{C}D\\ \bar{A}BCD\end{array}$$

c) Ha két minterm szomszédos, akkor a nagyobb bináris súlyúnak a decimális indexe is nagyobb. Az *a)* pontban bemutatott példa az m_2^4 és m_6^4 mintermekre ezt a feltételt is szemlélteti. A *b)* pontban bemutatott ellenpéldában éppen ez a feltétel nem teljesült.

Bizonyítható, hogy az *a)*, *b)* és *c)* pontban leírt feltételek együttesen két minterm szomszédosságának szükséges és elégséges feltételét adják. Eszerint tehát két minterm akkor és csak akkor szomszédos, ha

— decimális indexeik különbsége 2 egész kitevőjű hatványa,
— bináris súlyaik különbsége 1, és
— a nagyobb bináris súlyú mintermnek a decimális indexe is nagyobb a másikénál.

A szomszédossági feltétel megfogalmazása után módszert kell találnunk a mintermek páronkénti vizsgálatára decimális indexeik alapján. A grafikus minimalizáláshoz hasonlóan most is valamelyik normálalakból indulunk ki. A továbbiakban használni fogjuk a normálalakok megadására az alábbi szimbolikus jelöléseket:

$$F = \sum^{4} [(0, 1, 3, 7, 11, 15) + (2, 13)],$$

$$F = \prod^{5} [(0, 1, 3, 17, 19, 30) + (21, 28, 31)],$$

ahol

a $\left.\begin{matrix}\sum \\ \prod\end{matrix}\right\}$ jel jelöli, hogy $\left.\begin{matrix}\text{diszjunktív} \\ \text{konjunktív}\end{matrix}\right\}$ alakról van szó, a $\left.\begin{matrix}\sum \\ \prod\end{matrix}\right\}$ jel fölé írt szám jelöli a változók számát, az első zárójelben levő számok jelölik a függvény $\left.\begin{matrix}\text{diszjunktív} \\ \text{konjunktív}\end{matrix}\right\}$ normálalakjában szereplő $\left.\begin{matrix}\text{mintermek} \\ \text{maxtermek}\end{matrix}\right\}$ decimális indexeit, a második zárójelben levő számok jelölik a nem teljesen határozott függvény közömbös értékéhez tartozó $\left.\begin{matrix}\text{mintermeket} \\ \text{maxtermeket}\end{matrix}\right\}$.

A számjegyes minimalizálás bemutatásához induljunk ki egy négyváltozós függvény diszjunktív normálalakjából. A kis változószám miatt a számjegyes eljárás lépéseit Karnaugh-táblán is szemléltethetjük, és így az eljárás lényege egyszerűbben magyarázható. Az egyszerűsítendő függvény a fentiekben ismertetett szimbolikus megadásban:

$$F = \sum^{4} (0, 1, 3, 7, 11, 12, 14, 15),$$

vagyis a függvény teljesen határozott.

A páronkénti szomszédosságvizsgálathoz csoportosítsuk a felsorolt decimális indexeket bináris súlyuk szerint, a különböző bináris súlyokhoz tartozókat vízszintes vonallal elválasztva egymástól.

bináris kombináció	bináris súly	decimális index
0000	0	0
0001	1	1
0011	2	3
1100	2	12
0111	3	7
1011	3	11
1110	3	14
1111	4	15

Az indexek ilyen csoportosítása azért előnyös, mert a párbaválogatáskor lényegesen kevesebb összehasonlítást kell végeznünk, mintha azt más sorrendben kísérelnénk meg. Az egyes csoportokban levő indexeknek ugyanis csak az eggyel nagyobb vagy eggyel kisebb súlyú csoportban lehet szomszédjuk. Így tehát a párba válogatást például úgy végezhetjük, hogy az egyes csoportok minden egyes számjegyét kivonjuk az eggyel nagyobb súlyú csoport minden egyes számjegyéből. Ha találunk két olyan számot, amelynek különbsége kettő egész kitevőjű hatványa, akkor a két szám mellé „√" jelet teszünk, ezzel azt jeleztük, hogy e mintermeket már tartalmazza legalább egy szorzat. Új oszlopot (II. oszlop) kezdünk, amelyben az előbbi összevont számpár elemeit növekvő sorrendben egymás mellé írjuk, valamint zárójelben feltüntetjük a különbségüket. Ez utóbbi szám kettes alapú logaritmusa jelöli ki az elhagyható változó helyértékét, hiszen ebből állapítható meg, hogy melyik az a helyérték, ahol a két bináris kombináció eltér, vagyis melyik az a változó, amely az összevont szorzatból elhagyható. Egy számot elegendő csak egyszer megjelölni az összevonás során, hiszen csupán azt kell jelölnünk, hogy keletkezett olyan szorzat, amely a számnak megfelelő mintermet tartalmazza

I.	II.	III.
0 √	0, 1 (1) *a*	3, 7, 11, 15 (4, 8) *e*
1 √	1, 3 (2) *b*	
3 √	3, 7 (4) √	
12 √	3, 11 (8) √	
7 √	12, 14 (2) *c*	
11 √	7, 15 (8) √	
14 √	11, 15 (4) √	
15 √	14, 15 (1) *d*	

Az I. oszlop első és második csoportjának összehasonlításaként kapjuk a II. oszlop első csoportját, az I. oszlop második és harmadik csoportjából adódik a II. oszlop második csoportja stb.

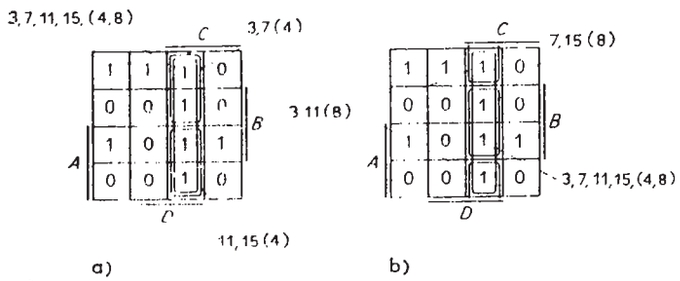
Az I. oszlop elemeinek párba válogatásából egyrészt kialakul a II. oszlop, másrészt az I. oszlopban meg nem jelölt számoknak megfelelő mintermek biztosan prímimplikánsok lesznek, hiszen mivel nem akadt szomszédjuk, nem hagyható el belőlük egyetlen változó sem. Nyilvánvaló, hogy ezek a kanonikus szorzatok (mintermek) egyben megkülönböztetett mintermek is.

Példánkban az I. oszlop egyik eleme sem prímimplikáns. A II. oszlopban levő számpárok a Karnaugh-táblán kialakítható összes lehetséges kettes hurkoknak felelnek meg.

A III. oszlop kialakítását az előzőével azonos módon végezzük. Minden elemet összehasonlítunk a következő csoport minden elemével. Két, már egyszerűsített

szorzat akkor lesz szomszédos, ha a decimális indexek különbsége páronként (az első elemek különbsége, ill. a második elemek különbsége) kettőnek ugyanaz az egész kitevőjű hatványa, valamint a zárójelbe írt számok megegyeznek. Az utóbbi feltétel azt jelenti, hogy mindkét, már egyszerűsített szorzatból ugyanannak a változónak kell hiányoznia ahhoz, hogy további összevonásokat lehessen végezni.

A példánkban szereplő III. oszlopban csak egy elem lesz, de az a II. oszlop két-két elempárjának az összevonásából is létrejöhet. Természetesen ilyen esetben mind a négy elemet meg kell jelölni a II. oszlopban. Minden oszlopban a jelölés nélkül maradó elemek prímimplikánsok, mert azoknak nem adódott szomszédjuk, vagyis további változót nem lehetett elhagyni a nekik megfelelő szorzatokból. Ezeket a prímimplikánsokat kisbetűvel is megjelöltük. A III. oszlop egyetlen elemének kialakítását bemutatjuk Karnaugh-táblán is (2.30. ábra). Az *a)* Karnaugh-táblán ábrázoltuk a logikai függvényt, és berajzoltuk a 3, 7 (4)-nek és 11, 15 (4)-nek megfelelő kettes, valamint ezek egyesítéséből adódó 3, 7, 11, 15 (4, 8)-nak megfelelő négyes hurkot is. A *b)* Karnaugh-táblán pedig a 3, 11 (8)-nak és 7, 15 (8)-nak megfelelő négyes hurkot tüntettük fel. Látható, hogy mindkét esetben ugyanaz a négyes hurok keletkezett.



2.30. ábra. A számjegyes minimalizálás szemléltetése Karnaugh-táblán

A 3, 7, 11, 15 (4, 8)-nak megfelelő szorzat alakja a Karnaugh-táblából könnyen kiolvasható: CD.

Ez a szorzat természetesen a 3, 7, 11, 15 (4, 8) szimbolikus jelölési módból is kiolvasható. A decimális indexeknek megfelelő mintermek az alábbiak:

$$2^3 2^2 2^1 2^0$$

$$m_3^4 = \bar{A}\bar{B}CD$$

$$m_7^4 = \bar{A}BCD$$

$$m_{11}^4 = A\bar{B}CD$$

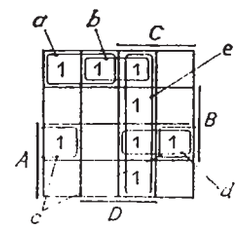
$$m_{15}^4 = ABCD$$

Ha a fenti mintermek bármelyikéből a $2^2=4$ és a $2^3=8$ helyértékeken levő változókat A-t és B-t elhagyjuk, akkor valóban a CD szorzatot kapjuk. A továbbiakban elegendő tehát a számcsoport egyetlen elemét figyelembe venni az egyszerűsített szorzat algebrai alakjának meghatározásához, például a 3, 7, 11, 15 (4, 8)-ból a 3 (4, 8)-at vesszük figyelembe:

$$3(4, 8) \rightarrow \bar{A}\bar{B}CD$$

Ez tehát azt jelenti, hogy a 3 indexű mintermből ($\bar{A}\bar{B}CD$) a $4=2^2$ és $8=2^3$-nak megfelelő $\bar{A}$ és $\bar{B}$ tényezőket hagyjuk el.

A 2.31. ábrán Karnaugh-táblán szemléltettük a II. és III. oszlopban kapott prímimplikánsokat. Látható, hogy a legegyszerűbb diszjunktív alak meghatározásához még el kell végezni a prímimplikánsok válogatását, azaz az optimális lefedést.



2.31. ábra. A számjegyes minimalizálás során kapott prímimplikánsok szemléltetése Karnaugh-táblán

Példánkban a Karnaugh-tábla jól szemlélteti, hogy a *b* és *d* prímimplikánsok feleslegesek a lefedéshez. Bonyolultabb esetben és amikor a függvényt nem ábrázolhatjuk Karnaugh-táblán, akkor az optimális lefedést is a decimális indexek alapján kell elvégeznünk. Ehhez nyújt segítséget az ún. *prímimplikánstábla,* amely azt foglalja össze, hogy az egyes kanonikus termeket (mintermeket) mely prímimplikánsok tartalmazzák, vagyis fedik le. A prímimplikánstábla annyi sorból áll, ahány prímimplikáns keletkezett az összevonás során és annyi oszlopa van, ahány kanonikus term alkotja az egyszerűsítendő logikai függvényt. A példánkra vonatkozó prímimplikánstábla a 2.32. ábrán látható. A táblázat kitöltése úgy történik, hogy egy-egy prímimplikánssal kijelölt sornak abba a rovatába írunk x jelet, amelyhez tartozó mintermet az illető prímimplikáns tartalmazza.

A 2.32. ábrán látható, hogy van olyan minterm, amely alatt csak egyetlen x jel van. Ez azt jelenti, hogy a minterm megkülönböztetett és az azt tartalmazó prím-

Mintermek / Prímimplikánsok		0	1	3	7	11	12	14	15
0,1 (1)	a^*	X	X						
1,3 (2)	b		X	X					
12,14 (2)	c^*						X	X	
14,15 (1)	d							X	X
3,7,11,15 (4,8)	e^*			X	X	X			X

2.32. ábra. A prímimplikánstábla felépítése

implikáns lényeges prímimplikáns, tehát a logikai függvény egyszerűsített alakjában feltétlenül szerepelnie kell. Példánkban ilyen lényeges prímimplikáns a 0, 1 (1), a 12, 14 (2) és a 3, 7, 11, 15 (4, 8). Ezeket * jellel meg is jelöltük a prímimplikánstáblán. A prímimplikánstáblából egyszerű esetben azonnal látszik, hogy mely prímimplikánsok valósítják meg legegyszerűbben a függvényt. A 2.32. ábrából valóban könnyen kiolvasható, hogy az *a*, *c* és *e* prímimplikánsok logikai összege a logikai függvényben szereplő összes mintermet képes helyettesíteni.

Bonyolultabb esetben sokkal nehezebb a táblázatot áttekinteni a legegyszerűbb alak felírása céljából. A továbbiakban egy módszert ismertetünk a legegyszerűbb alak felírására a prímimplikánstábla alapján.

A legegyszerűbb alak felírásakor a prímimplikánsok közül általában választási lehetőségünk van. A prímimplikánsok logikai összegével képzett bármilyen függvényalak csak akkor egyenértékű a kiindulási függvénnyel, ha a kiindulási függvény minden egyes mintermjét a logikai összegben szereplő prímimplikánsok közül legalább egy helyettesíti. A prímimplikánstáblából rendre kiolvasható, hogy melyek azok a prímimplikánsok, amelyeknek az egyes mintermek helyettesítéséhez, azaz lefedéséhez szerepelniük kell a prímimplikánsok logikai összegével alkotott függvényalakban. A 2.32. ábrán szereplő tábla alapján pl. megállapítható, hogy a 0 minterm megkülönböztetett minterm, ezért az *a* prímimplikánsnak mindenképpen szerepelnie kell a logikai összegben. Az 1 minterm helyettesítéséhez az *a* vagy *b* prímimplikánsok szükségesek.

Hasonló módon folytatva:

3 jelű minterm helyettesítéséhez *b* vagy *e* szükséges,
7 jelű minterm helyettesítéséhez *e* szükséges,
11 jelű minterm helyettesítéséhez *e* szükséges,
12 jelű minterm helyettesítéséhez *c* szükséges,
14 jelű minterm helyettesítéséhez *c* vagy *d* szükséges,
15 jelű minterm helyettesítéséhez *d* vagy *e* szükséges.

A legegyszerűbb diszjunktív alak felírása céljából tehát az összes prímimplikáns közül úgy kell kiválasztani a lehető legkevesebbet, hogy a belőlük képzett logikai összeg a felsorolt feltételek mindegyikét kielégítse. A felsorolt feltételeknek tehát ÉS kapcsolatban kell lenniük egymással az összes kiindulási minterm helyettesítése céljából. Az összes kiindulási minterm helyettesítésének feltételét logikai függvény alakjában is megfogalmazhatjuk. Ennek a definiált segédfüggvénynek csak akkor legyen 1 az értéke, ha a prímimplikánsok logikai összege az összes felsorolt feltételt teljesíti, vagyis az összes kiindulási mintermet helyettesíti. A segédfüggvény logikai változóit jelöljük a prímimplikánsok betűjelével. A változók értéke akkor és csak akkor legyen 1, ha az illető prímimplikáns szerepel a diszjunktív alakban. Így a fentiekben megfogalmazott feltételt kifejező segédfüggvény algebrai alakját úgy kapjuk meg, hogy az egyes mintermek lefedésének fenti feltételeit ÉS kapcsolatba hozzuk.

$$s = a \cdot (a+b) \cdot (b+e) \cdot e \cdot e \cdot c \cdot (c+d) \cdot (d+e),$$

ahol *s* jelöli az összes kiindulási minterm helyettesítésének feltételét jelképező logikai függvény értékét, a jobb oldalon szereplő változók pedig az előbbiek értelmében az egyes prímimplikánsok jelenlétét jelentik a prímimplikánsok logikai összegével felírt függvényalakban.

Az *s* segédfüggvény bevezetésének az a nagy előnye, hogy algebrai alakban fogalmaztuk meg azt a feltételt, amelyet minden, prímimplikánsok logikai összegéből álló függvényalaknak, így a legegyszerűbb diszjunktív alaknak is teljesítenie kell ahhoz, hogy az egyszerűsítendő logikai függvénnyel egyenértékű legyen. Szavakban tehát: *s* értéke akkor és csak akkor 1, ha a prímimplikánsok logikai összegében szerepel

a prímimplikáns és
a vagy *b* prímimplikáns és
b vagy *e* prímimplikáns és
e prímimplikáns és
e prímimplikáns és
c prímimplikáns és
c vagy *d* prímimplikáns és
d vagy *e* prímimplikáns.

Ha ezek után meghatározzuk az *s* függvény diszjunktív alakját, akkor az ebben szereplő szorzatok mindegyike természetesen előidézheti $s=1$ értéket.

Ha ugyanis valamelyik szorzatban szereplő összes változónak megfelelő prímimplikánsokat tartalmazza a kiindulási függvénynek a prímimplikánsok logikai összegéből álló alakja, vagyis formailag a szorzatban szereplő összes logikai változó értéke 1, akkor az illető szorzat 1 értékű, ami $s=1$-et eredményez.

A kiindulási függvény legegyszerűbb diszjunktív alakját akkor kapjuk meg, ha a lehető legkevesebb prímimplikánst hozzuk VAGY kapcsolatba úgy, hogy *s* értéke 1 legyen. Könnyen belátható ezek után, hogy az ilyen feltételnek eleget tevő függvényalakot az *s* függvény diszjunktív alakjából kiolvashatjuk. Elegendő ugyanis *s* diszjunktív alakjában megkeresni azt a szorzatot, amelyik a legkevesebb tényezőt tartalmazza. Azokat a prímimplikánsokat kell VAGY kapcsolatba hozni a legegyszerűbb diszjunktív alakban, amelyeknek megfelelő változók ebben a szorzatban szerepelnek, hiszen ezek együttes jelenléte esetén $s=1$ lesz. Az is megállapítható az *s* függvény diszjunktív alakja alapján, hogy kevesebb prímimplikánssal semmiképpen sem tudjuk a kiindulási függvény összes mintermjét helyettesíteni, hiszen az *s* kifejezésében szereplő szorzatok közül a kiválasztott szorzat tartalmazza a legkevesebb változót.

Általában előfordul, hogy *s* diszjunktív alakjában több olyan szorzat található, amelyek mindegyike azonos számú tényezőt tartalmaz és nem található ezeknél kevesebbet tartalmazó szorzat. Ilyenkor ezek közül azt a szorzatot kell kiválasztanunk, amelyben a tényezők által képviselt prímimplikánsok a lehető legegyszerűbbek, azaz a lehető legkevesebb változót tartalmazzák. Ha a szorzatok ilyen szempontból is egyenértékűek, akkor azok alapján több — az egyszerűség szempontjából —

egyenértékű legegyszerűbb diszjunktív alakja írható fel az egyszerűsítendő függvénynek.

Könnyen belátható, hogy az s függvény diszjunktív alakját elegendő egyszerű beszorzással felírni, mert az így kiadódó diszjunktív alak legkevesebb tényezőt tartalmazó szorzatából biztosan nem lehet egyetlen tényezőt sem elhagyni. Az s függvény felírási módjából következően ugyanis, semmiképpen sem szerepelhetnek benne tagadott változók, ami kizárja szomszédos szorzatok létezését. Elegendő tehát, ha a beszorzás után megkeressük a legkevesebb tényezőt tartalmazó szorzatot az s függvényben és eszerint írjuk fel az egyszerűsítendő függvény legegyszerűbb diszjunktív alakját. Ha az előzőekben felírt s függvényre elvégezzük a beszorzást, akkor

$$s = abecd + aecd + abec + aec$$

adódik. A beszorzás során természetesen alkalmazni kell az ismétlődő szorzatok elkerülése céljából az

$$A + A \equiv A$$

azonosságot. Látható, hogy az s-re így kiadódó diszjunktív alak nem a legegyszerűbb, de bárhogy alakítjuk is át az

$$AB + A \equiv A$$

azonosság alkalmazásával, a legkevesebb változót tartalmazó szorzat változatlanul aec marad, és a továbbiak szempontjából csak ez lényeges.

Példánkban tehát az egyszerűsítendő függvény legegyszerűbb diszjunktív alakja az a, c és e prímimplikánsok logikai összege lesz, amelyet szimbolikusan az alábbi módon jelölhetünk

$$F = a + c + e = 0{,}1(1) + 12,\ 14(2) + 3,\ 7,\ 11,\ 15(4,\ 8) =$$
$$= 0000 + 1100 + 0011 = \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}\bar{B}CD.$$

A prímimplikánsok jelölésére az összevonási eljárás során használt rövid jelölést alkalmaztuk. Ennek megfelelően például az a prímimplikánst 0, 1 (1)-gyel jelöltük stb. A prímimplikánsokat végül bináris kombinációkkal is felírtuk, és áthúztuk azokat a változóhelyértékeket, melyek nem szerepelnek majd a függvényben. Így a legegyszerűbb diszjunktív alak:

$$F(A, B, C, D) = \bar{A}\bar{B}\bar{C} + AB\bar{D} + CD.$$

A nem teljesen határozott logikai függvények számjegyes minimalizálását az alábbi példával mutatjuk be:

$$F = \sum^{5} [(0, 1, 2, 3, 4, 8, 10, 15, 16, 18, 28, 29) + (6, 9, 11, 13, 17, 19, 20, 22, 31)].$$

Az egyszerűsítés során, vagyis az összes prímimplikáns képzésekor a közömbös függvényértéknek megfelelő mintermeket úgy kell tekintenünk, mintha 1 függvényérték tartozna hozzájuk. Így biztosítható ugyanis, hogy a prímimplikánsok létrehozásában azok az összes lehetséges módon hassanak. Miután képeztük az összes prímimplikánst, az optimális lefedés meghatározásakor a közömbös függvényértéknek megfelelő mintermeket nem kell felvennünk a prímimplikánstáblába, hiszen azok lefedéséről nem kell gondoskodnunk, mert valójában nincs hozzájuk rendelve 1 függvényérték.

Példánkban az összevonás lépései az alábbiak:

I.	II.	III.	IV.
0√	0, 1 (1)√	0, 1, 2, 3 (1, 2)√	0, 1, 2, 3, 8, 9, 10, 11 (1, 2, 8)*e*
1√	0, 2 (2)√	0, 1, 8, 9 (1, 8)√	0, 1, 2, 3, 16, 17, 18, 19 (1, 2, 16)*f*
2√	0, 4 (4)√	0, 1, 16, 17 (1, 16)√	0, 2, 4, 6, 16, 18, 20, 22 (2, 4, 16)*h*
4√	0, 8 (8)√	0, 2, 4, 6 (2, 4)√	
8√	0, 16 (16)√	0, 2, 8, 10 (2, 8)√	
16√	1, 3 (2)√	0, 2, 16, 18 (2, 16)√	
3√	1, 9 (8)√	0, 4, 16, 20 (4, 16)√	
6√	1, 17 (16)√	1, 3, 9, 11 (2, 8)√	
9√	2, 3 (1)√	1, 3, 17, 19 (2, 16)√	
10√	2, 6 (4)√	2, 3, 10, 11 (1, 8)√	
17√	2, 10 (8)√	2, 3, 18, 19 (1, 16)√	
18√	2, 18 (16)√	2, 6, 18, 22 (4, 16)√	
20√	4, 6 (2)√	4, 6, 20, 22 (2, 16)√	
11√	4, 20 (16)√	8, 9, 10, 11 (1, 2)√	
13√	8, 9 (1)√	16, 17, 18, 19 (1, 2)√	
19√	8, 10 (2)√	16, 18, 20, 22 (2, 4)√	
22√	16, 17 (1)√	9, 11, 13, 15 (2, 4)*c*	
28√	16, 18 (2)√	13, 15, 29, 31 (2, 16)*d*	
15√	16, 20 (4)√		
29√	3, 11 (8)√		
31√	3, 19 (16)√		
	6, 22 (16)√		
	9, 11 (2)√		

I.	II.	III.	IV.
	9, 13 (4) √		
	10, 11 (1) √		
	17, 19 (2) √		
	18, 19 (1) √		
	18, 22 (4) √		
	20, 22 (2) √		
	20, 28 (8) *a*		
	11, 15 (4) √		
	13, 15 (2) √		
	13, 29 (16) √		
	28, 29 (1) *b*		
	15, 31 (16) √		
	29, 31 (2) √		

A prímimplikánsok legcélszerűbb kiválasztásához vegyük fel a prímimplikánstáblát (2.33. ábra). A táblában a közömbös függvényértékekhez tartozó mintermeket nem tüntetjük fel lefedendőkként. Az ábrán látható, hogy az m_1^5 minterm megkülönböztetett minterm, hiszen csak a *h* prímimplikáns fedi le. Hasonlóan megkülönböztetett minterm az m_8^5, melyet az *e* prímimplikáns, és az m_{10}^5, melyet szintén az *e* prímimplikáns fed le.

Prímimplikánsok \ Mintermek		0	1	2	3	4	8	10	15	16	18	28	29
20, 28 (8)	*a*											×	
28, 29 (1)	*b*											×	×
9, 11, 13, 15 (2, 4)	*c*								×				
13, 15, 29, 31 (2, 16)	*d*								×				×
0, 1, 2, 3, 8, 9, 10, 11 (1, 2, 8)	*e**	×	×	×	×		×	×					
0, 1, 2, 3, 16, 17, 18, 19 (1, 2, 16)	*f*	×	×	×	×					×	×		
0, 2, 4, 6, 16, 18, 20, 22 (2, 4, 16)	*h**	×		×		×				×	×		

2.33. ábra. Példa nem teljesen határozott függvény prímimplikánstáblájára

Az *e* és *h* prímimplikánsok így lényeges prímimplikánsok, ezért a logikai függvényben feltétlenül szerepelni fognak.

Írjuk fel a legegyszerűbb diszjunktív alak meghatározásához a prímimplikánstábla alapján a segédfüggvényt:

$$s = (e+f+h)(e+f)(e+f+h)(e+f)hee(c+d)(f+h)(f+h)(a+b)(b+d).$$

Átalakítva:

$$s = eh(a+b)(b+d)(c+d) = eh(ad+bc+bd) = adeh+bceh+bdeh.$$

A segédfüggvény három teljesen egyenértékű megoldást szolgáltat. Ezek közül a másodikat kiválasztva:

$$F = b+c+e+h = 28, 29\ (1)+9, 11, 13, 15\ (2, 4)+0, 1, 2, 3, 8, 9, 10, 11\ (1, 2, 8)+$$
$$+0, 2, 4, 6, 16, 18, 20, 22\ (2, 4, 16).$$

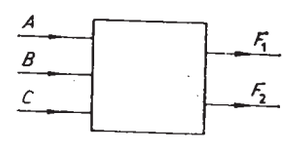
Az egyszerűsített logikai függvény algebrai alakja:

$$F(A, B, C, D, E) = ABC\bar{D}\bar{E} + \bar{A}B\bar{C}\bar{D}E+\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}+\bar{A}\bar{B}\bar{C}\bar{D}\bar{E} =$$
$$= ABC\bar{D}+\bar{A}BE+\bar{A}\bar{C}+\bar{B}\bar{E}.$$

2.3.3. Több kimenetű kombinációs hálózatok kimeneti függvényeinek grafikus minimalizálása

Az eddigiekben egykimenetű kombinációs hálózatokat leíró logikai függvények egyszerűsítésével foglalkoztunk. Ha a kombinációs hálózatnak több kimenete van, akkor — mint láttuk — kimenetenként egy-egy logikai függvénnyel írhatjuk le működését. Ha ezeket a függvényeket külön-külön egyszerűsítjük a megismert módszerekkel, akkor külön-külön eljuthatunk a legegyszerűbb diszjunktív vagy konjunktív alakokhoz. Ha ezek alapján felrajzoljuk a több kimenetű kombinációs hálózat elvi logikai rajzát, akkor nem biztos, hogy az ugyanúgy a legegyszerűbb kétszintű megvalósítást jelképezi, mint az egykimenetű esetben. Nem szabad ugyanis megfeledkeznünk arról, hogy az összes kimeneti függvény ugyanazon független változókon értelmezett (ezek a hálózat bemeneteit képviselik), miáltal a kimenetenkénti kétszintű megvalósításhoz képest egyszerűsítésekre adódhat lehetőség. Ennek szemléltetésére vizsgáljuk meg az alábbi egyszerű példát. Kétkimenetű, hárombemenetű hálózat legegyszerűbb kétszintű elvi logikai rajzát kell meghatároznunk (2.34. ábra). A két kimeneti függvény diszjunktív normálalakja:

$$F_1 = \bar{A}B\bar{C}+\bar{A}BC+ABC,$$
$$F_2 = \bar{A}BC+A\bar{B}C+ABC.$$



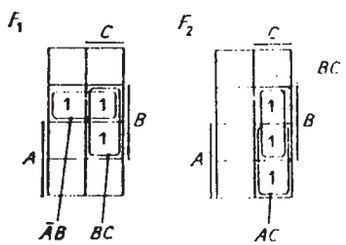
2.34. ábra. Példa több kimenetű kombinációs hálózatra

A 2.35. ábrán követhető a két függvény egyszerűsítése Karnaugh-táblán különkülön. A legegyszerűbb diszjunktív alakok:

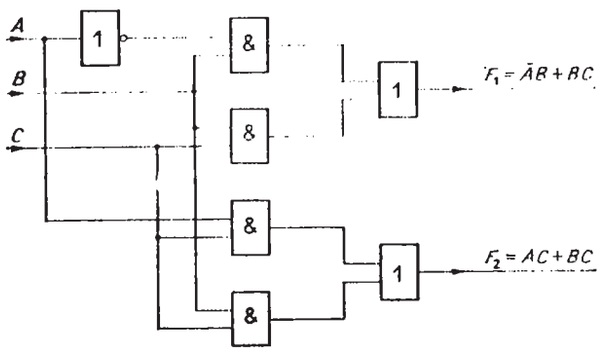
$$F_1 = \bar{A}B + BC; \quad F_2 = AC + BC.$$

A kétkimenetű hálózat kétszintű elvi logikai rajza ezek alapján a 2.36. ábrán látható. Mivel a BC prímimplikáns mindkét logikai függvényben előfordul, nyilván felesleges azt ugyanabban a hálózatban kétszer megvalósítani. Ezért a 2.36. ábra elvi logikai rajza nem felel meg a legegyszerűbb kétszintű megvalósításnak, hiszen a 2.37. ábrán bemutatott módon tovább egyszerűsíthető anélkül, hogy kétszintű jellege megváltozna. Az egyszerűsítést az tette lehetővé, hogy a két kimeneti függvénynek van ún. *közös prímimplikánsa:* a BC szorzat, amelyet a hálózatban elegendő csak egyszer megvalósítani.

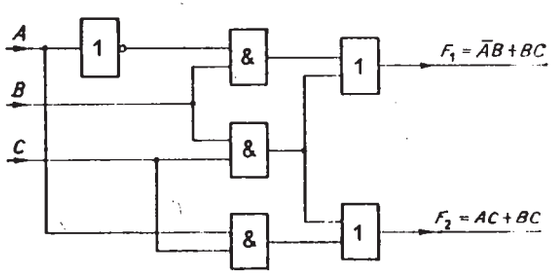
Fentiekből nyilvánvaló, hogy több kimenet esetén a megismert minimalizálási eljárásunkat módosítanunk kell. A cél a közös prímimplikánsok meghatározása,



2.35. ábra. Példa több kimenetű kombinációs hálózat kimeneti függvényeinek egyszerűsítésére



2.36. ábra. Kimenetenkénti minimalizálással kapott elvi logikai rajz

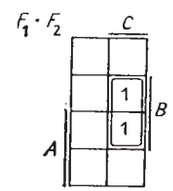


2.37. ábra. A közös prímimplikáns egyszeri megvalósításával kapott elvi logikai rajz

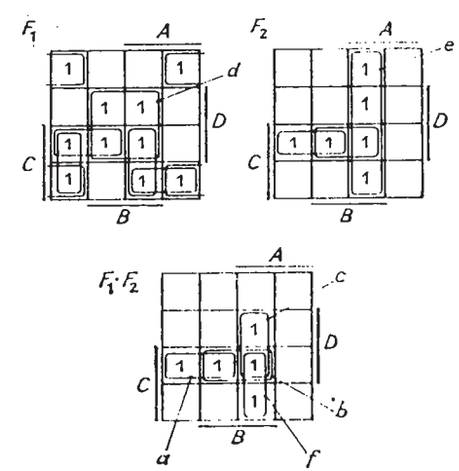
ami bonyolultabb esetekben természetesen nem olyan egyszerű, mint a bemutatott példában.

A közös prímimplikánsok szisztematikus megkeresésének az alapgondolata az, hogy két függvény közös prímimplikánsai biztosan prímimplikánsai a két függvény szorzatának. A szorzatfüggvény értéke ugyanis 1, ha mindkét függvény értéke 1; a közös prímimplikáns pedig éppen azért közös, mert az általa lefedett mintermeket mindkét függvény tartalmazza. A Karnaugh-tábla alapján a szorzatfüggvényt egyszerűen képezhetjük, hiszen formálisan olyan Karnaugh-táblás ábrázolást kell képeznünk, amely csak azokon a helyeken tartalmaz 1 bejegyzést, ahol mindkét táblán 1 szerepel. Ennek megfelelően a példánkban adott F_1 és F_2 függvények szorzata a 2.38. ábra Karnaugh-tábláján látható.

A szorzatfüggvényeknek természetesen lehetnek olyan prímimplikánsai, amelyek nem közös prímimplikánsok. Ezt szemlélteti a 2.39. ábra, amelyen bejelöltük az F_1 és F_2 függvény, valamint az $F_1 \cdot F_2$ szorzatfüggvény összes prímimplikánsát. Megfigyelhetjük, hogy a szorzatfüggvény Karnaugh-tábláján a-val jelölt prímimplikáns közös, mert F_1 és F_2 tábláján egyaránt szerepel prímimplikánsként. A b jelű prímimplikáns nem közös, mert az F_1 függvény tábláján nem prímimplikánsként, hanem a d jelű prímimplikáns képzéséhez felhasználható, tovább egyszerűsíthető szorzatként fordul elő. A c jelű prímimplikáns szintén nem közös, mert F_1 tábláján a d, F_2 tábláján pedig az e jelű prímimplikánst létrehozó, tovább egyszerűsíthető szorzatnak tekinthető. Az f jelű prímimplikáns hasonló okok miatt nem közös.



2.38. ábra. Példa szorzatfüggvény Karnaugh-táblájára



2.39. ábra. Példa a szorzatfüggvények prímimplikánsainak tulajdonságaira

Az optimális lefedés meghatározásakor nem szabad figyelmen kívül hagynunk a szorzatfüggvényeknek azokat a prímimplikánsait sem, amelyek nem közösek. Ezek ugyanis — mint láttuk — közös szorzatok, és így ezeket is csak egyszer kell megvalósítani a több kimenetű hálózatban. Igaz ugyan, hogy nem prímimplikánsok, de azáltal, hogy több kimeneti függvény előállításához használhatók, előfordulhat olyan eset, hogy figyelembevételük a legegyszerűbb diszjunktív alakban előnyösebb, mintha csak prímimplikánsokat használnánk.

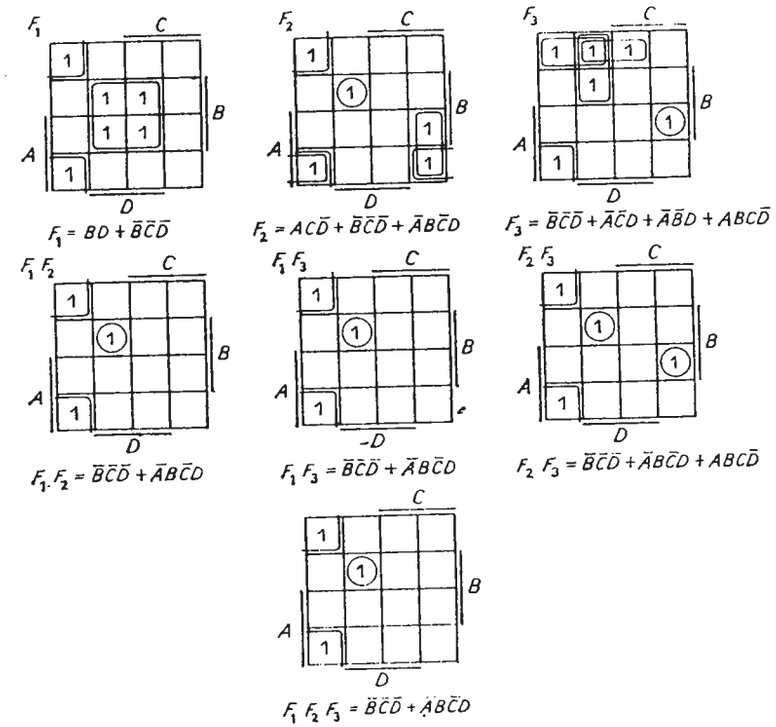
Több kimenetű esetben tehát az optimális lefedés meghatározásához az egyes kimeneti függvények prímimplikánsain kívül képeznünk kell az összes lehetséges szorzatfüggvény prímimplikánsait is. Az eljárás lényegének bemutatásához példaként határozzuk meg egy 4 bemenetű, 3 kimenetű kombinációs hálózat legegyszerűbb kétszintű ÉS—VAGY alakú elvi logikai rajzát. A három kimeneti függvény az alábbi:

$$F_1 = \sum^{4} (0, 5, 7, 8, 13, 15),$$

$$F_2 = \sum^{4} (0, 5, 8, 10, 14),$$

$$F_3 = \sum^{4} (0, 1, 3, 5, 8, 14).$$

A 2.40. ábrán láthatók az egyes kimeneti függvények és az összes lehetséges szorzatfüggvény Karnaugh-táblái, amelyeken bejelöltük az összes képezhető prímimplikánst. Az egyes Karnaugh-táblák alatt a prímimplikánsok alapján felírt legegyszerűbb diszjunktív algebrai alakok szerepelnek. Természetesen nem biztos, hogy a különkülön egyszerűsített kimeneti függvények legegyszerűbb diszjunktív alakjai alapján képezhetjük a legegyszerűbb kétszintű elvi logikai rajzot. Az optimális lefedéshez a szorzatfüggvények prímimplikánsait is figyelembe kell vennünk. Az optimális lefedés meghatározásához már ilyen egyszerű esetben is nehéz áttekinteni a Karnaugh-táblákat, ezért össze kell állítanunk a prímimplikánstáblát. A 2.41. ábrán láthatjuk a több kimenetű esetre módosított prímimplikánstáblát. A kiegészítés csupán annyi, hogy a lefedendő mintermek kimeneti függvényenként vannak csoportosítva, hiszen az összes kimeneti függvény minden egyes mintermjét le kell fednünk. Ezért a különböző kimeneti függvények azonos mintermjei annyiszor fordulnak elő, ahány kimeneti függvény azokat tartalmazza. A lefedéshez felhasználandó prímimplikánskészlet bővül az összes szorzatfüggvény összes prímimplikánsával. Azokat a szorzatfüggvény-prímimplikánsokat, amelyek egyben közös prímimplikánsok, természetesen, elegendő egyszer — a szorzatfüggvényhez tartozóan — feltüntetni a prímimplikánstáblában. A kimeneti függvények összes prímimplikánsait és az összes lehetséges szorzatfüggvény összes prímimplikánsait a továbbiakban a *több kimenetű prímimplikánsok* készletének nevezzük. A feladat tehát az, hogy a több kimenetű prímimplikánsok készletéből kiválasszuk azokat, amelyek figyelembevétele a legegyszerűbb kétszintű elvi logikai rajzot eredményezi. Ehhez fel kell írnunk a már megismert módon a lefedési segédfüggvény kifejezését a prímimplikánstábla alapján. A 2.41. ábrán a lényeges prímimplikánsok betűszimbólumait *-gal megjelöltük és bejegyzé-



2.40. ábra. Példa grafikus minimalizálásra több kimenetű esetben

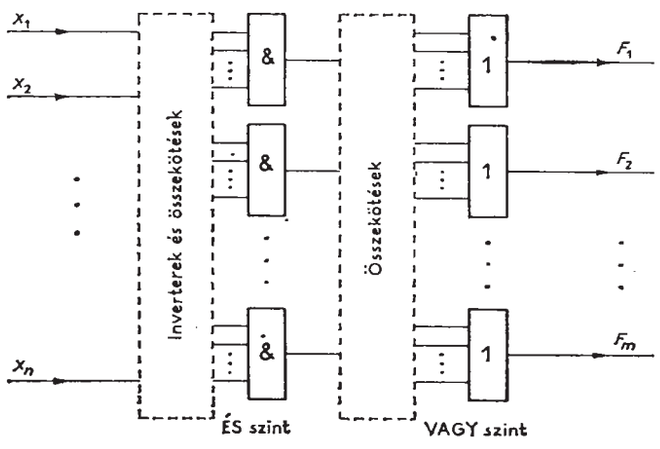
Kimeneti függvények, ill. szorzatfüggvények	Kimeneti függvények / Más kimenetű primimplikánsok \ Mintermek		F_1						F_2					F_3					
			$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}BC\bar{D}$	$\bar{A}BCD$	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}C\bar{D}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}BC\bar{D}$	$AB\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}BC\bar{D}$	$A\bar{B}C\bar{D}$	$ABC\bar{D}$
F_1	BD *	a		(×)	(×)		(×)	(×)											
F_2	$AC\bar{D}$	b										×	×						
	$A\bar{B}\bar{D}$	c									×	×							
F_3	$\bar{A}\bar{B}\bar{C}$	d												×	×				
	$\bar{A}\bar{C}D$	e													×		×		
	$\bar{A}\bar{B}D$ *	f													(×)	(×)			
$F_2 F_3$	$ABC\bar{D}$ *	g											(×)						(×)
$F_1 F_2 F_3$	$\bar{B}\bar{C}\bar{D}$ *	h	(×)			(×)			(×)		(×)			(×)				(×)	
	$\bar{A}B\bar{C}D$ *	i		(×)						(×)							(×)		

2.41. ábra. Példa a prímimplikánstáblára több kimenet esetén

seiket bekarikáztuk. Ezek betűjeleit a segédfüggvényben kiemelten leírhatjuk, hiszen ezek minden keletkező segédfüggvénybeli szorzatban szerepelni fognak:

$$s = afghi(a+i)(c+h)(b+c)(b+g)(d+h)(d+e+f)(e+i).$$

Egyetlen kimenet esetén a segédfüggvény alapján úgy határoztuk meg az optimális lefedést, hogy a szorzatösszeg-alak felírása után kiválasztottuk a legkevesebb tényezőt tartalmazó szorzatot. Több kimenet esetén nem állíthatjuk, hogy ilyen módon minden esetben a legegyszerűbb kétszintű elvi logikai rajzhoz jutunk. Könnyen belátható ugyanis, hogy a segédfüggvény ilyen szorzatának a kiválasztása csupán azoknak a több kimenetű prímimplikánsoknak a kiválasztását jelenti, amelyek megvalósítása — diszjunktív esetben — az elvi logikai rajzon az ÉS szinten a legkevesebb kapubemenetet igényli (2.42. ábra). Egyetlen kimenet esetén ez a megoldás azért eredményezte a legegyszerűbb kétszintű elvi logikai rajzot, mert a VAGY szinten csak egyetlen VAGY kapu volt, és ennek bemenetszámát is minimalizálta a legkevesebb tényezőt tartalmazó szorzat alapján történő megvalósítás. Több kimenet esetén a VAGY szinten több VAGY kapu van és az ÉS szinten megvalósított ÉS kapuk mindegyike több VAGY kapura is csatlakozhat. Ezáltal előfordulhat, hogy a segédfüggvénynek olyan szorzata adódik, amely annak ellenére, hogy nem a legkevesebb tényezős szorzat, mégis olyan prímimplikánsokat jelöl ki, amelyek megvalósítása a VAGY szinten szükséges kapubemenetek számát csökkenti. Összhatásában ez azt eredményezheti, hogy annak ellenére, hogy az ÉS szintet önmagában lehetne egyszerűsíteni, az elvi logikai rajz mégis így tartalmazza a legkevesebb kapubemenetet. Az elmondottakból következik, hogy több kimenet esetén a legegyszerűbb elvi logikai rajzhoz csak próbálgatással juthatunk el. A próbálgatás során elvileg a segédfüggvény minden egyes szorzata alapján elő kellene állítanunk az elvi logikai rajzot, és ezek közül kellene kiválasztanunk a legegyszerűbbet. A lefedési eljárás



2.42. ábra. Több kimenetű hálózat kétszintű ÉS—VAGY elvi logikai rajzának általános felépítése

számítógépes végrehajtásakor gyakran úgy hajtják végre a próbálgatási lépéseket, hogy a segédfüggvény szorzataihoz ún. jósági számokat rendelnek a bennük szereplő prímimplikánsok száma, a bennük szereplő közös prímimplikánsok száma, a befolyásolt kimeneti függvények száma stb. alapján. A szorzatoknak a jósági szám szerinti rangsorolásával néhány próbálgatással nagy valószínűséggel el lehet jutni a legegyszerűbb elvi logikai rajzhoz.

Egyszerű példánkban elegendő, ha a próbálgatás során a legkevesebb tényezőt tartalmazó szorzatokra szorítkozunk:

afghib vagy *afghic*.

Válasszuk először az utóbbi szorzatot. Ezzel kiválasztottuk az összes több kimenetű prímimplikáns közül azokat, amelyeket az elvi logikai rajzban ÉS kapukkal valósítunk meg. Ezeknek az ÉS kapuknak a kimeneteit úgy kell VAGY kapuk bemeneteire vezetnünk, hogy az előírt kimeneti függvények előálljanak és a VAGY szinten a kapubemenetek száma az adott választás esetén minimális legyen. Ez utóbbi feltételt úgy tudjuk általánosan megfogalmazni, hogy a már kiválasztott prímimplikánskészlettel — jelen esetben *afghic*-vel — kell optimális lefedést megvalósítani minden egyes kimeneti függvényre külön-külön. Ehhez felhasználhatjuk a kiindulási prímimplikánstáblának azokat a sorait, amelyek a kiválasztott prímimplikánskészlet elemeinek felelnek meg. Az egyes kimeneti függvényekhez tartozó mintermek lefedését kimenetenként felírt segédfüggvényekkel fogalmazhatjuk meg. Példánkban az F_1 függvény összes mintermjének a lefedési feltétele a kiválasztott prímimplikánskészlettel:

$$s_1 = h(a+i)a = ah+ahi.$$

Hasonló módon az F_2 és F_3 függvényekhez tartozó segédfüggvények:

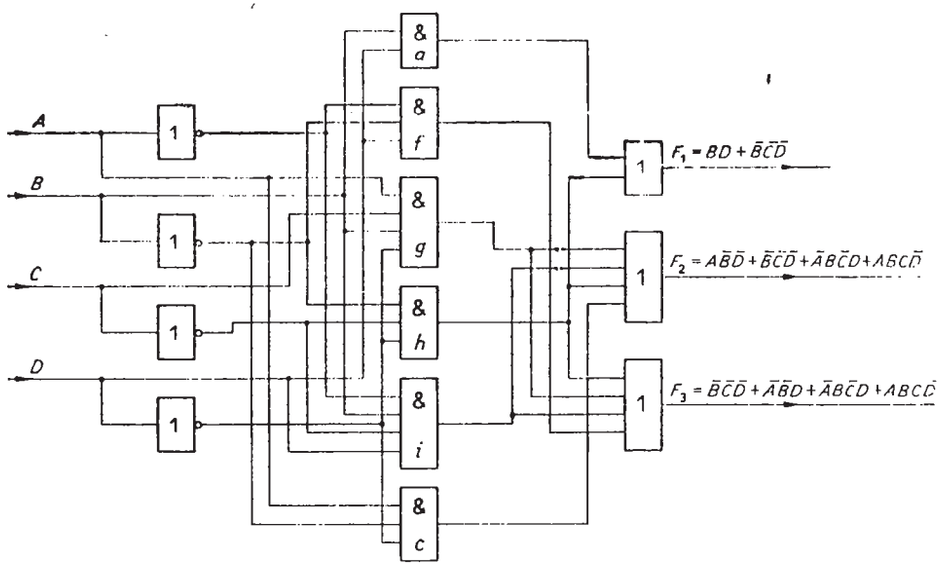
$$s_2 = hi(c+h)cg = chig,$$

$$s_3 = hfig.$$

A kimenetenkénti segédfüggvények szorzatösszeg-alakjának szorzataiból azt állapíthatjuk meg, hogy a kiválasztott több kimenetű prímimplikánsok közül melyek azok, amelyeket az egyes kimeneti függvények előállításához fel kell használnunk. Az egyes szorzatok tehát azt jelzik, hogy a tényezőinek megfelelő prímimplikánsokat megvalósító ÉS kapuk kimeneteit VAGY kapcsolatba hozva előáll az illető kimeneti függvénynek megfelelő kimeneti jel. A legegyszerűbb VAGY szintet nyilvánvalóan akkor kapjuk, ha a megvalósítást a legkevesebb tényezőt tartalmazó szorzat alapján végezzük.

A bemutatott egyszerű példában csak s_1 áll több szorzatból, ezek a szorzatok is egyenértékűek. Válasszuk a megvalósításhoz az *ah* szorzatot. A végeredményként kiadódó elvi logikai rajz a 2.43. ábrán látható.

Határozzuk meg ezek után az elvi logikai rajzot az *s* segédfüggvény *afghib* szorzatából kiindulva, amely a tényezők számát tekintve az *afghic*-vel egyenértékű. Képez-



2.43. ábra. A 2.40. ábrán megadott hálózat elvi logikai rajza az *afghic* szorzat alapján

zük az a, f, g, h, i, b prímimplikánsokat felhasználva a leírt módon a kimenetenkénti segédfüggvényeket:

$$s_1 = h(a+i) = ah+ai,$$

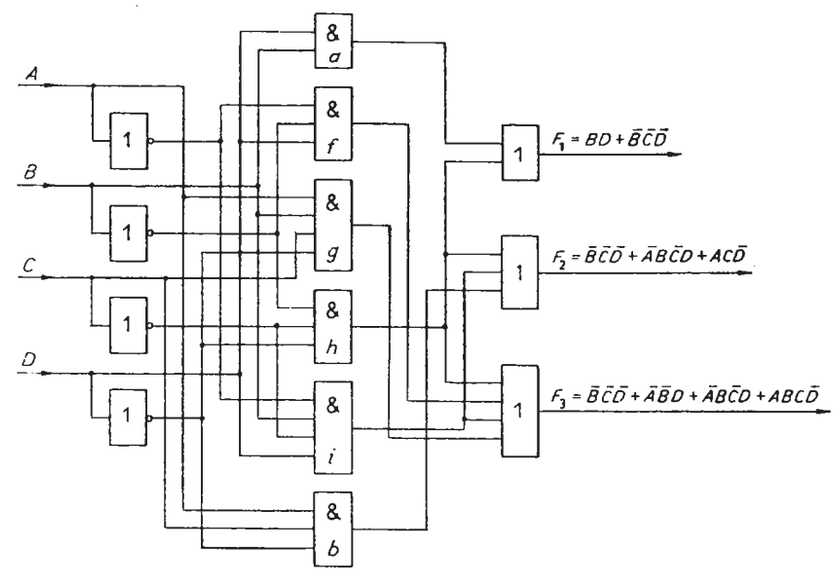
$$s_2 = hib(b+g) = hib+hibg,$$

$$s_3 = hfig.$$

Látható, hogy csak s_2 kifejezésében van eltérés az előző esethez képest. A *hib* szorzat csak három tényezőből áll, ami egyszerűsítést tesz lehetővé a VAGY szinten. Az így kialakítható elvi logikai rajz a 2.44. ábrán látható.

Megállapíthatjuk tehát, hogy az *s* függvény kiválasztott szorzatának a hatását csak úgy tudjuk megítélni, hogy képezzük a kimenetenkénti segédfüggvényeket. Példánkban láthatjuk, hogy a kétféle választás eredménye nem különbözött egyszerűség szempontjából az elvi logikai rajz ÉS szintjén, a VAGY szinten viszont az utóbbi választás egy kapubemenet megtakarítását tette lehetővé. Bonyolultabb feladatok esetében természetesen a különböző próbálgatások során keletkező elvi logikai rajzok nagyobb mértékben különbözhetnek.

Vizsgáljuk meg a 2.44. ábrán szereplő elvi logikai rajzot abból a szempontból, hogy az egyes kimeneti függvények megvalósításai megfelelnek-e a külön-külön képezhető legegyszerűbb diszjunktív alakjuknak. Ezek a diszjunktív alakok a 2.40. ábra Karnaugh-táblái alatt szerepelnek. A 2.44. ábra alapján felírható algebrai ala-



2.44. ábra. A 2.40. ábrán megadott hálózat elvi logikai rajza az *afghib* szorzat alapján

kok az egyes kimenetekre vonatkozóan:

$$F_1 = a+h = BD+\bar{B}\bar{C}\bar{D},$$

$$F_2 = h+i+b = \bar{B}\bar{C}\bar{D}+\bar{A}B\bar{C}D+AC\bar{D},$$

$$F_3 = h+f+i+g = \bar{B}\bar{C}\bar{D}+\bar{A}\bar{B}D+\bar{A}B\bar{C}D+ABC\bar{D}.$$

Megfigyelhetjük, hogy F_3 kifejezése nem egyezik meg a legegyszerűbb diszjunktív alakkal. Az $\bar{A}\bar{C}D$ prímimplikáns helyett az $\bar{A}B\bar{C}D$ minterm szerepel benne. Ennek oka, hogy a több kimenetű hálózatban nem volna értelme az $\bar{A}\bar{C}D$ prímimplikáns megvalósításának, mert az F_2 kimenet előállítása céljából az $\bar{A}B\bar{C}D$ mintermet külön is meg kell valósítani, az $\bar{A}\bar{C}D$-t létrehozó másik mintermet pedig előnyösen le lehet fedni az $\bar{A}\bar{B}D$ prímimplikánssal. Azáltal tehát, hogy F_3 előállítása önmagában bonyolultabb lett, a több kimenetű hálózat összességében egyszerűsödött.

2.3.4. Több kimenetű kombinációs hálózatok kimeneti függvényeinek számjegyes minimalizálása (Quine—McCluskey-módszer)

Több kimenet esetén az ismert számjegyes eljárást úgy kell kiegészíteni, hogy alkalmas legyen a több kimenetű prímimplikánsok előállítására. Ehhez minden mintermről, szorzatról, ill. prímimplikánsról tudni kell az eljárás során, hogy a kimeneti függvények közül melyekben fordul elő. Kiindulásképpen az összes kimeneti függ-

vény összes mintermjét a megismert módon csoportosítjuk bináris súlyok szerint. Minden mintermhez hozzárendelünk egy ún. jelzőkaraktert, amelynek rendeltetése az, hogy ábrázolja az egyes mintermek hozzárendelését a kimeneti függvényekhez. A jelzőkarakter célszerűen annyi bináris számjegyből áll, amennyi az egyszerűsítendő hálózat kimeneteinek száma. A jelzőkarakter azokon a helyértékeken 1 értékű, amelyeknek megfelelő kimeneti függvény az illető mintermet tartalmazza. Az eljárás bemutatása céljából egyszerűsítsük az alábbi két kimeneti függvénnyel adott kétkimenetű hálózatot:

$$F_1 = \sum^{4} (0, 1, 3, 5, 10, 11, 14, 15),$$

$$F_2 = \sum^{4} (0, 1, 6, 7, 8, 10, 11, 12, 14, 15).$$

Ha a jelzőkarakter helyértékeihez az F_1, F_2 sorrendben rendeljük hozzá a kimeneti függvényeket, akkor például a 3 jelű minterm jelzőkaraktere 10, mert csak F_1 tartalmazza, a 0 jelű mintermé pedig 11, mert F_1 is és F_2 is tartalmazza. Ennek megfelelően alakítható ki a számjegyes eljárás első oszlopa a jelzőkarakterekkel kiegészítve.

I.

	F_1	F_2	
0	1	1	√
1	1	1	√
8	0	1	√
3	1	0	√
5	1	0	√
6	0	1	√
10	1	1	√
12	0	1	√
7	0	1	√
11	1	1	√
14	1	1	√
15	1	1	√

II.

	F_1	F_2	
0, 1 (1)	1	1	*a*
0, 8 (8)	0	1	*b*
1, 3 (2)	1	0	*c*
1, 5 (4)	1	0	*d*
8, 10 (2)	0	1	√
8, 12 (4)	0	1	√
~~3, 7 (4)~~	~~0~~	~~0~~	
3, 11 (8)	1	0	*e*
~~5, 7 (2)~~	~~0~~	~~0~~	
6, 7 (1)	0	1	√
6, 14 (8)	0	1	√
10, 11 (1)	1	1	√
10, 14 (4)	1	1	√
12, 14 (2)	0	1	√
7, 15 (8)	0	1	√
11, 15 (4)	1	1	√
14, 15 (1)	1	1	√

III.

	F_1	F_2	
8, 10, 12, 14 (2, 4)	0	1	*f*
~~3, 7, 11, 15 (4, 8)~~	~~0~~	~~0~~	
6, 7, 14, 15 (1, 8)	0	1	*g*
10, 11, 14, 15 (1, 4)	1	1	*j*

Nyilvánvaló, hogy a leírt módszerrel az I. oszlopban 00 jelzőkarakter nem keletkezhet, hiszen ez azt jelentené, hogy az illető mintermet egyik kimeneti függvény sem tartalmazza. Látni fogjuk, hogy a II. oszloptól kezdve az eljárás során kiadódhat a csupa 0-t tartalmazó jelzőkarakter.

A II. oszlop képzésében egy pótlólagos szempontot kell figyelembe vennünk az egykimenetű esethez képest. A II. oszlopba, mint ismeretes, azok a szorzatok kerülnek, amelyek két szomszédos minterm összevonásából keletkeztek, így ezeknél is meg kell jelölnünk, hogy mely kimeneti függvények előállításában vesznek részt. Ha az I. oszlop két mintermje az egykimenetű esetben ismertetett összevonási feltételnek eleget tesz, akkor még nem biztos, hogy több kimenetű esetben a keletkezett szorzat fel is lesz tüntetve a II. oszlopban. Több kimenetű esetben csak akkor vonható össze két minterm az I. oszlopban, ill. két szorzat a többi oszlopban, ha jelzőkaraktereikben van legalább egy olyan helyérték, ahol mindkét jelzőkarakterben 1 szerepel. Könnyen belátható, hogy ez a feltétele annak, hogy legyen legalább egy olyan kimeneti függvény, amelyben mindkét összevonandó minterm, ill. szorzat szerepel. Nyilvánvaló, hogy az összevonás eredményeként létrejövő szorzat (végső esetben prímimplikáns) csak azokat a kimeneti függvényeket fogja befolyásolni, amelyekben mindkét összevonandó minterm, ill. szorzat szerepelt, vagyis amelyeknek megfelelő helyértékeken mindkét összevonandó minterm, ill. szorzat jelzőkaraktere 1-et tartalmaz. Ezek alapján könnyen megfogalmazhatjuk, hogyan kell formailag kezelnünk a jelzőkaraktereket az összevonás során:

— Az összevonás során keletkező karakter csak azokon a helyértékeken tartalmaz 1-et, amelyeken mindkét összevonandó minterm, ill. szorzat karaktere 1-et tartalmaz. A többi helyértékre természetesen 0-t kell írnunk, hiszen az ezeknek megfelelő kimeneti függvényekben nem szerepel mindkét összevonandó minterm, ill. szorzat.
— Nem végezhetjük el azokat az összevonásokat, amelyek során a fenti módon képzett karakter összes helyértékére 0 adódik. Ez ugyanis azt jelenti, hogy az összevonás során keletkező szorzatot egyik kimeneti függvény sem tartalmazza, mert az egyébként szomszédos mintermek, ill. szorzatok különböző kimeneti függvényekben szerepelnek.

Az előbbiekben ismertetett módon tehát minden oszlopban az egyes szorzatok mellett fel kell tüntetnünk a keletkezett karaktereket.

Így példánk 0 jelű mintermének (karaktere: 11) és 1 jelű mintermének (karaktere: 11) összevonása eredményeként a 0, 1 (1) jelű szorzatot kapjuk (ez a lépés az egykimenetű esetben ismertetett eljárással teljesen megegyezik), és ennek a karaktere a két karakter összevonásaként 11-re adódik. Ez azt jelenti, hogy a 0, 1 (1) egyszerűsített szorzat mindkét kimeneti függvényben előállítható, ill. a grafikus minimalizálással szemléltetve a kimeneti függvényeknek megfelelő két Karnaugh-táblán egyaránt létrehozható ugyanaz a 0, 1 (1) jelű kettes hurok. Ilyenformán az eredeti két kanonikus term mellé „√" jelet tehetünk, hiszen a 0, 1 (1) 11 jelölés pontosan arra utal, hogy ez az egyszerűsített szorzat mindkét függvényben szerepel. Ebből az látszik, hogy az összevonás eredményeként kapott szorzat kialakításában részt vevő, előző oszlopbeli elem mellé csak akkor tehetünk „√" jelet, ha az összevonás eredményeként kapott karakterrel egyező karaktere volt, ha ez nem áll fenn, akkor azt az elemet

nem szabad megjelölni. Ilyenkor ugyanis az összevonás során keletkező szorzat nem helyettesíti a vele nem azonos karakterrel rendelkező és az összevonásban részt vevő mintermeket, ill. szorzatokat, hiszen ekkor ezek nem ugyanazokban a kimeneti függvényekben szerepelnek. Az összevonást mégis célszerű elvégezni, mert a keletkező szorzat biztosan eggyel kevesebb változót tartalmaz, mint az összevont mintermek, ill. szorzatok. A csupa 0-tól különböző bármilyen karakter pedig azt jelenti, hogy a keletkezett szorzat legalább egy kimeneti függvényben szerepel, így annak egyszerűsítésében felhasználható lehet.

Például a 0 jelű minterm (karaktere: 11) és a 8 jelű minterm (karaktere: 01) összevonásaként a 0, 8 (8) jelű szorzatot (karaktere: 01) kapjuk. Ennek a két mintermnek az összevonása alapján csak a 8 jelű minterm mellé lehet „√" jelet tenni. Ez azt jelenti, hogy a 0, 8 (8) jelű szorzat (kettes hurok) csak az F_2-ben alakítható ki, így ebben az összevonásban az F_1 függvény 0 jelű mintermje nem szerepel, ezért az I. oszlop 0 jelű mintermjét ez alapján nem is szabad megjelölni. (A 0 jelű minterm mellé más összevonás eredményeként került a „√" jel.)

A II. oszlop kialakítása az előbbiek alapján elvégezhető, így azt nem részletezzük, a továbbiakban csupán a bevezető gondolatmenetünkben már érintett jelenségre térünk ki. Az I. oszlop harmadik csoportjában levő 3 jelű minterm és a negyedik csoportjában levő 7 jelű minterm összevonásaként a 3, 7 (4) jelű szorzatot kapjuk. A 3-minterm karaktere: 10, a 7-mitermé 01. Ezek összevonásából a 00 karakter adódik, ami annyit jelent, hogy ilyen szorzat nem alakítható ki, hiszen a 3 jelű minterm csak az F_1-ben, a 7 jelű minterm csak az F_2-ben szerepel. Ezért a 3, 7 (4) 00-t a II. oszlopból el kell hagyni. Ugyanez vonatkozik az 5, 7 (2) 00-ra és a III. oszlop kialakításakor kapott 3, 7, 11, 15 (4, 8)-ra is, amely a II. oszlopbeli 3, 11 (8) 10 és 7, 15 (8) 01 összevonásából adódott.

Azok a szorzatok a prímimplikánsok, amelyek mellé az összes lehetséges összevonás elvégzése után sem került „√" jel. Az egyes prímimplikánsok karaktere egyértelműen megadja, hogy melyik kimeneti függvény prímimplikánsáról, vagy mely kimeneti függvények közös prímimplikánsáról, vagy melyik szorzatfüggvény prímimplikánsáról van szó.

A prímimplikánstábla kitöltése ez alapján pontosan úgy történik, mint a grafikus minimalizálás esetén (2.45. ábra).

A prímimplikánstáblából látszik, hogy az F_1 kimeneti függvény 0, 5, 10, 14 és 15 jelű mintermjei és az F_2 kimeneti függvény 1, 6, 7, 11 és 12 jelű mintermjei megkülönböztetett mintermek, ezért az őket realizáló a, d, f, g és j prímimplikánsok lényegesek.

A prímimplikánstábla egyszerűsége folytán a segédfüggvény könnyen felírható:

$$s = adfgj(c+e) = acdfgj+adefgj.$$

Válasszuk ki a két szorzat közül az elsőt, ezzel a prímimplikánsokat megvalósító ÉS kapukat meghatároztuk. Az egyes kimeneteket előállító VAGY kapuk egyszerűsége érdekében írjuk fel a kiválasztott szorzat alapján adódott prímimplikánskészlet

Kimeneti függvények és szorzatfüggvény	Kimeneti függvények / Mintermek / Primimplikánsok	F_1								F_2									
		0	1	3	5	10	11	14	15	0	1	6	7	8	10	11	12	14	15
F_1	1,3 (2) c		×	×															
	1,5 (4) d		⊗		⊗														
	3,11 (8) e			×			×												
F_2	0,8 (8) b									×				×					
	8,10,12,14 (2,4) f													⊗	⊗		⊗	⊗	
	6,7,14,15 (1,8) g											⊗	⊗					⊗	⊗
F_1F_2	0,1 (1) a	⊗	⊗							⊗	⊗								
	10,11,14,15 (1,4) j					⊗	⊗	⊗	⊗						⊗	⊗		⊗	⊗

2.45. ábra. A számjegyes minimalizálást több kimenetű esetben bemutató példa prímimplikánstáblája

felhasználásával az egyes kimeneti függvényekre külön-külön a segédfüggvényeket:

$$s_2 = acdj \quad \text{és} \quad s_2 = afgj.$$

Ezek alapján:

$$F_1 = a+c+d+j,$$
$$F_2 = a+f+g+j,$$

és ebből az algebrai kifejezések:

$$F_1(A, B, C, D) = \bar{A}\bar{B}\bar{C}\bar{D}+\bar{A}\bar{B}\bar{C}D+\bar{A}\bar{B}\bar{C}D+A\bar{B}C\bar{D} = \bar{A}\bar{B}\bar{C}+\bar{A}\bar{B}D+\bar{A}\bar{C}D+AC,$$
$$F_2(A, B, C, D) = \bar{A}\bar{B}\bar{C}\bar{D}+A\bar{B}\bar{C}\bar{D}+\bar{A}BC\bar{D}+A\bar{B}C\bar{D} =$$
$$= \bar{A}\bar{B}\bar{C}+A\bar{D}+BC+AC.$$

Az elvi logikai rajz ÉS, VAGY kapukkal és inverterrel történő megvalósítás esetén a 2.46. ábrán látható.

Eddigi több kimenetű példáinkban az összes kimeneti függvény teljesen határozott volt. Ha a függvények nem teljesen határozottak, akkor ugyanúgy kell eljárni a közömbös függvényértékek rögzítése során, mint egyetlen kimenet esetén. A grafikus minimalizálási eljárásban a függvények és a szorzatfüggvények Karnaugh-tábláin a közömbös bejegyzések rögzítése a megismert szemléletes módon történik a legkedvezőbb prímimplikáns-elrendeződés kialakítására törekedve. A számjegyes eljárás során — az egykimenetű esethez hasonlóan — a közömbös függvényértékhez tartozó mintermeket az I. oszlop képzésekor ugyanúgy vesszük figyelembe, mintha 1 függvényértékhez tartoznának, a prímimplikánstábla lefedendő mintermjei között viszont nem tüntetjük fel őket.

Példaként tervezzük meg az alábbi nem teljesen határozott kimeneti függvényekkel adott 4 bemenetű, 3 kimenetű kombinációs hálózat legegyszerűbb kétszintű ÉS—VAGY elvi logikai rajzát:

$$F_1 = \sum^{4} [(0, 5, 7, 8, 10) + (2, 4, 13, 15)],$$

$$F_2 = \sum^{4} [(3, 7, 8, 10, 11) + (0, 15)],$$

$$F_3 = \sum^{4} [(0, 3, 6, 14, 15) + (7, 8)].$$

A prímimplikánsok képzését az alábbiakban követhetjük:

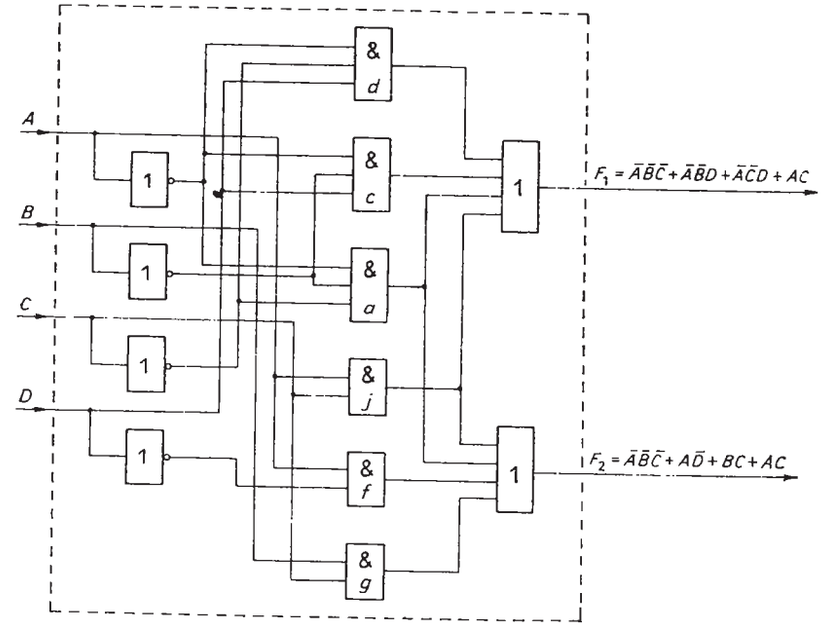
I.		II.		III.	
	$F_1 F_2 F_3$		$F_1 F_2 F_3$		$F_1 F_2 F_3$
0	1 1 1 √	0, 2 (2)	1 0 0 √	0, 2, 8, 10 (2, 8)	1 0 0 *h*
2	1 0 0 √	0, 4 (4)	1 0 0 *a*	3, 7, 11, 15 (4, 8)	0 1 0 *j*
4	1 0 0 √	0, 8 (8)	1 1 1 *b*	5, 7, 13, 15 (2, 8)	1 0 0 *k*
8	1 1 1 √	2, 10 (8)	1 0 0 √	6, 7, 14, 15 (1, 8)	0 0 1 *l*
3	0 1 1 √	4, 5 (1)	1 0 0 *c*		
5	1 0 0 √	8, 10 (2)	1 1 0 *d*		
6	0 0 1 √	3, 7 (4)	0 1 1 *e*		
10	1 1 0 √	3, 11 (8)	0 1 0 √		
7	1 1 1 √	5, 7 (2)	1 0 0 √		
11	0 1 0 √	5, 13 (8)	1 0 0 √		
13	1 0 0 √	6, 7 (1)	0 0 1 √		
14	0 0 1 √	6, 14 (8)	0 0 1 √		
15	1 1 1 √	10, 11 (1)	0 1 0 *f*		
		7, 15 (8)	1 1 1 *g*		
		11, 15 (4)	0 1 0 √		
		13, 15 (2)	1 0 0 √		
		14, 15 (1)	0 0 1 √		

A prímimplikánstábla a 2.47. ábrán látható. Az ábrán *a*, *b*, *e* és *l* jelű lényeges prímimplikánsok bejegyzéseit bekarikáztuk. A prímimplikánstábla alapján felírhatjuk az összes kimeneti függvény összes mintermjének lefedésére vonatkozó segédfüggvényt:

$$s = bel(c+k)(k+g)(h+d)(f+d)(f+j) =$$

$$= bel(fhk+dfk+djk+cfgh+cdfg+cdgj) =$$

$$= befhkl+bdefkl+bdejkl+bcefghl+bcdefgl+bcdegjl.$$



2.46. ábra. A számjegyes minimalizálást több kimenetű esetben bemutató példa elvi logikai rajza

Kimeneti függvények és szorzat-függvények	Kimeneti függvények / Mintermek / Primimplikánsok	F_1					F_2					F_3				
		0	5	7	8	10	3	7	8	10	11	0	3	6	14	15
F_1	0,4 (4) *a*	×														
	4,5 (1) *c*		×													
	0,2,8,10 (2,8) *h*	×			×	×										
	5,7,13,15 (2,8) *k*		×	×												
F_2	10,11 (1) *f*									×	×					
	3,7,11,15 (4,8) *j*						×	×			×					
F_3	6,7,14,15 (1,8) *l*													⊗	⊗	⊗
$F_1 F_2$	8,10 (2) *d*				×	×			×	×						
$F_2 F_3$	3,7 (4) *e*						⊗	⊗					⊗			
$F_1 F_2 F_3$	0,8 (8) *b*	⊗			⊗				⊗			⊗				
	7,15 (8) *g*			×				×								×

2.47. ábra. Számjegyes minimalizálási példa prímimplikánstáblája nem teljesen határozott több kimenetű esetben

Az eljárás folytatásához válasszuk ki a

$bdejk!$

szorzatot. Ennek alapján a kimenetenkénti segédfüggvények:

$s_1 = bdk,$

$s_2 = dj,$

$s_3 = bel.$

Így a kimeneti függvények algebrai alakjai:

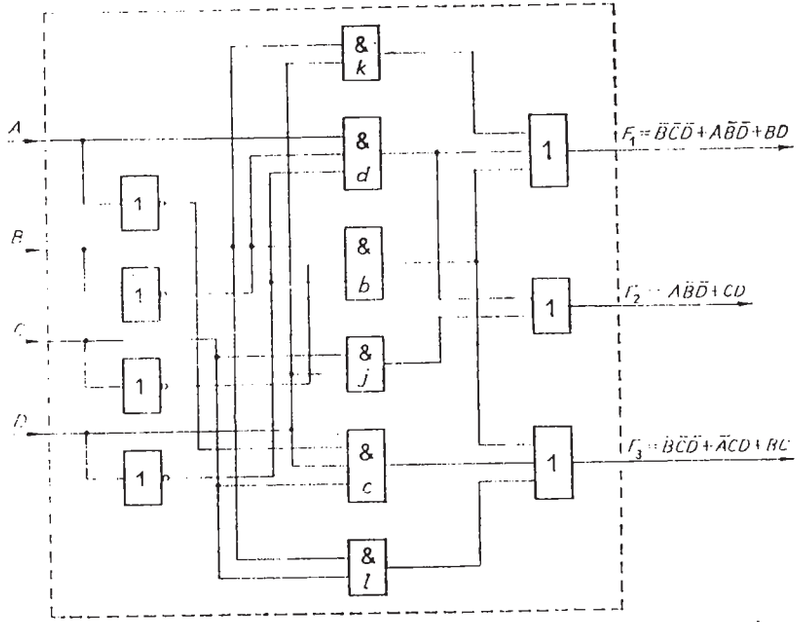
$F_1 = b+d+k = \bar{B}\bar{C}\bar{D}+A\bar{B}\bar{D}+BD,$

$F_2 = d+j = A\bar{B}\bar{D}+CD,$

$F_3 = b+e+l = \bar{B}\bar{C}\bar{D}+\bar{A}CD+BC.$

Az s függvény kiválasztott szorzata alapján tehát a 2.48. ábrán látható kétszintű ÉS—VAGY alakú elvi logikai rajzot kapjuk.

Az eddigiekben a számjegyes minimalizálást mindig a diszjunktív normálalakból kiindulva mutattuk be. Könnyen belátható azonban, hogy az eljárás lépései formailag változatlanok maradnak egy és több kimenetű esetben egyaránt, ha a konjunktív normálalakból indulunk ki. Ilyenkor természetesen a decimális számok a maxtermek indexeit jelentik, a prímimplikánstábla alapján a maxtermeket kell lefednünk, továbbá a prímimplikánsok nem szorzatok, hanem összegek, így a végeredmény alapján közvetlenül felrajzolható elvi logikai rajz VAGY—ÉS alakú. Az eljárás formai azonossága tulajdonképpen azon alapul, hogy — mint azt már korábban bemutattuk — két szomszédos maxterm helyettesíthető egyetlen összeggel, amelyből a két szomszédos maxtermet megkülönböztető változó hiányzik.



2.48. ábra. Számjegyes minimalizálási példa elvi logikai rajza nem teljesen határozott több kimenetű esetben

2.3.5. Szimmetrikus logikai függvények

A megismert minimalizálási eljárások természetesen csak akkor lehetnek eredményesek, ha az egyszerűsítendő függvény tartalmaz szomszédos mintermeket. Ellenkező esetben ugyanis a mintermek egyúttal prímimplikánsok és a diszjunktív normálalak egyben a legegyszerűbb diszjunktív alak. Vizsgáljuk meg ebből a szempontból a 2.49. ábrán igazságtáblával és Karnaugh-tábiával megadott teljesen határozott logikai függvényt. A diszjunktív normálalak:

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

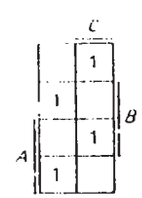
A megismert módszerekkel egyszerűsítésre nincs lehetőség. Ezzel szemben a függvény azzal a speciális tulajdonsággal rendelkezik, hogy érzéketlen a változók páronkénti felcserélésére. Például, ha az A és B változót felcseréljük, akkor az így kiadódó algebrai alak:

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

amelyről megállapíthatjuk, hogy azonos a kiindulási alakkal. Hasonló módon győződhetünk meg a többi páronkénti változócsere hatástalanságáról. Az igazságtábla alapján is levonható ugyanez a következtetés, ha a változóknak megfelelő oszlopok páronkénti felcserélésének a hatását vizsgáljuk.

Azokat a logikai függvényeket, amelyek a független változók tetszőleges páronkénti felcserélése esetén változatlanok maradnak, *szimmetrikus logikai függvényeknek* nevezik.

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



2.49. ábra. Példa szimmetrikus függvényre

A példaként vizsgált függvény igazságtáblájából azt is megállapíthatjuk, hogy a függvény értéke akkor és csak akkor 1, ha a három változó közül vagy csak egynek (de bármelyiknek), vagy mindháromnak az értéke 1. Ez a tulajdonság a szimmetrikus logikai függvényekre bizonyítható alábbi általános törvényszerűségből következik.

Minden szimmetrikus függvényhez megadható legalább egy olyan pozitív egész szám, amelyet szimmetriaszámnak neveznek és azt jelenti, hogy az 1 függvényértéket hány változó 1 értéke állítja elő. A szimmetrikus függvény értéke tehát akkor és csak akkor 1, ha 1 értéket adunk annyi változójának, amennyi a szimmetriaszámok valamelyikének értéke. A szimmetriából az is következik, hogy a változók közül tetszőlegesen választhatók ki azok, amelyeknek 1 értéket adunk, hiszen a szimmetrikus függvény érzéketlen a változók felcserélésére.

Példánkban a szimmetriaszámok ennek megfelelően: 1 és 3. Nyilvánvaló, hogy a szimmetriaszámok között a legnagyobb érték a függvény változóinak a száma lehet. A legkisebb szimmetriaszám, a 0 pedig önmagában azt jelenti, hogy a függvény értéke akkor és csak akkor 1, ha egyetlen változójának az értéke sem 1.

Bizonyítható, hogy a szimmetriaszámok létezése szükséges és elégséges feltétele annak, hogy a függvény szimmetrikus legyen. A szimmetrikus függvényeket egyértelműen jellemezhetjük a szimmetriaszámok és a független változók megadásával. Ezért az alábbi szimbolikus jelölést vezethetjük be a szimmetrikus függvények tömör megadására

$$S^n_{a_1, \dots, a_K}(x_1, \dots, x_n),$$

ahol n a változók száma, $a_1, \dots a_K$ a szimmetriaszámok.

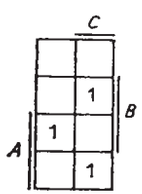
Példánkban a bevezetett jelöléssel:

$$F(A, B, C) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C} = S^3_{1,3}(A, B, C).$$

Azokat a szimmetrikus függvényeket, amelyeknek csak egyetlen szimmetriaszámuk van, *kanonikus szimmetrikus* függvényeknek nevezzük. Ilyen a 2.50. ábrán megadott függvény, amelynek diszjunktív normálalakja:

$$F(A, B, C) = \bar{A}BC + AB\bar{C} + A\bar{B}C = S^3_2(A, B, C).$$

Vizsgálataink célja, hogy az eddig megismert módszerekkel nem egyszerűsíthető szimmetrikus függvények gazdaságos megvalósítására találjunk valamilyen megoldást. Ehhez nyújt segítséget a szimmetrikus függvényekkel végezhető műveletek néhány alábbi tulajdonsága:



2.50. ábra. Példa kanonikus szimmetrikus függvényre

1. Két, ugyanazon független változókon értelmezett szimmetrikus logikai függvény logikai összege olyan szimmetrikus függvény, amelynek a szimmetriaszámai az eredeti két függvény szimmetriaszám-halmazainak egyesítéséből adódnak. Például:

$$S^4_{2,3}(A, B, C, D) + S^4_{0,2}(A, B, C, D) = S^4_{0,2,3}(A, B, C, D).$$

Az állítás helyessége könnyen belátható a logikai VAGY kapcsolat tulajdonságai alapján.

Az $S^4_{2,3}$ szimmetrikus logikai függvény algebrai alakját úgy célszerű felírni, hogy rendre képezzük az összes lehetséges olyan mintermeket, amelyben két változó ponált értékével fordul elő, majd az összes lehetséges olyan mintermet írjuk fel, amelyben három változó szerepel ponált értékével, vagyis

$$S^4_{2,3}(A, B, C, D) = \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + \\ + AB\bar{C}\bar{D} + \bar{A}BCD + A\bar{B}CD + AB\bar{C}D + ABC\bar{D}.$$

Az $S^4_{0,2}$ szimmetrikus logikai függvény algebrai alakjának felírása az előzőekhez hasonlóan történik, azzal a különbséggel, hogy a szimmetriaszámok ebben az esetben 0 és 2, vagyis

$$S^4_{0,2}(A, B, C, D) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + \\ + A\bar{B}C\bar{D} + AB\bar{C}\bar{D}.$$

A két szimmetrikus logikai függvény logikai összegzése az algebrai alakban előforduló összes minterm VAGY kapcsolatba hozását jelenti. Ezzel nyilvánvalóan belátható, hogy olyan szimmetrikus logikai függvény keletkezett, amelyben szerepel az összes lehetséges olyan minterm, amelyben nincs ponált változó, két ponált változó, ill. három ponált változó van. Ez azt jelenti, hogy a kapott szimmetrikus logikai függvény értéke minden olyan esetben és csak akkor 1, ha egyik változója sem 1 értékű, vagy kettő vagy három logikai változójának értéke 1. Ennek a tulajdonságnak az alapján tetszőleges szimmetrikus függvény felírható kanonikus szimmetrikus függvények összegeként. Például:

$$S^4_{0,3}(A, B, C, D) = S^4_0(A, B, C, D) + S^4_3(A, B, C, D).$$

2. Két, ugyanazon független változókon értelmezett szimmetrikus függvény logikai szorzata is szimmetrikus függvény, ha a két függvény szimmetriaszámai között van legalább egy azonos. A szorzatfüggvény szimmetriaszámai a két függvény szimmetriaszám-halmazainak közös elemei. Például:

$$S^4_{1,2,3}(A, B\ C, D) \cdot S^4_{1,3,4}(A, B, C, D) = S^4_{1,3}(A, B, C, D).$$

Az állítás helyességéről legegyszerűbben úgy győződhetünk meg, hogy megvizsgáljuk mi a feltétele a szorzatfüggvény 1 értékének. A szorzatfüggvény értéke akkor és csak akkor 1, ha mindkét tényezőfüggvény értéke 1. Mivel a tényezőfüggvények szimmetrikusak, az 1 értékükhöz szükséges változókombinációkat, ill. mintermeket a szimmetriaszámok alapján képezhetjük. Nyilvánvaló, hogy mindkét tényezőfügg-

vény értéke — és ezáltal a szorzatfüggvény értéke — olyan és csak olyan változókombinációk, ill. mintermek mellett lesz 1, amelyeket a mindkét tényezőfüggvényben azonos szimmetriaszámok alapján képeztünk.

Ha két tényezőfüggvénynek nincsenek azonos szimmetriaszámai, akkor természetesen nem létezik olyan változókombináció, amely mellett mindkét tényezőfüggvény egyaránt 1 értékű volna, miáltal a szorzatfüggvény értéke azonosan 0.

A szimmetrikus függvények szorzatára bemutatott tulajdonságot felhasználhatjuk kanonikus szimmetrikus függvények előállítására. Például:

$$S^4_{2,3}(A, B, C, D) \cdot S^4_{1,3,4}(A, B, C, D) = S^4_3(A, B, C, D).$$

3. Egy n változós szimmetrikus logikai függvény negáltja szintén szimmetrikus. A tagadott függvény szimmetriaszámai az összes olyan n-nél nem nagyobb természetes számok, amelyek a kiindulási függvény szimmetriaszámai között nem szerepelnek. A tagadás művelete tehát a szimmetriaszám-halmaz kiegészítő halmazának képzését jelenti. Például:

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

Az állítás helyessége könnyen belátható annak alapján, hogy egy teljesen határozott logikai függvény tagadottja minden olyan változókombináció mellett 1, amelyhez a kiindulási függvény 0 értéke tartozik.

4. Ha egy n változós szimmetrikus függvényt a változóinak a tagadottján értelmezünk, akkor olyan szimmetrikus függvényt kapunk, amelynek szimmetriaszámai rendre a kiindulási függvény szimmetriaszámaiból képezhetők n-ből történő kivonás útján. Például:

$$S^4_{2,3}(A, B, C, D) = S^4_{1,2}(\bar{A}, \bar{B}, \bar{C}, \bar{D}).$$

Az állítás általános érvényességéről könnyen meggyőződhetünk, ha a szimmetriaszámok definíciójából indulunk ki. Eszerint a szimmetriaszám megadja, hogy hány változónak kell 1 értékűnek lennie az 1 függvényértéket előállító változókombinációkban. A szimmetriaszám alapján természetesen azt is meghatározhatjuk, hogy hány változónak kell 0 értékűnek lennie az 1 függvényértéket előállító változókombinációkban. Ezt a számot nyilvánvalóan úgy kapjuk, hogy a szimmetriaszámot kivonjuk a változók számából, hiszen a szimmetriaszám definíciószerűen az 1 értékű változók száma, és a 0 értékű változók számával kiegészítve azt, minden változókombinációban a függvény változóinak száma adódik. Ha ezek után a tagadott változókat tekintjük változóknak, akkor a korábbi 0 értékű változók lesznek az 1 értékű változók és a változók számából kivont kiindulási szimmetriaszámok lesznek az új szimmetriaszámok.

Ez a szabály tulajdonképpen a szimmetrikus függvények egyfajta függetlenváltozó transzformációjának végrehajtására vonatkozik.

A bemutatott négy műveleti tulajdonság, valamint a szimmetriaszámok értelmezése alapján adott szimmetrikus függvényekből egyszerűen előállíthatunk más szimmetrikus függvényeket. Így, ha az építőelem-készletben rendelkezésünkre állnak bizonyos szimmetrikus függvények, akkor ezekből felépítve sok más szimmetrikus függvényt is egyszerűen megvalósíthatunk anélkül, hogy a megvalósítandó szimmetrikus függvény említett minimalizálási nehézségei gondot okoznának.

Ha a felhasználható építőelemek között például KIZÁRÓ VAGY kapu is rendelkezésünkre áll, akkor azt szimmetrikus függvényt megvalósító építőelemként is felhasználhatjuk. Az $S^2_1(x_1, x_2)$ függvény ugyanis nem más, mint KIZÁRÓ VAGY kapcsolat (azaz mod—2-es összeg, ill. antivalencia):

$$S^2_1(x_1, x_2) = x_1\bar{x}_2 + \bar{x}_1 x_2.$$

A 2.51. ábrán példaként bemutattuk, hogy milyen függvényt valósíthatunk meg az S^2_1 építőelemek vázolt sorba kapcsolásával. A szimmetriaszámok definíciója alapján könnyen beláthatjuk, hogy az egyes elemek kimeneti függvényét rendre az alábbi módon írhatjuk fel:

$$F_1 = S^2_1[S^2_1(x_1, x_2), x_3] = S^3_{1,3}(x_1, x_2, x_3),$$

$$F_2 = S^2_1[S^3_{1,3}(x_1, x_2, x_3), x_4] = S^4_{1,3}(x_1, x_2, x_3, x_4),$$

és így tovább,

$$F_6 = S^8_{1,3,5,7}(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8).$$

Általánosabban:

$$F_{k-1} = S^2_1[S^k_{1,3,5,\ldots,m}(x_1, \ldots, x_k), x_{k+1}] =$$

$$= \begin{cases} S^{k+1}_{1,3,5,\ldots,m,m+2}(x_1, \ldots, x_{k+1}), & \text{ha} \quad m < k \\ S^{k+1}_{1,3,5,\ldots,m}(x_1, \ldots, x_{k+1}), & \text{ha} \quad m = k \end{cases}$$

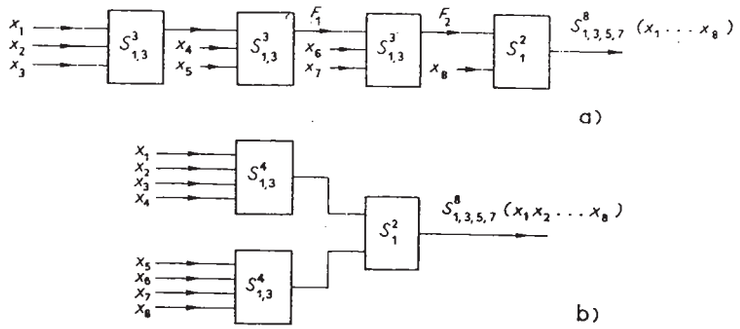
$$m = 1, 3, 5, 7, \ldots,$$

$$k = 2, 3, 4, 5, 6, 7 \ldots$$

Kevesebb szinttel valósíthatjuk meg az $S^8_{1,3,5,7}$ függvényt, ha rendelkezésünkre áll olyan építőelem is, amely az $S^3_{1,3}$, ill. $S^4_{1,3}$ függvényt valósítja meg. A 2.52*a)* ábrán bemutatott megoldás helyességéről is a szimmetriaszámok értelmezése alapján győződhetünk meg. Például az F_1-gyel jelölt kimeneti függvény:

$$F_1 = S^3_{1,3}[S^3_{1,3}(x_1, x_2, x_3), x_4, x_5] = S^5_{1,3,5}(x_1, x_2, x_3, x_4, x_5).$$

2.51. ábra. Példa az $S^8_{1,3,5,7,1}(x_1, \ldots, x_8)$ szimmetrikus függvény megvalósítására az $S^2_1(x_1, x_2)$ szimmetrikus függvényt megvalósító építőelemek felhasználásával



2.52. ábra. Az $S^8_{1,3,5,7}(x_1, \ldots, x_8)$ szimmetrikus függvény felépítése $S^3_{1,3}$, $S^3_{1,4}$ és S^2_1 szimmetrikus függvényeket megvalósító építőelemek felhasználásával

Általánosabban:

$$F_{k-2} = S^3_{1,3}[S^k_{1,3,\ldots,m}(x_1, x_2, \ldots, x_k),\ x_{k+1}, x_{k+2}] = S^{k+2}_{1,3,\ldots,m,m+2}(x_1, x_2, \ldots, x_{k+2}),$$

$$m = 1, 3, 5, 7\ldots$$

$$k = 3, 4, 5, 6, 7, 8\ldots$$

A 2.52*b)* ábrán a szintek számát tovább csökkenthettük az $S^4_{1,3}$ építőelem felhasználásával. A megoldás helyességét az alábbi gondolatmenettel szemléltethetjük:

$$S^2_1[S^4_{1,3}(x_1, x_2, x_3, x_4),\ S^4_{1,3}(x_5, x_6, x_7, x_8)] =$$

$$= S^4_{1,3}(x_1, x_2, x_3, x_4) \cdot \overline{S^4_{1,3}(x_5, x_6, x_7, x_8)} + \overline{S^4_{1,3}(x_1, x_2, x_3, x_4)} \cdot S^4_{1,3}(x_5, x_6, x_7, x_8) =$$

$$= S^4_{1,3}(x_1, x_2, x_3, x_4) \cdot S^4_{0,2,4}(x_5, x_6, x_7, x_8) + S^4_{0,2,4}(x_1, x_2, x_3, x_4) \cdot S^4_{1,3}(x_5, x_6, x_7, x_8).$$

Az így kapott kifejezés logikai értékéről a szimmetriaszámok jelentése alapján megállapítható, hogy az összes olyan és csak olyan $x_1x_2\ldots x_8$ értékkombinációk behelyettesítése esetén 1, amelyek 1-es, vagy 3-as, vagy 5-ös, vagy 7-es bináris súllyal rendelkeznek.

A fentiekhez hasonló megoldások vizsgálatakor gyakran hasznos, ha a kifejezésekben az ÉS, VAGY, NAND, NOR kapcsolatokat is formális anszimmetrikus függvényként szerepeltetjük. A szimmetriaszámok definíciója értelmében ugyanis könnyen beláthatjuk, hogy

$$S^2_2(x_1, x_2) = x_1x_2; \qquad S^2_{1,2}(x_1, x_2) = x_1 + x_2;$$

$$S^2_{0,1}(x_1, x_2) = \overline{x_1x_2}; \qquad S^2_0(x_1, x_2) = \overline{x_1 + x_2}.$$

További példaként tételezzük fel, hogy az építőelem-készletben rendelkezésünkre állnak az alábbi szimmetrikus függvények:

$$S^3_{1,3}(x_1, x_2, x_3); \qquad S^3_2(x_1, x_2, x_3).$$

Valósítsuk meg az adott készlet felhasználásával az $S^3_{2,3}(x_1, x_2, x_3)$ szimmetrikus függvényt.

A feladat próbálgatásos jellegű megoldásához előnyösen alkalmazhatjuk a bemutatott műveleti tulajdonságokat:

$$S^3_{1,3}(x_1, x_2, x_3) + S^3_2(x_1, x_2, x_3) = S^3_{1,2,3}(x_1, x_2, x_3)$$

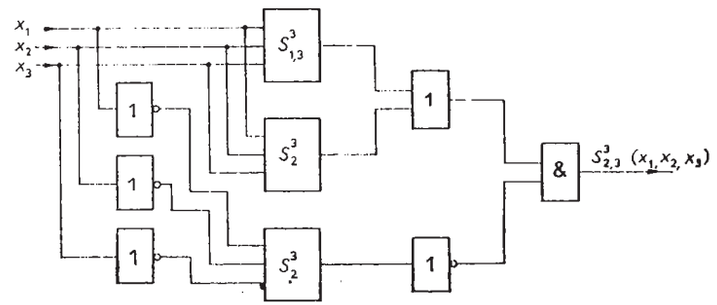
$$S^3_{1,2,3}(x_1, x_2, x_3) \cdot \overline{S^3_1(x_1, x_2, x_3)} = S^3_{1,2,3}(x_1, x_2, x_3) \cdot S^3_{0,2,3}(x_1, x_2, x_3) =$$

$$= S^3_{2,3}(x_1, x_2, x_3).$$

Ezek után már csak $S^3_1(x_1, x_2, x_3)$ előállítását kell megoldani az adott építőelem-készlettel, amihez például az

$$S^3_1(x_1, x_2, x_3) = S^3_2(\bar{x}_1, \bar{x}_2, \bar{x}_3)$$

függetlenváltozó-transzformációt használhatjuk fel. Az így kiadódó elvi logikai rajz a 2.53. ábra alapján követhető.



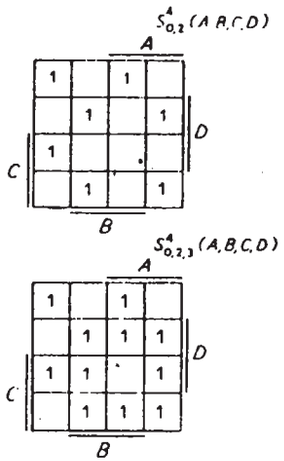
2.53. ábra. Példa az $S^3_{2,3}(x_1, x_2, x_3)$ szimmetrikus függvény megvalósítására

A bemutatott egyszerű példákból is látható, hogy a szimmetrikus függvények felépítése adott szimmetrikus építőelemekből alapvetően próbálgatásos intuitív jellegű eljárás és már egyszerű esetekben is több szintű elvi logikai rajzot eredményez. A későbbiekben látni fogjuk, hogy a több szintű hálózatok azon kívül, hogy lassúbb működésűek, a jelterjedési késleltetések által okozott átmeneti kimeneti kombinációk szempontjából is nehezebben kezelhetők, mint a kétszintűek. A gyakorlati esetek jelentős részében azonban mégis célszerű a szemléltetett felépítési módot választani a szimmetrikus függvények minimalizálási nehézségei miatt.

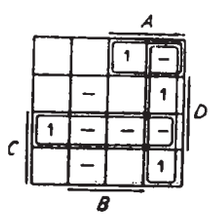
A 2.49. és 2.50. ábrákon olyan szimmetrikus függvényeket ábrázoltunk, amelyek kanonikus normálalakja egyben a legegyszerűbb diszjunktív alak. A szimmetriaszámoktól függően azonban előfordulhat, hogy egy szimmetrikus függvény kismértékben egyszerűsíthető. A szimmetriaszámok ugyanis azt is kifejezik, hogy a függvény tartalmaz-e szomszédos mintermeket. A kanonikus szimmetrikus függvények például biztosan nem tartalmaznak szomszédos mintermeket, mert csak egy szim-

metriaszámuk van, ami a függvény bármely két mintermjének megfelelő változókombinációkban egynél több helyértéken jelent eltérést. Általában is megállapíthatjuk, hogy nem egyszerűsíthetők azok a szimmetrikus függvények, amelyek szimmetriaszámai között nincs legalább két egymástól 1-gyel különböző. Ha ugyanis bármely két szimmetriaszám közötti különbség nagyobb 1-nél, akkor kizárt, hogy a függvény szomszédos mintermeket tartalmazzon. Ha egy függvénynek viszont van legalább két olyan szimmetriaszáma, amelyek közötti különbség 1, akkor az ezeknek megfelelő mintermek között biztosan akadnak szomszédosak. A 2.54. ábrán példaként Karnaugh-táblán ábrázoltuk az $S^4_{0,2}(A, B, C, D)$ és az $S^4_{0,2,3}(A, B, C, D)$ szimmetrikus függvényeket. Megállapíthatjuk, hogy az utóbbiban éppen a 3-as szimmetriaszám pótlólagos felvétele miatt alakíthatók ki két szomszédos mintermet helyettesítő szorzatok.

Ha egy megvalósítandó logikai függvény nem teljesen határozott, akkor mindig célszerű megvizsgálni, hogy a közömbös értékek megfelelő rögzítésével a függvény nem tehető-e szimmetrikussá. Ha ugyanis szimmetrikus függvényeket megvalósító építőelemek is rendelkezésünkre állnak, akkor sok esetben előnyösebb ezek felhasználása, mint például a legegyszerűbb diszjunktív alakból kiindulni. Például a 2.55. ábrán Karnaugh-táblával adott függvény a bejelölt módon viszonylag kedvezően minimalizálható. A legegyszerűbb diszjunktív alakból származtatható elvi



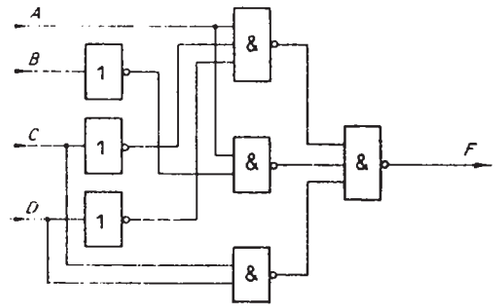
2.54. ábra. A szimmetriaszámok hatása a minimalizálhatóságra



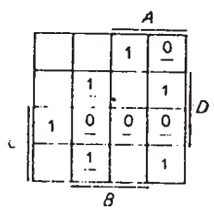
2.55. ábra. Példa szimmetrikussá tehető nem teljesen határozott logikai függvény minimalizálására szimmetriavizsgálat nélkül

logikai rajz a 2.56. ábrán látható. Ha az összevonás előtt megvizsgáljuk a közömbös értékek rögzítésének lehetőségét a szimmetria szempontjából, akkor eljuthatunk a 2.57. ábrán jelölt rögzített értékekhez. Ezáltal az $S^4_2(A, B, C, D)$ kanonikus szimmetrikus függvényt alakítottuk ki, amelynek megvalósítása szimmetrikus építőelemekből az előzőekben vázolt módon egyszerűbb elvi logikai rajzhoz vezethet a 2.56. ábrához képest.

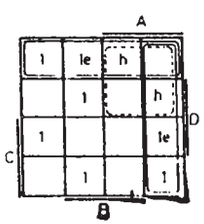
Sok esetben akkor is célszerű megvizsgálni, hogy a megvalósítandó logikai függvény szimmetrikussá tehető-e, ha ez csak a függvény megváltoztatásával érhető el. Azok a függvények ugyanis, amelyek kismértékben különböznek egy szimmetrikus függvénytől, nyilvánvalóan szintén kedvezőtlenül minimalizálhatók, és a szimmetrikus építőelemekből történő felépítést nem végezhetjük el az előzőekben vázolt módon közvetlenül. Például a 2.58. ábrán megadott függvény szimmetrikus lenne, ha az *e*-vel jelölt mintermeket elhagynánk belőle, és a *h*-val jelölteket hozzávennénk. Így az $S^4_{0,2}(A, B, C, D)$ szimmetrikus függvényhez jutnánk. A kiindulási függvény nem szimmetrikus, mégis viszonylag kedvezőtlenül minimalizálható. A legegyszerűbb diszjunktív alakból a 2.59. ábrán látható elvi logikai rajz adódik. Ha ezek után feltételezzük, hogy az építőelem-készletben rendelkezésünkre áll az $S^4_{0,2}(A, B, C, D)$-t megvalósító elem, akkor ebből úgy állíthatjuk elő a kiindulási



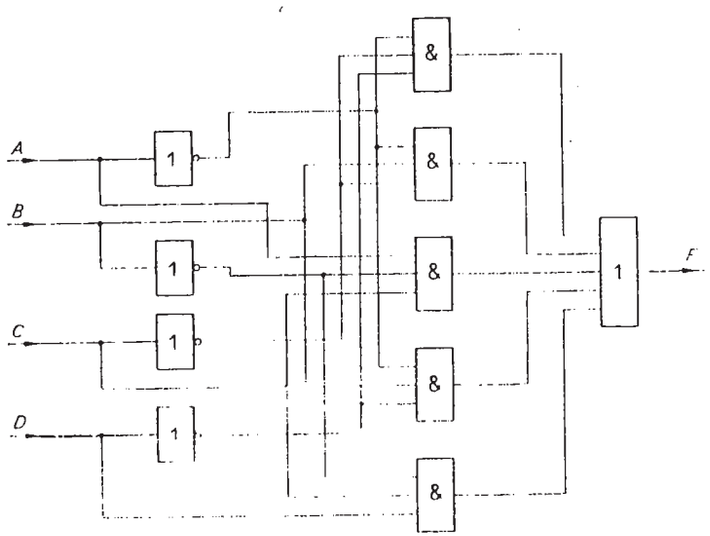
2.56. ábra. A 2.55. ábrán bejelölt összevonásból származtatható elvi logikai rajz



2.57. ábra. A 2.55. ábrán adott függvény közömbös értékeinek rögzítése a szimmetria kialakítása céljából



2.58. ábra. Példa mintermek elhagyásával és hozzátételével kialakított szimmetriára



2.59. ábra. A 2.58. ábrán szereplő kiindulási függvény legegyszerűbb díszjunktív alakjából adódó elvi logikai rajz

függvényt, hogy a hozzávett és az elhagyott mintermeknek megfelelően módosítjuk a függvény értékét. Az $S^4_{0,2}(A, B, C, D)$ függvény értékéből úgy állíthatjuk elő a kiindulási függvény értékét, hogy pótlólagosan biztosítjuk az 1 értéket az elhagyott mintermeknek megfelelő változókombinációk mellett, és megtiltjuk az 1 értéket a hozzávett mintermeknek megfelelő változókombinációk mellett. Az 1 függvényérték pótlólagos biztosítása VAGY kapcsolatot, tiltása pedig a tagadottal való ÉS kapcsolatot jelent:

$$F(A, B, C, D) = [S^4_{0,2}(A, B, C, D) + E(A, B, C, D)] \cdot \overline{H(A, B, C, D)},$$

ahol $F(A, B, C, D)$ a kiindulási függvény, $E(A, B, C, D)$ az elhagyott mintermek logikai összegéből alkotott függvény, $H(A, B, C, D)$ a hozzávett mintermek logikai összegéből alkotott függvény.

A pótlólagos 1 értékek biztosításának és a tiltásnak a sorrendjét felcserélhetjük:

$$F(A, B, C, D) = S^4_{0,2}(A, B, C, D) \cdot \overline{H(A, B, C, D)} + E(A, B, C, D).$$

Nem teljesen határozott esetben az E és H függvények tartalmazhatnak azonos mintermeket is a közömbös bejegyzések egymástól független rögzíthetősége következtében. Az $E(A, B, C, D)$ és $H(A, B, C, D)$ függvények képzésekor természetesen a legegyszerűbb alakra célszerű törekedni. E célból $H(A, B, C, D)$ előállításához felhasználhatjuk a h jelű mintermeken kívül a kiindulási függvény 0 értékét előállító bármely változókombinációjának megfelelő mintermeket, hiszen $H(A, B, C, D)$-nek

értelemszerűen 1 értéke lehet a kiindulási függvény minden 0 értéket előállító változókombinációja esetén. Ugyanilyen okból használhatjuk $E(A, B, C, D)$ előállításához az e jelű mintermeken kívül a kiindulási függvény 1 értékét előállító bármely változókombinációjának megfelelő mintermeket, hiszen $E(A, B, C, D)$-nek 1 értéke lehet a kiindulási függvény minden 1 értéket előállító változókombinációja esetén.

Általában további egyszerűsítések érhetők el ezáltal, hogy az $F = (S^4_{0,2} + E) \cdot \overline{H}$ szerinti megvalósításban az E függvény képzésekor felhasználunk olyan mintermeket is, amelyeknek megfelelő változó kombináció esetén $H = 1$ és $F = 0$. Így $E \cdot H = 0$ már nem teljesül, de a H függvénnyel való ÉS kapcsolat biztosítja, hogy az eredő függvény (F) mindig 0 értéket állítson elő, ha $H = 1$.

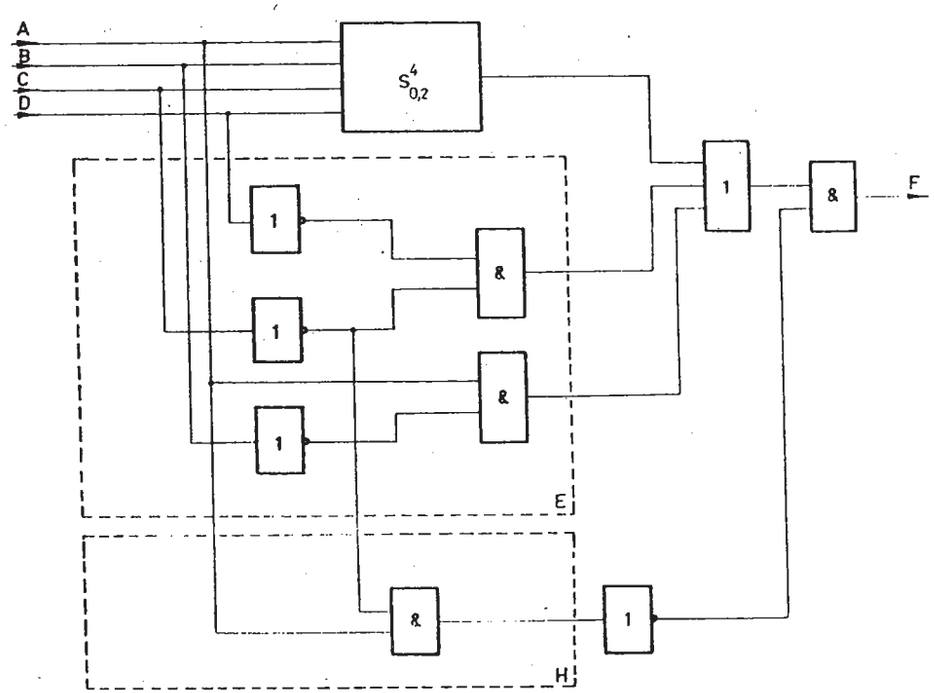
Ugyanilyen gondolatmenettel látható be, hogy az $F = S^4_{0,2} \cdot \overline{H} + E$ szerinti megvalósításban viszont megengedhető, hogy a H függvény képzésekor felhasználjunk olyan mintermeket is, amelyeknek megfelelő változókombináció esetén $E = 1$ és $F = 1$. Ekkor ugyanis az E függvénnyel megvalósított VAGY kapcsolat van közelebb a kimeneti szinthez, így az eredő függvény (F) mindig 1 értéket állít elő, ha $E = 1$. Fentiek alapján általában csak próbálgatással dönhető el, hogy az $F = (S^4_{0,2} + E) \cdot \overline{H}$ vagy az $F = S^4_{0,2} \cdot \overline{H} + E$ szerinti felépítésből adódik-e a legegyszerűbb megvalósítás az E és H függvényekre.

A 2.58. ábrán bemutatott megoldás az $F = (S^4_{0,2} + E) \cdot \overline{H}$ szerinti felépítést tételezi fel. Folytonos vonallal az E függvény, szaggatott vonallal pedig a H függvény primimplikánsait jelöltük. Látható, hogy E-nek és H-nak három közös mintermje is van, miáltal a primimplikánsokat jelentősen egyszerűsíthettük.

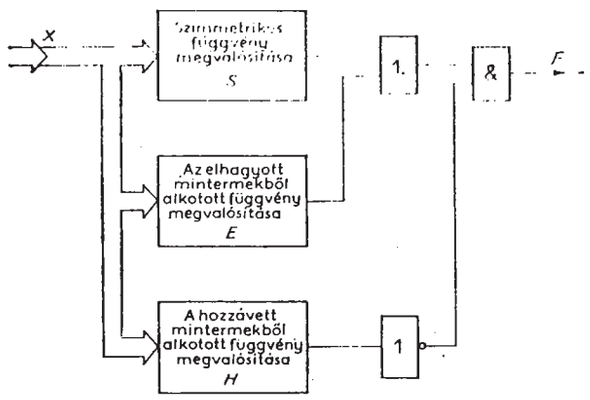
Példánkban ilyen módon a 2.60. ábrán látható elvi logikai rajzhoz juthatunk. A szükséges kapubemenetek száma alapján összehasonlítva a 2.59. ábrán és a 2.60. ábrán látható elvi logikai rajzokat, megállapíthatjuk, hogy példánkban a szimmetria létrehozásával kialakított hálózat a kedvezőbb. Ennek oka nyilvánvalóan az, hogy az E és a H jelű hálózatrészek viszonylag egyszerűek. Minél kevésbé tér el a kiindulási függvény a szimmetrikustól, annál egyszerűbbek ezek a hálózatrészek és annál érdemesebb a pl. 2.61. ábrán bemutatott és példánkban alkalmazott általános felépítést követni. Természetesen adott esetben akár az E, akár a H hálózatrész elmaradhat, ha nem szükséges mintermek elhagyása vagy hozzávétele a szimmetria biztosításához.

A fentiekben vázolt eljárások intuitív jellegűek, eredményességük megítéléséhez próbálgatásra van szükség és többnyire több szintű elvi logikai rajzhoz vezetnek. A helyzetet általában még az is nehezíti, hogy sok változó esetén már nem alkalmazhatjuk a Karnaugh-táblát a szimmetria szemléletes megállapítására. Ha a szimmetria nem derül ki egyértelműen a szöveges feladatból, akkor a szimmetria vizsgálatához előnyös lehet az alábbi eljárás.

A módszer azon alapul, hogy a szimmetriaszámok tulajdonképpen a függvényben szereplő mintermeknek megfelelő bináris kombinációk bináris súlyai. Ha egy függvény szimmetrikus, akkor a szimmetriaszámok definíciójából következően minden



2.60. ábra. A 2.58. ábrán bejelölt mintermek elhagyásával, ill. hozzávételével kialakított szimmetrikus függvény felhasználásával kapott elvi logikai rajz



2.61. ábra. Mintermek elhagyásával és hozzávételével szimmetrikussá tett függvény megvalósításának általános felépítése

egyes szimmetriaszámnak megfelelő bináris súlyt képviselő összes képezhető mintermet tartalmazza. Így, ha a változók számát n-nel, az egyik szimmetriaszámot pedig k-val jelöljük, akkor a szimmetrikus függvény tartalmazza az összes képezhető, k értékű bináris súlyt képviselő n változós mintermet, s ezek száma $\binom{n}{k}$. A szimmetriavizsgálatot tehát azzal kezdhetjük, hogy a függvényben szereplő mintermeknek megfelelő bináris kombinációkban ellenőrizzük a különböző bináris súlyok előfordulásának számát. A 2.62. ábrán példaként az

$$F = \sum^{5} (3, 4, 8, 12, 13, 14, 17, 18, 23, 27, 28)$$

függvény mintermjeinek megfelelő bináris kombinációkat és bináris súlyokat írtuk

A	B	C	D	E	Bináris súlyok
0	0	0	1	1	2
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	0	2
0	1	1	0	1	3
0	1	1	1	0	3
1	0	0	0	1	2
1	0	0	1	0	2
1	0	1	1	1	4
1	1	0	1	1	4
1	1	1	0	0	3
5/6	6/5	6/5	5/6	5/6	

2.62. ábra. Példa a bináris súlyok alapján végzett szimmetriavizsgálatra

fel. Vizsgáljuk meg, hogy minden egyes bináris súly előfordulásának száma kielégíti-e a fenti feltételt. Eszerint

az 1 bináris súlynak $\binom{5}{1} = 5$-ször,

a 2 bináris súlynak $\binom{5}{2} = 10$-szer,

a 3 bináris súlynak $\binom{5}{3} = 10$-szer,

a 4 bináris súlynak $\binom{5}{4} = 5$-ször

kellene előfordulnia. Ez az ábrán láthatóan nem teljesül. Ebből azonban még nem következik, hogy a függvény nem tehető szimmetrikussá. A további vizsgálat alapját az képezheti, hogy egy szimmetrikus függvény bármely két változóját felcserélhetjük anélkül, hogy ez a függvényt megváltoztatná. Ebből az is következik, hogy szimmetria esetén bármely két változónak azonos számúszor kell ponáltan és negáltan előfordulnia a diszjunktív normálalakban. Ez azt jelenti, hogy ha a 2.62. ábrához hasonlóan felírjuk egymás alá a szimmetrikus függvény mintermjeinek megfelelő bináris kombinációkat, akkor az egyes változók oszlopai nem különbözhetnek egymástól az 1 és 0 értékek előfordulásának számában. A 2.62. ábrán az egyes oszlopok alá írt törtszámok számlálója az oszlopban levő 1 értékek számát, a nevezője pedig az oszlopban levő 0 értékek számát jelenti. Mint látható az ábrán, a vizsgált függvény esetén ezek a törtszámok nem azonosak, tehát ez is arra utalna, hogy a függvény nem szimmetrikus. Megfigyelhetjük azonban, hogy példánkban a törtszámok között csak olyan

eltérés fordul elő, hogy egyik a másiknak a reciproka. Ezért a törtszámok azonossá tehetők, ha a megfelelő változók helyett a tagadott változókat vezetjük be, hiszen ekkor az érintett oszlopban minden érték ellentettjére változik, miáltal az oszlophoz tartozó törtszám a reciprokára változik. Az oszlopokhoz rendelt törtszámok azonossá tétele után ismét célszerű a bináris súlyok előfordulási száma szerinti ellenőrzést elvégezni, mert az elvégzett változótranszformációval felírható függvény szimmetrikus lehet. A 2.63. ábrán a B és C változók helyett azok negáltját vezettük be

A	$\bar{B}$	$\bar{C}$	D	E	Bináris súlyok
0	1	1	1	1	4
0	1	0	0	0	1
0	0	1	0	0	1
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
1	1	1	0	1	3
1	1	1	1	0	4
1	1	1	1	1	4
1	0	1	1	1	4
1	0	0	0	0	1
5/6	5/6	5/6	5/6	5/6	

2.63. ábra. A 2.62. ábra módosulása változótranszformáció hatására

változóként. Az egyes bináris súlyok előfordulásának számát ellenőrizve megállapíthatjuk, hogy

a 0 bináris súly $\binom{5}{0}=1$-szer,

az 1 bináris súly $\binom{5}{1}=5$-ször,

a 4 bináris súly $\binom{5}{4}=5$-ször fordul elő.

Így a kiindulási függvény az alábbi szimmetrikus függvénnyel azonos:

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

Természetesen úgy is kimutathattuk volna a szimmetriát, hogy a B és C változók negáltjai helyett az A, D és E változók negáltjait vezetjük be változóként a kiindulási függvényben. Ez a bemutatott műveleti tulajdonságok alapján is könnyen belátható, hiszen

$$S^5_{0,1,4}(A, \bar{B}, \bar{C}, D, E) = S^5_{1,4,5}(\bar{A}, B, C, \bar{D}, \bar{E}).$$

A vázolt szimmetriavizsgálati eljárást tehát célszerű az egyes változók oszlopaihoz tartozó törtszámok ellenőrzésével kezdeni. Ha ezek azonosak, akkor a bináris súlyok alapján eldönthető, hogy a függvény szimmetrikus-e. Ha a törtszámok közötti eltérés

csupán annyi, hogy reciprok értékek is előfordulnak, akkor a szemléltetett változótranszformáció után a bináris súlyok alapján vizsgálható tovább a szimmetria. Ha a törtszámok reciprokképzéssel nem tehetők azonossá, akkor a függvény biztosan nem szimmetrikus.

A szimmetrikus függvények megvalósítására bemutatott eljárások több kimenetű kombinációs hálózat esetén kimenetenként alkalmazhatók. Az egyes kimeneti függvények megvalósításakor természetesen célszerű törekedni a közös hálózatrészek kialakítására, ami általában pótlólagos próbálgatást igényel.

2.4. A jelterjedési idők hatása a kombinációs hálózatok működésére

A bemutatott minimalizálási eljárásokkal eljutottunk a megvalósítandó kombinációs hálózat elvi logikai rajzához, amelyről hallgatólagosan feltételeztük, hogy a kapuk a bemeneteikre jutó jelek megérkezésével egyidejűleg állítják elő a megfelelő kimeneti jelértéket, és a kapuk közötti összeköttetéseken a jelterjedés nem igényel időt. A valóságban azonban a kapuk a bemeneteikre jutó jelek megérkezése után adott késéssel képesek csak a megfelelő kimeneti jelérték előállítására, és a kapuk közötti összeköttetéseken is keletkeznek időkésések a jel terjedésének véges sebessége miatt. Nyilvánvaló, hogy a megvalósított kombinációs hálózatokban a kapuk és az összeköttetések jelterjedési késleltetése nem mindig elhanyagolható. A bemeneti kombináció változásakor az egyes jelek terjedésében mutatkozó különböző késleltető hatások átmenetileg olyan kimeneti kombinációkat kozhatnak létre, amelyeket az eddigi tervezési eljárások figyelmen kívül hagytak. Az ilyen átmenetileg létrejövő kimeneti kombinációk zavart okozhatnak a kimenetre kapcsolódó további logikai hálózatok, ill. funkcionális egységek működésében. E hatások veszélyességét fokozza az a tény, hogy az említett jelterjedési késleltetések értékei nem vehetők előzetesen pontosan számításba, mivel azok a környezeti feltételektől függően is változhatnak. Így az átmenetileg létrejövő kimeneti kombinációk is mások lehetnek a mindenkor éppen fennálló késleltetési viszonyoktól függően. Az átmeneti kimeneti kombináció fellépése tehát nem rendszeres, hanem a késleltetési viszonyok alakulásától függően tulajdonképpen véletlenszerű, ami a hiba okának felderítését a logikai rendszerben megnehezíti. Az ilyen hibajelenségeket rendszertelen fellépésük és véletlenszerű jellegük miatt *hazárdjelenségeknek* nevezik. A tervezéskor a feladattól függően gyakran arra kell törekednünk, hogy a megvalósított kombinációs hálózat működése a lehető legnagyobb mértékben független legyen a késleltetési viszonyok alakulásától.

2.4.1. A jelterjedés késleltetésének okai

A megvalósított kapuk által okozott késleltetés abból származik, hogy bármilyen is az építőelem-készlet fizikai felépítése, a kimeneti jel értéke bár rövid, de véges idő alatt változik meg. A kimeneti jel új értékének eléréséhez szükséges időt a megvalósított kapu *megszólalási idejének* nevezik.

Az integrált áramköri építőelemek katalógusaiban is fontos adat ez az idő, hiszen alapvetően ez szabja meg az adott építőelem-készletből megvalósított logikai rendszerrel elérhető legnagyobb működési sebességet. (Az angol nyelvű katalógusokban a megszólalási időt *propagation delay*-nek nevezik.)

Az építőelem-készlet technológiai fejlődésével, az integráltsági fok növekedésével egyre kisebb megszólalási idejű kapuk megvalósítása válik lehetővé, ezáltal egyre nagyobb működési sebességű logikai hálózatok, ill. logikai rendszerek hozhatók létre. Az egyre kisebb megszólalási időkkel azonban egyre jobban összemérhető a kapuk közötti összeköttetések jelterjedési késleltetése. Az összeköttetésekkel jelzett jelterjedési utak megtételéhez a jelnek egyrészt azért van szüksége véges időre, mert véges sebességgel terjed, másrészt az összeköttetések megvalósításától függően további késleltető hatások léphetnek fel. Ez utóbbiakat alapvetően az összeköttetések mentén kialakuló kapacitív és induktív hatások szabják meg. Elkerülhetetlen ugyanis, hogy az összekötő vezetéknek ne legyen ún. *szórt kapacitása*, ill. *induktivitása*. Az így kialakuló szórt, és a vezeték mentén elosztott impedanciák, olyan áramköri hatást fejtenek ki a jelterjedésre, mintha a jel késleltető-áramkörökön haladna keresztül. Minél kisebbek az adott építőelem-készletben a megszólalási idők, annál fontosabb a megvalósításban az összeköttetések jelterjedési késleltetésének csökkentése ahhoz, hogy a kis megszólalási idők által megengedett nagy működési sebességet elérhessük. Az összeköttetéseket tehát a lehető legrövidebb vezetékekkel kell megvalósítani, és a szórt kapacitásokat, ill. induktivitásokat a lehető legnagyobb mértékben le kell csökkenteni. Az integráltsági fok növekedése és a miniatürizálás ebből a szempontból is kedvező.

A megszólalási idők az építőelemek katalógusaiban általában minimális, maximális és ún. tipikus értékekkel vannak megadva. Ez azt jelenti, hogy egy konkrét megvalósított kapu megszólalási ideje a minimális és a maximális érték között bármekkora lehet (legnagyobb valószínűséggel a tipikus érték körüli). Így a megszólalási idő azonos fajtájú kapuk esetén is általában különböző, és idővel változhat is a megadott határokon belül pl. melegedés, öregedés stb. miatt.

Megváltozhat az összeköttetések jelterjedési késleltetése is, mert a szórt impedanciák értéke függ a környezeti feltételektől.

A megszólalási időknek és az összeköttetéseknek a működés során bekövetkező együttes jelterjedési késleltetései nem mérhetők megbízhatóan, mert a mérőműszer jelenléte is alapvetően befolyásolhatja például a szórt impedanciák értékét.

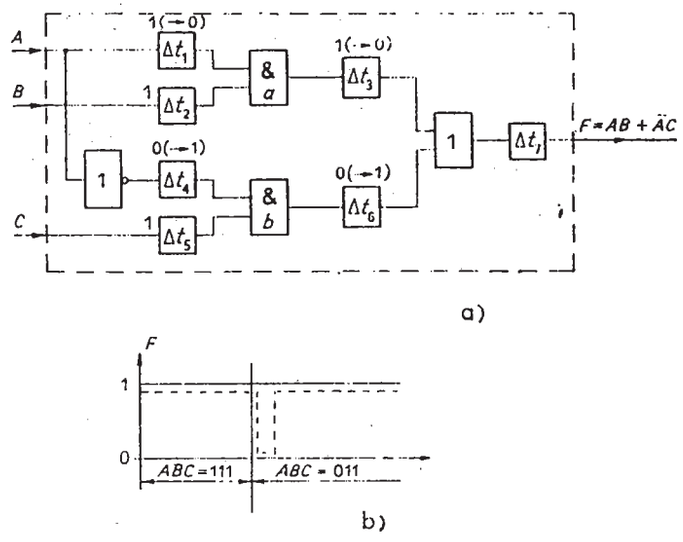
Az elvi logikai rajzon úgy tudjuk legegyszerűbben ábrázolni a késleltetéseket, hogy minden kapu bemenetére és kimenetére késleltetőelemeket helyezünk el, amelyek koncentráltan képviselik az összeköttetések jelterjedési késleltetéseit és a megszólalási időket. Ilyen elemek berajzolása után már feltételezzük, hogy a kapuk megszólalási ideje zérus értékű és a vezetékeknek sincs késleltetésük. A koncentrált késleltetések ilyen feltételezésével természetesen nem írhatjuk le minden szempontból pontosan a valóságos helyzetet, mert például a szórt impedanciák elosztott jellegét nem tudjuk figyelembe venni. A hazárdjelenségekkel kapcsolatos további vizsgálataink azonban a koncentrált késleltetések feltételezése elegendő.

2.4.2. A statikus hazárd

A további vizsgálatok céljából példaként a 2.64. ábrán látható elvi logikai rajzot kiegészítettük a koncentrált késleltetésekkel. Az egyes késleltetési időket a későbbi hivatkozás érdekében $\Delta t_1, \ldots, \Delta t_7$-tel jelöltük. Az ábrán szereplő egyéb jelzéseket egyelőre ne vegyük figyelembe. A hazárdjelenségek vizsgálatakor először feltételezzük, hogy a hálózatot szomszédos bemeneti változás éri, vagyis csak egyetlen bemeneti jel változásának hatását követjük. A 2.64. ábrán látható elvi logikai rajzzal adott kombinációs hálózat kimenete az

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

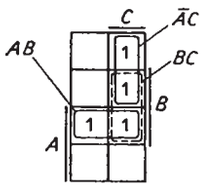
logikai függvényt valósítja meg. Az ábrázolt elvi logikai rajzhoz vezető összevonást



2.64. ábra. Példa az elvi logikai rajznak a koncentrált késleltetésekkel történő kiegészítésére és a statikus hazárd szemléltetésére

a 2.65. ábrán látható Karnaugh-táblán követhetjük. A szaggatottan jelölt BC prímimplikánst egyelőre ne vegyük figyelembe.

Vizsgáljuk meg a hálózat működését abban az esetben, amikor az $ABC=111$ bemeneti kombináció a vele szomszédos $ABC=011$ bemeneti kombinációra változik. Ez azt jelenti, hogy B és C értéke változatlan marad, csupán az A logikai változó értéke változik 1-ről 0-ra. Ezt az elvi logikai rajzon is feltüntettük. Nyilvánvaló, hogy az A logikai változó $1\to0$ átmenete az inverter kimenetén $0\to1$ átmenetet jelent.



2.65. ábra. Példa statikus hazárd vizsgálatára

Tételezzük fel, hogy a vizsgálat kezdetének pillanatában a kombinációs hálózat minden elemének kimenete megegyezik az $ABC=111$ bemeneti kombináció által az elemre előírt értékkel, továbbá minden egyes koncentrált késleltetés bemeneti és kimeneti értéke is megegyezik. Ilyenkor az a jelű ÉS kapu kimenetén 1, a b jelű ÉS kapu kimenetén 0, a VAGY kapu és ezzel hálózat kimenetén 1 van.

A bemenetekre ezután az $ABC=011$ kombináció kerül. Erre a bemeneti kombinációra a kimeneti függvény értéke nem változhat, hiszen a logikai függvény erre a bemeneti kombinációra is 1 értékű kimenetet ír elő. Nézzük meg, hogy a kimenet állandó értékű marad-e a késleltetések egymáshoz képesti arányaitól függetlenül is.

A jelváltozásokat a 2.64. ábrán a megfelelő koncentrált késleltetések fölé írva jelöltük.

Vizsgálatunkkal arra az esetre szorítkozunk, amikor a koncentrált késleltetések értékei között a $\Delta t_1+\Delta t_3<\Delta t_4+\Delta t_6$ egyenlőtlenség áll fenn. Ekkor ugyanis a VAGY kapunak az a jelű ÉS kapu kimenetével összekötött bemenetén előbb megy végbe az $1\to0$ átmenet, mint a másik bemenetén a $0\to1$ átmenet, vagyis a VAGY kapu felső bemenetén előbb tűnik el az 1 érték, mint az alsó bemenetén megjelenne a b jelű kapu által szolgáltatott 1 érték. Ennek következtében a VAGY kapu bemenetén átmenetileg kialakul a 00 kombináció, ami a hálózat kimenetén 0 értéket hoz létre. További $(\Delta t_4+\Delta t_6)-(\Delta t_1+\Delta t_3)$ idő elteltével természetesen a VAGY kapu alsó bemenetén megjelenik az 1 és ez a hálózat kimenetét is 1-be viszi. Így tehát ahelyett, hogy a kimenet értéke változatlan maradt volna, az $1\to0\to1$ hibás értékváltozás zajlott le (2.64*b)* ábra).

Nyilvánvaló, hogy ez a hibás működés a késleltetési viszonyoktól függően nem minden esetben lép fel. (Például, ha a VAGY kapu alsó bemenetén előbb jelenik meg az 1 érték, mint a felsőn a 0.) Azt azonban látni kell, hogy a *hálózat felépítése* a hibás működés lehetőségét magában hordozza. Nyilvánvaló ugyanis, hogy a vizsgált változással ellentétes bemeneti változás éppen fordított késleltetési arányok esetén okoz

hazárdjelenséget. A késleltetések egymáshoz képesti aránya idővel megváltozhat a megépített hálózatban öregedés, melegedés, kapacitásváltozás stb. miatt. A továbbiakban feladatunk ennek a bizonytalanságnak a kiküszöbölése.

A jelenséget először megfogalmazzuk általánosabban is. Adott két szomszédos bemeneti kombináció X és X', és e két bemeneti kombináció fellépésekor a függvényérték azonos, $F(X)\equiv F(X')$. Ha a bemeneti kombináció az adott két szomszédos kombináció egyikéről a másikára változik, és ezalatt a kimenetén átmenetileg olyan F^* érték jelenik meg, amelyre $F^*\neq F(X)=F(X')$, vagyis a kimeneten az $F\to F^*\to F$ függvényérték-sorozat játszódhat le a késleltetési viszonyoktól függően, akkor a hálózatban ún. *statikus hazárd* van.

A statikus hazárd oka, mint a hálózat elvi logikai rajzán láttuk a két ÉS kapu kimenetének ellentétes változása. A két ÉS kapu, mint tudjuk két prímimplikánst valósít meg. Nyilvánvalóan, amíg például az AB prímimplikánshoz (2.65. ábra) tartozó bemeneti kombinációk között vannak a változások, addig az a jelű ÉS kapu kimenetén biztosan 1 van. Ugyanígy az $\bar{A}C$ kettes hurokhoz tartozó két bemeneti kombináció közötti közvetlen átmenetek esetén a b jelű ÉS kapu kimenetén van állandóan 1.

Ebből következik, hogy a példában szereplő $ABC=111\to011$ bemeneti kombinációváltozás során sem működhetne hibásan a hálózat, ha a fenti két bemeneti kombináció is közös prímimplikánshoz tartozna. E célból, egy újabb kettes hurok, a BC prímimplikáns megvalósítása szükséges és ezzel a hálózat is egy újabb ÉS kapuval egészül ki. Ennek az ÉS kapunak a kimenetén mindaddig folyamatosan 1 van, amíg a bemeneti kombináció a BC prímimplikánshoz tartozó értékek között változik.

Az előbbiek alapján általánosan is megfogalmazhatjuk a kétszintű ÉS—VAGY hálózatok statikus hazárdtól mentes megvalósításának a feltételét.

A kétszintű, ÉS—VAGY (diszjunktív) hálózat akkor és csak akkor mentes a statikus hazárdtól, ha az 1 kimeneti értéket előállító bemeneti kombinációk közül bármely két szomszédoshoz található legalább egy olyan ÉS kapu, amelynek kimenete mindkét szomszédos bemeneti kombináció esetén 1 értékű. Ez tulajdonképpen azt jelenti, hogy a diszjunktív alakban bármely két szomszédos mintermhez található legalább egy olyan prímimplikáns, amely mindkét mintermet lefedi. A legegyszerűbb hazárdmentes diszjunktív alak tehát úgy állítható elő, hogy a legegyszerűbb diszjunktív alakot a lehető legkevesebb és legegyszerűbb prímimplikánssal egészítjük ki a fenti feltétel biztosítása érdekében.

Megjegyezzük, hogy fenti megállapításaink természetesen a legegyszerűbb hazárdmentes kétszintű VAGY—ÉS (konjunktív) hálózatok tervezésére is érvényesek. A módosítás csupán annyi, amennyi a konjunktív alakból definíciószerűen is következik, hogy ugyanis a 0 kimeneti értéket előállító bemeneti kombinációk közül bármely két szomszédoshoz tartoznia kell legalább egy olyan közös VAGY kapunak, amely az említett bemeneti kombinációk fellépése esetén a 0-t biztosan tartja a kimeneten. Megállapíthatjuk, hogy diszjunktív alak (ÉS—VAGY) esetén a kimeneten a 0-t tartó, konjunktív alak (VAGY—ÉS) esetén pedig az 1-et tartó szomszédos

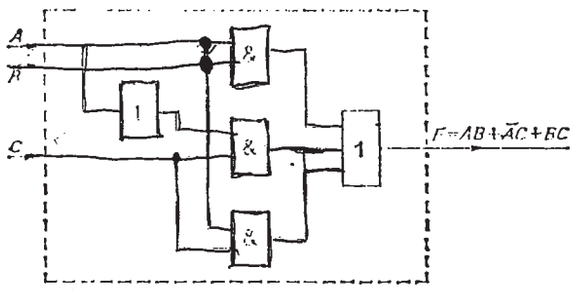
bemeneti változások nem okozhatnak statikus hazárdot. Az ilyen bemeneti változások ugyanis nem idéznek elő jelváltozást a megfelelő hálózat egyetlen pontján sem.

Az elmondottakból következik, hogy a kétszintű ÉS—VAGY hálózatok statikus hazárdtól mentes alakja nem feltétlenül a legegyszerűbb diszjunktív alak, hiszen logikailag redundáns prímimplikánst is tartalmazhat.

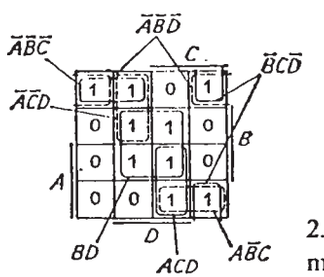
Az előbbiek szerint a példában szereplő hálózat legegyszerűbb, statikus hazárdtól mentes kialakítása a 2.66. ábrán látható. Az ábrán a koncentrált késleltetéseket nem tüntettük fel, hiszen azokat a továbbiakban külön jelölés nélkül is mindig fel kell tételeznünk.

Példaként határozzuk meg az $F(A, B, C, D) = BD + \bar{A}\bar{B}\bar{C} + ACD + \bar{B}C\bar{D}$ legegyszerűbb diszjunktív alakjával adott kombinációs hálózat legegyszerűbb hazárdmentes kétszintű ÉS—VAGY alakú elvi logikai rajzát.

A függvényt ábrázoltuk Karnaugh-táblán a 2.67. ábrán. A táblába berajzoltuk



2.66. ábra. Az $F(A, B, C) = AB + \bar{A}C$ logikai függvény legegyszerűbb, statikus hazárdtól mentes, kétszintű megvalósítása

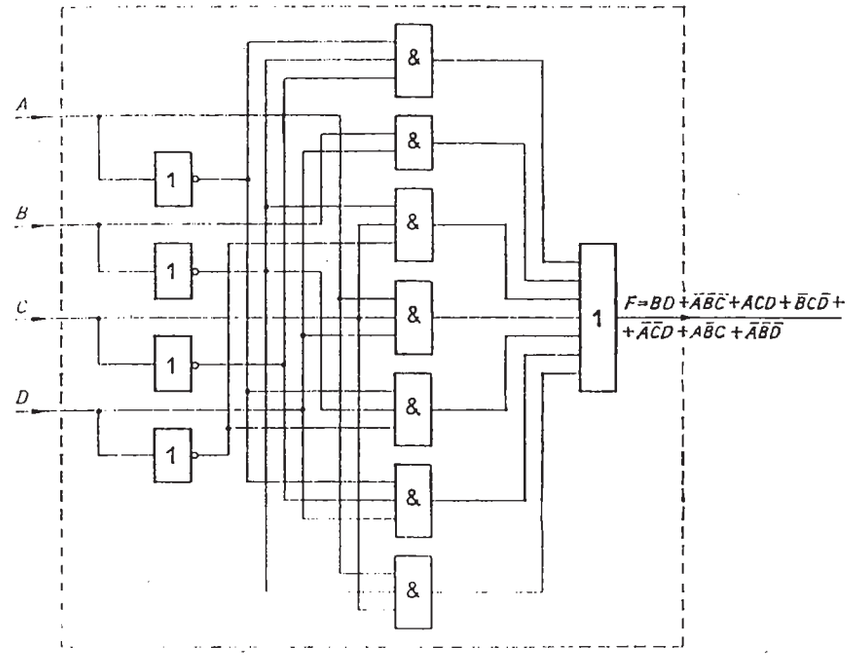


2.67. ábra. Példa a statikus hazárdot kiküszöbölendő minimalizálásra

a legegyszerűbb diszjunktív alaknak megfelelő hurkokat. Világosan látszik, hogy a szaggatott vonallal kijelölt prímimplikánsokat is meg kell valósítani a hazárdmentesség érdekében, hiszen az ezekhez tartozó bemeneti kombinációk is szomszédosak. Emiatt a fentiek értelmében bármely kettőhöz tartoznia kell legalább egy olyan prímimplikánsnak, amely mindkettőt lefedi. Így a diszjunktív alak három prímimplikánssal bővül. A kiadódó elvi logikai rajz a 2.68. ábrán látható.

Eddigi egykimenetű, teljesen határozott hazárdmentesítési példáinkból levonhatjuk az alábbi következtetést:

A legegyszerűbb, statikus hazárdtól mentes kétszintű elvi logikai rajzot — teljesen



2.68. ábra. Az $F = BD + \bar{A}\bar{B}\bar{C} + ACD + \bar{B}C\bar{D}$ függvény legegyszerűbb, statikus hazárdtól mentes, kétszintű elvi logikai rajza

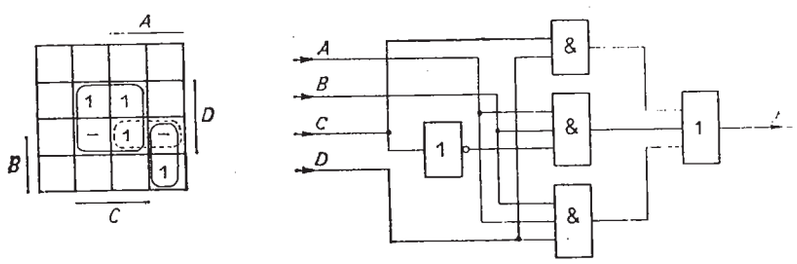
határozott logikai függvény esetében — akkor kapjuk meg, ha az *összes prímimplikánst* megvalósítjuk.

Az állítás helyességéről legkönnyebben indirekt módon győződhetünk meg. Tételezzük fel, hogy létezik egy olyan prímimplikáns, amelynek megvalósítása nem szükséges a statikus hazárdmentesítés szempontjából. Ez a fentiek értelmében csak úgy képzelhető el, hogy ez a prímimplikáns nem fed le egyetlen olyan szomszédos mintermpárt sem, amelyet valamelyik másik prímimplikáns ne fedne le. Így a feltételezett prímimplikáns bármely két szomszédos mintermjét legalább egy másik prímimplikáns is lefedi. Ebből viszont az következik, hogy a feltételezett prímimplikáns azonos valamelyik másik prímimplikánssal, amelyről viszont kiindulásként feltételeztük, hogy megvalósítása szükséges a legegyszerűbb kétszintű, hazárdmentes hálózatban.

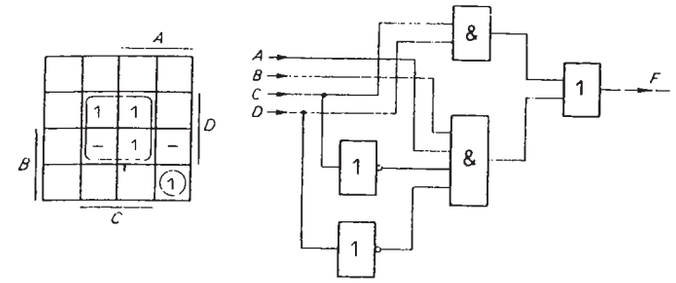
Több kimenetű kombinációs hálózatokban a statikus hazárd jelensége minden kimeneten felléphet. Kiküszöböléséhez — ha a kimenetek függvények teljesen határozottak — biztosítani kell, hogy minden egyes kimeneti függvény bármely két szomszédos mintermjéhez található legyen a megvalósításban legalább egy olyan prímimplikáns, amely mindkettőt lefedi. Ezért a legegyszerűbb kétszintű, statikus hazárdtól mentes elvi logikai rajzot úgy kaphatjuk meg, hogy kimenetenként az összes prímimplikánst megvalósítjuk. A közös prímimplikánsokat természetesen elegendő egyszer megvalósítani.

Teljesen határozott függvények esetén tehát a megismert minimalizáló eljárások segítségével úgy határozhatjuk meg a legegyszerűbb, kétszintű statikus hazárdtól mentes elvi logikai rajzot, hogy az eljárások által előállított összes prímimplikánst megvalósítjuk, azaz a prímimplikánstábla képzését és a lefedést elhagyjuk.

Ha a tervezendő kombinációs hálózatot leíró logikai függvények nem teljesen határozottak, akkor a legegyszerűbb, statikus hazárdtól mentes kétszintű elvi logikai rajz meghatározása sokkal kevésbé szisztematikus. Ennek szemléltetésére vizsgáljuk meg a 2.69. ábrán Karnaugh-táblával adott logikai függvény hazárdmentesítési lehetőségeit. Az ábrán az eddigi megállapításaink alapján a legkedvezőbb prímimplikánselőállító hatásuknak megfelelően rögzítve a közömbös bejegyzéseket, a szaggatottan jelölt hazárdmentesítő prímimplikánssal együtt az összes prímimplikánst megvalósítottuk. A kérdés csupán az, hogy valóban szükség van-e a szaggatottan jelölt prímimplikánsra. Ha a feladat olyan, hogy a közömbös bejegyzésekhez tartozó bemeneti kombinációk biztosan nem léphetnek fel, akkor természetesen a szaggatottan jelölt prímimplikáns megvalósítása nem szükséges, hiszen az általa lefedett szomszédos bemeneti változás nem lép fel, tehát hibás átmeneti kimeneti érték sem keletkezhet. Ha a közömbös bejegyzésekhez tartozó bemeneti kombinációkat úgy értelmezzük, hogy felléphetnek ugyan, de a kimenet értékére semmiféle előírás nincs, akkor szintén felesleges azokat a szomszédos mintermpárokat figyelembe venni a statikus hazárdmentesítés szempontjából, amelyeknek legalább az egyikéhez közömbös bejegyzés tartozik. Az ilyen mintermpárok olyan szomszédos bemeneti kombinációkat képviselnek, amelyek közül legalább az egyikhez nincs előírva a függvény értéke, vagyis az egyik ilyen kombinációról a másikra történő változáskor a kimenet értéke átmeneti állapotban sincs előírva. Ez azt jelenti, hogy a statikus hazárd jelensége sem minősíthető hibának. A 2.69. ábrán szaggatottan jelölt prímimplikánst tehát ebben az esetben sem kell megvalósítani. Természetesen ilyenkor is sokszor zavaró lehet, hogy a kimeneten lejátszódó jelváltozások mindig a pillanatnyi késleltetési viszonyoktól függenek. A közömbös kimeneti értéket sem értelmezhetjük mindig ilyen szabadon. Előfordulhat, hogy a tervezendő kombinációs hálózat kimeneteire kapcsolódó logikai hálózatok működésében akkor is zavart okoznak a pillanatnyi késleltetési viszonyoktól függő átmeneti változások, ha a változás során az állandósult értékek közül leg-



2.69. ábra. Példa nem teljesen határozott logikai függvénynek a statikus hazárdot kiküszöbölő egyszerűsítésére



2.70. ábra. Példa nem teljesen határozott logikai függvénynek a statikus hazárdot kiküszöbölő egyszerűsítésére

alább az egyik tetszőleges lehet. A kimenetre ugyanis olyan további hálózatok is kapcsolódhatnak, amelyek nem az állandósult jelértékekre, hanem a jelváltozásokra érzékenyek. Ezek sorrendi hálózatok, és tervezésük során nehézséget okoz, ha a megelőző hálózat statikus hazárdja miatti bemeneti változás bekövetkezése a pillanatnyi késleltetési viszonyoktól függ. Ilyenkor tehát a statikus hazárdot ki kell küszöbölnünk, és úgy tűnhet, hogy a 2.69. ábrán szaggatottan jelölt prímimplikáns megvalósítása feltétlenül szükséges. Nem szabad azonban megfeledkeznünk arról, hogy a *legegyszerűbb* kétszintű statikus hazárdtól mentes elvi logikai rajz meghatározása a célunk. Ehhez pedig az adott példában a 2.70. ábra szerinti összevonás szükséges, mert így kevesebb kapubemenet adódik.

Láthatjuk tehát, hogy a nem teljesen határozott logikai függvények statikus hazárdtól mentes egyszerűsítése a közömbös függvényértékek értelmezésétől függ. Ha a közömbös bejegyzésekre vonatkozóan is el kell végezni a hazárdmentesítést, akkor — amint azt a 2.70. ábra szemléltette — a közömbös értékek rögzítésében nem mindig a legegyszerűbb prímimplikánsok előállítása az egyedüli szempont. Az egyszerűsítés ilyenkor általában nem nélkülözheti a próbálgatásos lépéseket, mert a közömbös bejegyzések adott rögzítésének hatását csak a szükséges hazárdmentesítő prímimplikánsok meghatározása után ítélhetjük meg.

Ha a közömbös kimeneti értékekre vonatkozóan nem kell hazárdmentesítést végezni, akkor a megismert egyszerűsítési eljárások kiegészítésével szisztematikus módszerrel határozhatjuk meg a legegyszerűbb, statikus hazárdtól mentes, kétszintű elvi logikai rajzot. Ilyenkor az összes prímimplikáns képzését ugyanúgy végezhetjük, mint a hazárdmentesítés nélküli esetben. A prímimplikánstáblában azonban a lefedendő mintermeken kívül lefedendő elemekként kell feltüntetnünk — több kimenetű esetben kimenetenként — mindazokat a két mintermből álló szorzatokat, amelyek nem tartalmaznak közömbös kimeneti értékhez tartozó mintermet. Ezáltal a közömbös kimeneti értékeket a legegyszerűbb prímimplikánsok képzésének megfelelően rögzítjük, de a közömbös mintermet is tartalmazó szorzatok megvalósítását nem írjuk elő a statikus hazárdmentesség feltételeként.

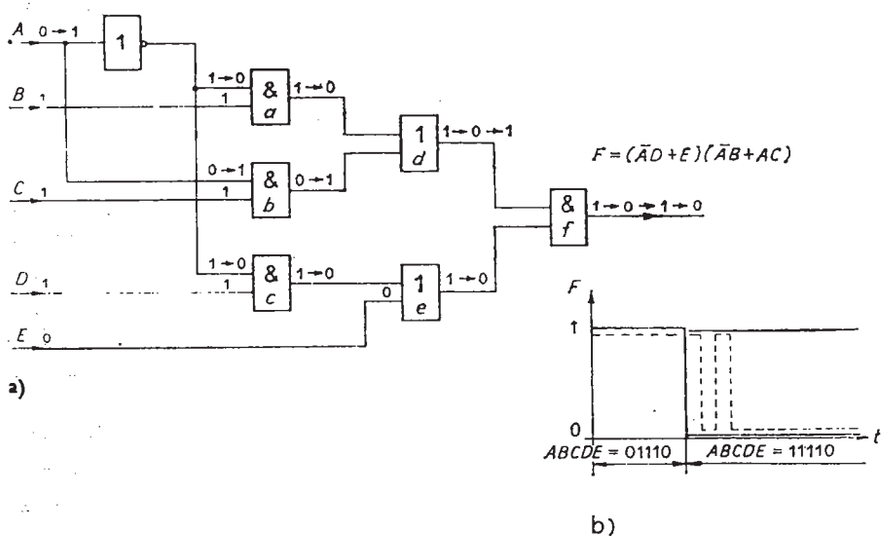
A megismert egyszerűsítési eljárások tehát a statikus hazárd kiküszöbölését próbálgatásos lépések közbeiktatásával, ill. a prímimplikánstábla növelésével — azaz bonyolultabb lefedési segédfüggvénnyel — teszik lehetővé. Az így módosított eljárások részleteire az alapgondolat fenti ismertetésén túlmenően nem térünk ki, mert már viszonylag kevés változó esetén is indokolt a számítógépes végrehajtás.

2.4.3. A dinamikus hazárd

Vizsgáljuk meg a 2.71*a)* ábrán elvi logikai rajzával adott háromszintű kombinációs hálózat működését a koncentrált késleltetések feltételezésével. Az elvi logikai rajz alapján felírható logikai függvény:

$$F(A, B, C, D, E) = (\bar{A}D+E)(\bar{A}B+AC).$$

Az algebrai alakról is látszik, hogy a hálózat nem kétszintű, hanem két kétszintű hálózat ÉS kapcsolata, azaz ÉS—VAGY—ÉS szintekre bontható.



2.71. ábra. Példa dinamikus hazárdra

Vizsgáljuk meg a hálózat működését az $ABCDE=01110$ bemeneti kombinációinak a vele szomszédos 11110-ra való változása esetén. A 01110-hoz 1 kimeneti érték, az 11110-hoz 0 kimeneti érték tartozik. A 2.71. ábrán feltüntettük az egyes kapuk bemenetein és kimenetein lejátszódó jelváltozásokat. Figyeljük meg, hogy az *a*, *b* és *d* jelű kapuk által alkotott kétszintű hálózatrészben nincs kiküszöbölve a statikus hazárd. Ezért a késleltetési viszonyok kedvezőtlen alakulása esetén előfordulhat,

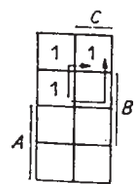
hogy a vizsgált bemeneti változás során a *d* jelű kapu kimenetén és az *f* jelű kapu felső bemenetén $1\to0\to1$ jelsorozat játszódik le. Ha a késleltetési viszonyokról még azt is feltételezzük, hogy az *e* jelű kapu kimenetén csak azután jelenik meg a 0, miután az *f* jelű kapu felső bemenetén az $1\to0\to1$ jelsorozat már lejátszódott, akkor a hálózat kimenetén az $1\to0\to1\to0$ jelsorozat lesz észlelhető. Az egyszeri $1\to0$ változás helyett tehát kétszeri változást tapasztalhatunk (2.71*b)* ábra). A jelenséget *dinamikus hazárdnak* nevezik, és ezt az alábbiakban általánosan is megfogalmazzuk.

Legyen adott két szomszédos X és X^* bemeneti kombináció, amelyekhez eltérő kimeneti érték tartozik, azaz $F(X)\neq F(X^*)$. Ha a bemeneti kombináció X és X^* közötti változásának hatására a kimeneten az $F(X)\to F(X^*)$ változás helyett $F(X)\to F(X^*)\to F(X)\to F(X^*)$ változás játszódhat le a késleltetési viszonyoktól függően, akkor a hálózat működésében dinamikus hazárd van.

Láttuk, hogy a több szintű hálózatban a dinamikus hazárd fellépését az okozta, hogy a hálózat valamelyik szintjén nem volt kiküszöbölve a statikus hazárd. Így, ha a több szintű hálózat minden szintjén kiküszöböljük a statikus hazárdot, akkor dinamikus hazárd sem léphet fel a kimeneten. A több szintű kombinációs hálózatok hazárdmentes megvalósításához tehát szintenként el kell végezni a statikus hazárdmentesítést a megismert módon.

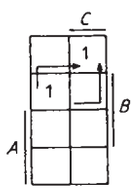
2.4.4. A funkcionális hazárd

Az eddigiekben csak szomszédos bemeneti kombinációváltozások mellett vizsgáltuk a jelterjedési késleltetések hatását. Ha a szomszédosságot nem tételezzük fel, akkor azt kell megvizsgálnunk, hogy a jelterjedési késleltetések hatására milyen jelenségek játszódnak le a hálózat kimenetén a bemeneti kombináció tetszőleges megváltozása esetén. Vizsgáljuk meg példaként, hogy a 2.72. ábra Karnaugh-táblájával megadott háromváltozós logikai függvény $ABC=010$ bemeneti kombinációja 001-re történő változásának hatására milyen jelenségek játszódhatnak le. A bemeneti változás során a *B* és a *C* változó vált értéket. A jelterjedési késleltetések miatt előfordulhat, hogy a hálózat egyes kapubemeneteihez a *B* változó megváltozása, más kapubemeneteihez pedig a *C* változó megváltozása jut el előbb. Így a hálózat egyes részei, a $010\to000\to001$ (*B* változik előbb) kombinációsorozatot, más részei pedig a $010\to011\to001$ (*C* változik előbb) kombinációsorozatot érzékelik. Nyilvánvaló, hogy az átmenetileg kialakuló közbenső kombinációk különböző jelsorozatokat hozhatnak létre a kimeneten a késleltetési viszonyoktól függően. A 2.72. ábrán nyilak-



2.72. ábra. Példa funkcionális hazárdra

kal jelöltük a kétféle változási sorozatot. Láthatjuk, hogy a 000 közbenső bemeneti kombináció kialakulása esetén a kimeneten nem jön létre hibás érték, de a 011 közbenső kombináció átmenetileg 0-t hoz létre a kimeneten. A hibás átmeneti érték fellépése a jelterjedési késleltetések pillanatnyi értékétől függ, vagyis attól, hogy a nem szomszédos bemeneti változást a hálózat egyes részei milyen szomszédos változások sorozataként érzékelik. A jelenséget *funkcionális hazárdnak* nevezik, és minden olyan esetben hibát okozhat a kimeneten, amikor a bemeneti változások nem szomszédosak. A funkcionális hazárd kiküszöbölésének egyik lehetséges módja bizonyos esetekben az, hogy a hálózatba szándékosan beépített késleltetésekkel úgy állítjuk be a jelterjedési késleltetések értékeit, hogy azok minden lehetséges megváltozása mellett is a hálózat minden kapubemenetén csak olyan közbenső kombinációk alakulhassanak ki (példánkban 000), amelyek nem hoznak létre átmeneti hibát a kimeneten. A késleltetések beépítése legegyszerűbben egymás után kapcsolt páros számú INVERTER-rel valósítható meg. Ez a megoldás természetesen lassítja a hálózat működését és ritkán vezet eredményre. Ha ugyanis az ellentétes bemeneti változás is létre jöhet a működés során, akkor az előbbi módon beépített késleltetések nyilvánvalóan éppen rossz irányban módosítják az arányokat. A 2.73. ábrán szerepelő függvény esetében nem is érdemes késleltetést beépíteni.



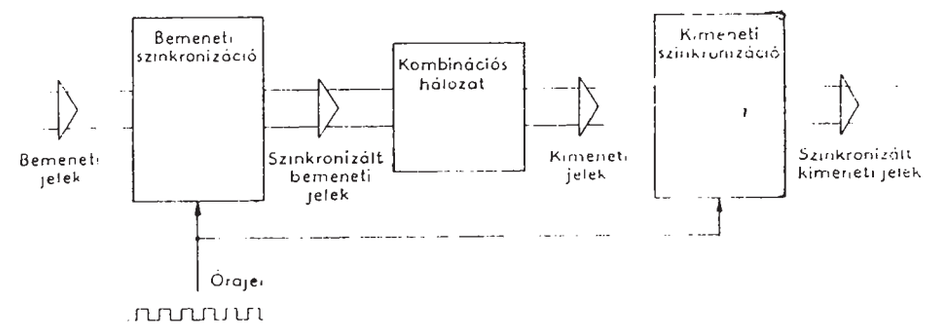
2.73. ábra. Példa a közbenső bemeneti kombinációk biztosításával nem kiküszöbölhető funkcionális hazárdra

Az ABC=010-ról 001-re történő bemeneti változás ugyanis mindkét közbenső bemeneti kombináció kialakulása mellett hibás átmeneti kimeneti értéket okoz. Ebben az esetben tehát nem találunk olyan közbenső bemeneti kombinációt, amely beépített késleltetésekkel a hibás kimeneti változásokat elhárítja.

A funkcionális hazárd kiküszöbölésének legbiztosabb módja az, hogy a hálózat bemenetén nem engedjük meg a nem szomszédos változásokat. Ezt természetesen a bemeneti kombinációkat előállító egységek működési feltételeiben kell megvalósítani. Feltételezésünk szerint a tervezendő kombinációs hálózat a logikai rendszer, ill. funkcionális egységek része, tehát bemeneti kombinációit más logikai hálózatok, funkcionális egységek vagy akár építőelemek állítják elő. Így a bemeneti kombinációváltozások szomszédosságára vonatkozó előírás gyakran tovább gyűrűzik és visszahat az egész logikai rendszer működési feltételeire.

Gyakran úgy küszöbölik ki a funkcionális hazárdot, hogy az átmeneti változások idejére a kombinációs hálózat kimenetének a logikai rendszer további részeire gyakorolt hatását letiltják. Ehhez természetesen ismerni kell a kombinációs hálózat bemenetein a változások időpontjait, és ezekhez képest adott késleltetéssel szabad csak engedélyezni a kimenet értelmezését. A bemeneti változások időpontjait akkor

tehetjük a legegyszerűbben ismertté, ha a bemeneti jelekből egy állandó frekvenciájú impulzussorozattal, egy ún. órajellel mintát veszünk, és az ilyen módon keletkező mintákból az ún. *szinkronizált* bemeneti jelekből állítjuk elő a kombinációs hálózat bemeneti kombinációit. Így a szinkronizált bemeneti kombináció csak az órajel meghatározott értékénél változhat, ami lehetőséget ad a kimeneti kombináció órajelhez képesti értelmezési tartományának a hazárdjelenséget kizáró kijelölésére (2.74. ábra). A bemeneti és kimeneti szinkronizáció megvalósításának részletes magyarázatához szükségesek a sorrendi hálózatokkal kapcsolatos alapfogalmak, azért erre csak a későbbiekben kerülhet sor. Több kimenetű kombinációs hálózatokban kimeneti szinkronizáció alkalmazásával is előfordulhat, hogy még szomszédos bemeneti változások hatására is nem szomszédos kombinációváltozás van előírva a kimeneten. Ha az ilyen eseteket a feladat megfogalmazásakor nem tudjuk elkerülni, akkor a nem szomszédos kimeneti változás által okozott funkcionális hazárd esetleges következményeit a logikai rendszer további egységeiben feltétlenül meg kell vizsgálnunk és szükség esetén gondoskodnunk kell a hazárdmentesítésről.



2.74. ábra. Funkcionális hazárd kiküszöbölése szinkronizációval

A bemutatott hazárdjelenségek hatására létrejövő átmeneti kimeneti változások természetesen nem minden esetben okoznak zavart. Láttuk, hogy az átmeneti értékek fennállásának időtartama az adott jelterjedési késleltetések különbségétől függ és minél gyorsabb működésűek az építőelemek, annál kisebb értékű koncentrált késleltetésekkel helyettesíthetjük a jelterjedési késleltetések összhatását. Kis és közepes integráltsági fokú normál építőelemek alkalmazásakor például kétszintű megvalósításban a hazárdjelenségek által okozott átmeneti kimeneti értékek általában nem állnak fenn 10–20 ns-nál hosszabb ideig. Nyilvánvaló, hogy az ilyen rövid ideig fennálló hibás kimeneti kombinációk csak akkor lehetnek károsak, ha azok az egységek, amelyeknek a bemenetére jutnak, érzékenyek az ilyen rövid ideig fennálló jelekre. Ha a kimenetre kapcsolódó egységek például a logikai rendszer környezetéhez sorolt beavatkozó, kijelző stb. berendezések teljesítményerősítői, akkor a hazárdjelenségek általában nem okoznak hibát. A teljesítményerősítők ugyanis viszonylag lassú működésűek, és a logikai rendszer környezete nem mindig érzékeny a hazárd-

jelenségek által okozott, rövid ideig fennálló átmeneti kimeneti értékekre. Ilyenkor tehát nincs szükség a megismert hazárdmentesítő eljárásokra. Általában azonban a hazárdjelenségek várható következményeit a tervezéskor gondosan kell mérlegelni, és ettől kell függővé tenni azt, hogy szükséges-e a hazárdmentesítés.

2.5. A több szintű kombinációs hálózatok tervezésének néhány alapgondolata

Már az elvi logikai rajz fogalmának bevezetésekor megállapítottuk, hogy integrált áramköri építőelem-készlet esetében nincs áramköri akadálya kettőnél több szintet tartalmazó kombinációs hálózatok megépítésének. A több szintű hálózatok várhatóan gazdaságosabbak a kétszintűeknél a felhasznált építőelemek száma és az összeköttetések egyszerűsége szempontjából. Ezt az is alátámasztja, hogy például a legegyszerűbb diszjunktív alak nem mindig a legegyszerűbb algebrai alakja a függvénynek. Például az

$$F(A, B, C, D) = AB+\bar{C}D+AC$$

kifejezés a legegyszerűbb diszjunktív alak, de nem a legegyszerűbb algebrai alak, mert kiemeléssel tovább egyszerűsíthető:

$$F(A, B, C, D) = A(B+C)+\bar{C}D.$$

A kiemeléssel kapott algebrai alak azonban se nem diszjunktív, se nem konjunktív, tehát több szintű elvi logikai rajzot eredményezne. A szintek számának növekedésével azonban növekszik a jelterjedési idő, vagyis csökken a hálózat működési sebessége.

A több szintű hálózatok tervezési módszereinek egyértelmű megfogalmazása nehezebb, és sokkal kevésbé tehetők szisztematikussá az eljárások. Sok esetben a végeredményként adódó több szintű hálózat gazdaságosságát csak úgy tudjuk értékelni, hogy összehasonlítjuk a kétszintű megoldással.

A hazárdjelenségek vizsgálata és kiküszöbölése — mint láttuk — több szintű esetben sokkal bonyolultabb feladat, mint a kétszintű hálózatokban. Elég, ha csak arra gondolunk, hogy a dinamikus hazárd kiküszöböléséhez szintenként statikus hazárdtól mentesnek kell lennie a megoldásnak.

A szimmetrikus függvények megvalósítására bemutatott eljárások több szintű hálózatot hoztak létre, és a 2.61. ábrán vázolt általános felépítés alapgondolata más eljárások esetén is követhető.

A továbbiakban kétfajta módszer alapgondolatát ismertetjük a több szintű kombinációs hálózatok megvalósítására. Az egyik a megvalósítandó logikai függvvény felbontásán, azaz dekompozícióján alapul, a másik pedig NAND, ill. NOR kapuk

felhasználásaival algebrai átalakítások útján teszi lehetővé a több szintű elvi logikai rajz létrehozását.

A bemutatandó két módszeren kívül más eljárások is léteznek, de azok is viszonylag kismértékben szisztematikusak, lényegében próbálgatásos jellegűek és legtöbbjük felhasználja valamilyen formában a bemutatandó két módszer alapgondolatát.

2.5.1. Több szintű kombinációs hálózatok tervezése a logikai függvények dekompozíciójának elvén

A logikai függvények dekompozíciója a függvény felbontását jelenti egyszerűbb részfüggvényekre. A felbontást folytatva a részfüggvényekre is, hierarchikus felépítésű több szintű hálózathoz juthatunk. A dekompozíciót sokféleképpen elvégezhetjük attól függően, hogy milyen követelményeket támasztunk a részfüggvényekkel, ill. részhálózatokkal, valamint a kialakuló több szintű hálózattal szemben. Láttuk, hogy a több szintű kombinációs hálózatokban a statikus és dinamikus hazárdot szintenként kell kiküszöbölni, ami nehezen fogalmazható meg a tervezés menetére vonatkozóan. Válasszunk ezért olyan dekompozíciós módot, amelynek alkalmazása esetén a kialakuló részhálózatokat elegendő önmagukban hazárdmentesíteni ahhoz, hogy a teljes több szintű hálózat hazárdmentes legyen. Ilyen előnyös tulajdonságú az ún. *diszjunkt dekompozíció*, amelynek legegyszerűbb változata alábbi módon definiálható.

Jelöljük egy n változós logikai függvény változóinak halmazát Q-val. Képezzük a változók halmazának két diszjunkt részhalmazát, Q_1-et és Q_2-t. A függvény minden egyes változóját tartalmazza valamelyik részhalmaz, de egy változó csak egy részhalmazban szerepel, azaz a két részhalmaz egyesítése az összes változó halmazát adja és a két részhalmaz közös része üres halmaz:

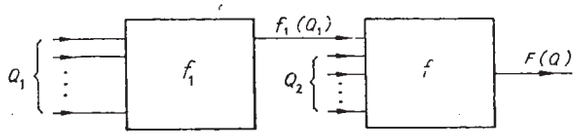
$$Q_1 \cup Q_2 = Q$$

$$Q_1 \cap Q_2 = \emptyset,$$

ahol $\cup$ a halmazok egyesítésének műveletét, $\cap$ a közösrészképzés műveletet, $\emptyset$ az üres halmazt jelöli. Ha az összes változón értelmezett $F(Q)$ logikai függvény kifejezhető az alábbi módon.

$$F(Q) = f[f_1(Q_1), Q_2],$$

akkor az f és f_1 függvényeket az F *egyszerű diszjunkt dekompozíciójának* nevezik. Az ilyen dekompozíciót megvalósító hálózat felépítését a 2.75. ábra szemlélteti. Ebben a felépítésben valóban elegendő az f és f_1 függvényeket megvalósító hálózatokat önmagukban hazárdmentesíteni ahhoz, hogy az egész hálózat hazárdmentes legyen, mert a két részhálózatnak nincs közös bemenőjele. E felépítésben a statikus és dinamikus hazárd szempontjából vizsgálandó szomszédos bemeneti változások mindig csak az egyik részhálózat bemenetén játszódnak le. Ha mindkét részhálózat



2.75. ábra. Az egyszerű diszjunkt dekompozíció felépítésének vázlata

önmagában hazárdmentes, akkor a felépítésből következően az F kimeneten sem keletkezhet hibás átmeneti kimeneti érték. Ha a Q_1 és Q_2 részhalmazok nem volnának diszjunktak, akkor előfordulnának olyan bemeneti jelek, amelyek közvetlenül rájutnának mindkét részhálózat bemenetére. Az ilyen bemeneti jelek megváltozása révén keletkező szomszédos bemeneti változások az f jelű hálózatban funkcionális hazárdot okozhatnának abban az esetben, ha a bemeneti változás hatására az f_1 hálózat kimenete értéket váltana. Ilyenkor ugyanis az f hálózat bemenetén az $f_1(Q_1)$ értékváltozása és a bemeneti változás együttesen nem szomszédos változást okozna, aminek következtében az F kimeneten hibás átmeneti értékek jelenhetnének meg annak ellenére, hogy az f és f_1 hálózatok önmagukban hazárdmentesek. A választott diszjunkt dekompozíció emiatt kedvezőbb a nem diszjunkt, vagyis az ún. *konjunkt dekompozíciónál.* A diszjunkt dekompozíció kedvező hatása természetesen megmarad, ha a felbontást tovább folytatjuk az f és f_1 függvényekre, majd az újabb részfüggvényekre is mindaddig, amíg a keletkező részfüggvények olyan egyszerűek nem lesznek, hogy megvalósításukat az adott építőelem-készlettel közvetlenül elvégezhetjük további felbontás nélkül.

A diszjunkt dekompozíció esetén a Q_1 részhalmazban levő változókat *kötött,* a Q_2-ben levőket pedig *szabad* változóknak nevezik. Az elnevezések arra utalnak, hogy a Q_1 részhalmazban levő változók csak közvetve, a Q_2-ben levők pedig közvetlenül hatnak a kimeneti szinten. Példaként határozzuk meg az alábbi logikai függvény egyszerű diszjunkt dekompozícióját:

$$F(A, B, C, D) = \bar{A}\bar{B}\bar{C}\bar{D}+\bar{A}B\bar{C}D+A\bar{B}CD+ABC\bar{D}.$$

A függvény első két tagjából emeljük ki $\bar{A}\bar{C}$ szorzatot, a második két tagjából pedig az AC szorzatot:

$$F(A, B, C, D) = \bar{A}\bar{C}(\bar{B}\bar{D}+BD)+AC(\bar{B}D+B\bar{D}).$$

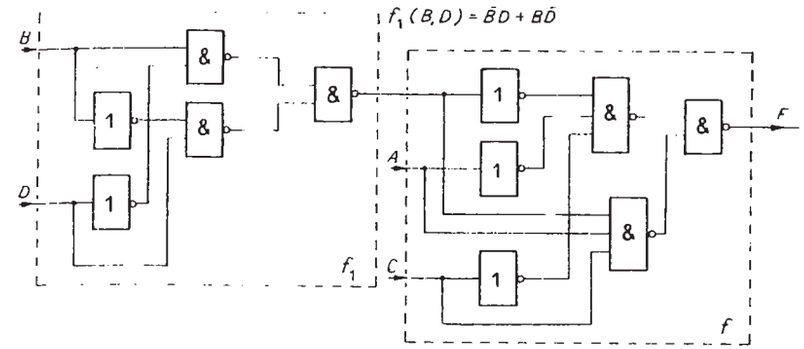
A $\overline{\bar{B}\bar{D}+BD}=B\bar{D}+\bar{B}D$ egyenlőség figyelembevételével a logikai függvény átírható a következő alakra:

$$F(A, B, C, E) = \bar{A}\bar{C}\cdot\overline{\bar{B}D+B\bar{D}}+AC\cdot(\bar{B}D+B\bar{D}).$$

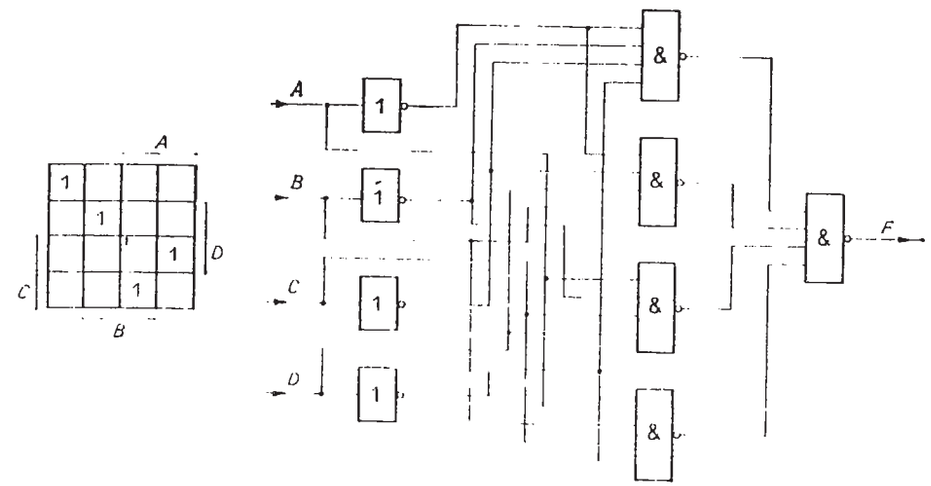
Az $f_1(B, D)=\bar{B}D+B\bar{D}$ függvénykapcsolatot bevezetve:

$$F(A, B, C, D) = \bar{A}\bar{C}\cdot\overline{f_1(B, D)}+AC\cdot f_1(B, D) = f[f_1(B, D), A, C].$$

Ez az előbbiekkel összevetve azt jelenti, hogy a változók Q_1 részhalmazába a B és D kötött változók, Q_2 részhalmazába pedig az A és C szabad változók tartoznak.



2.76. ábra. Az $F=\bar{A}\bar{B}\bar{C}\bar{D}+\bar{A}B\bar{C}D+A\bar{B}CD+ABC\bar{D}$ függvény elvi logikai rajza egyszerű diszjunkt dekompozíció alapján



2.77. ábra. A 2.76. ábrán egyszerű diszjunkt dekompozícióval megvalósított függvény kétszintű elvi logikai rajza

Az így kapott diszjunkt dekompozíció alapján például a 2.76. ábrán látható elvi logikai rajz szerint valósíthatjuk meg a függvényt.

Hasonlítsuk össze eredményünket a kétszintű megvalósítással. A kiindulási függvény Karnaugh-táblája és kétszintű elvi logikai rajza a 2.77. ábrán látható. A kapubemenetek számát tekintve a dekompozíciós megoldás kedvezőbb.

Példánkban az algebrai alakból kiindulva intuitív átalakításokkal határoztuk meg a változók részhalmazait, valamint az f és f_1 függvényeket. Sajnos nem minden logikai függvénynek létezik az egyszerű diszjunkt dekompozíciója, tehát az intuitív átalakítgatások nem mindig vezetnek célra. A továbbiakban bemutatunk egy eljárást, amelynek segítségével eldönthető, hogy adott logikai függvény esetén létezik-e egyszerű

diszjunkt dekompozíció, és ha igen, akkor az eljárás segítségével az meg is határozható. Az eljárás szemléltetése céljából induljunk ki az alábbi teljesen határozott logikai függvényből:

$$F = \sum^{4}(4, 5, 6, 7, 8, 13, 14, 15).$$

Ennek normálalakja

$$F(A, B, C, D) = \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}C\bar{D} + AB\bar{C}D +$$
$$+ ABC\bar{D} + ABCD.$$

Tételezzük fel, hogy a fenti függvénynek létezik egyszerű, diszjunkt dekompozíciója az alábbi változófelosztás mellett:

$$F(A, B, C, D) = f[f_1(A, D), B, C].$$

A fenti egyenlet szerint a független változók csoportosítását $Q_1 = (A, D)$ és $Q_2 = (B, C)$ módon tételeztük fel. Az F logikai függvény biztosan felírható az alábbi alakban is:

$$F(A, B, C, D) = \bar{B}\bar{C} \cdot p(A, D) + \bar{B}C \cdot r(A, D) + BC \cdot s(A, D) + B\bar{C} \cdot t(A, D),$$

ahol p, r, s és t egy-egy logikai függvénykapcsolatra utal. Ezt a függvényalakot egyszerűen úgy képezhetjük, hogy a diszjunktív normálalakban a mintermekből a B és C változók kombinációit rendre kiemeljük. Az adott logikai függvény esetében ezt az átalakítást az alábbiakban követhetjük:

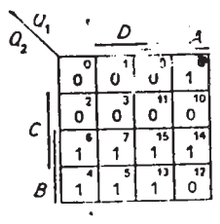
$$F(A, B, C, D) = \bar{B}\bar{C} \cdot (A\bar{D}) + \bar{B}C \cdot (0) + BC \cdot \underbrace{(\bar{A}\bar{D} + \bar{A}D + A\bar{D} + AD)}_{1} +$$
$$+ B\bar{C} \cdot (\bar{A}\bar{D} + \bar{A}D + AD).$$

A felírt algebrai alakból p, r, s és t függvénykapcsolatok kiolvashatók.

Ha p, r, s és t függvényekre nincs további kikötés, feltétel, akkor természetesen minden logikai függvény felírható a fenti alakban.

A további feltételek megfogalmazása céljából ábrázoljuk az F logikai függvényt Karnaugh-táblán. A Karnaugh-tábla kialakítása annyiban térjen el a már megismertektől, hogy egy-egy oldal peremezését a feltételezett Q_1 és Q_2 változócsoportok szerint végezzük (2.78. ábra). A cellákban a mintermeknek megfelelő bináris kombinációk decimális értékét is feltüntettük.

Az F logikai függvény előzőekben felírt alakját, valamint a Karnaugh-tábla sorait összehasonlítva megállapíthatjuk, hogy a $\bar{B}\bar{C} \cdot p(A, D)$ szorzat a Karnaugh-tábla



2.78. ábra. Ábrázolás a Karnaugh-táblán a dekompozíció létezésének vizsgálatához

első sorát, a $\bar{B}C \cdot r(A, D)$ szorzat a második sorát, a $BC \cdot s(A, D)$ szorzat a harmadik sorát, a $B\bar{C} \cdot t(A, D)$ szorzat pedig a negyedik sorát határozza meg.

Az egyszerű diszjunkt dekompozíció létezését feltételezve, az F logikai függvény előbbi felírásában szereplő p, r, s és t függvényekre kikötéseket kell tennünk. Az F logikai függvénynek ugyanis felírhatónak kell lennie a B és C változók, valamint egyetlen $f_1(A, D)$ változó függvényeként.

Ha ilyen f_1 függvény létezését feltételezzük, akkor p, r, s és t bármelyikét tekinthetjük f_1-nek. Ekkor azonban a többire már szigorú feltételeknek kell teljesülniük.

Ha például feltételezzük, hogy $p = f_1$, akkor a többi függvénynek (r, s, t) vagy azonosnak kell lennie p-vel, vagy $\bar{p}$-tal, vagy 0-val, vagy 1-gyel. Csak ebben az esetben írható át F logikai függvény fenti algebrai alakja úgy, hogy csak B, C, valamint f szerepeljenek benne független logikai változóként.

A változók Q_1 és Q_2 felosztása szerint peremezett Karnaugh-táblán az előbbiek azt jelentik, hogy az egyes sorok vagy azonosak, vagy egymás komplemensei, vagy csak 0-t, vagy csak 1-est tartalmaznak. A példánkban szereplő Karnaugh-tábla (2.78. ábra) éppen ilyen. A változók felosztása szerint peremezett táblát a továbbiakban *felosztási táblának* nevezzük.

A vizsgált F logikai függvény tehát a következő módon írható fel:

$$F(A, B, C, D) = \bar{B}\bar{C} \cdot f_1(A, D) + B\bar{C} \cdot \overline{f_1(A, D)} + BC = f[f_1(A, D), B, C],$$

ahol $f_1(A, D) = A\bar{D}$.

Ezek után megfogalmazhatjuk az F logikai függvény egyszerű, diszjunkt dekompozíciója létezésének általános feltételét az adott változófelosztás mellett.

Egy F logikai függvény felosztási táblája akkor és csak akkor felel meg a függvény egyszerű diszjunkt dekompozíciójának, ha a táblán az oszlopfajták száma nem nagyobb kettőnél.

Az állítást az alábbi gondolatmenettel szemléltethetjük. Ha egy logikai függvénynek létezik az egyszerű, diszjunkt dekompozíciója, akkor a felosztási tábla sorai vagy csak nullákat, vagy csak egyeseket, vagy f_1, vagy $\bar{f}_1$ értékeket tartalmazhatnak. A csupa 0-t, ill. 1-et tartalmazó sorok nem növelik az oszlopfajták számát, hiszen minden oszlopban azonosak. Ha egy oszlop 1-et (0-t) tartalmaz egy f_1-nek ($\bar{f}_1$-nak) megfelelő sorban, akkor ugyanez az oszlop 0-t (1-et) tartalmaz az $\bar{f}_1$-nek (f_1-nak) megfelelő sorban. A feltétel teljesülése esetén tehát valóban legfeljebb kétfajta oszlop keletkezhet.

Állításunk alapján lehetőségünk van az egyszerű diszjunkt dekompozíció megkeresésére. Ehhez természetesen az összes lehetséges változófelosztás (Q_1, Q_2 részhalmazokba való sorolás) szerint peremezett Karnaugh-táblák, ill. felosztási táblák mindegyikét meg kell vizsgálni az oszlopfajták száma szempontjából.

A 2.79. ábrán a négyváltozós esetre vonatkozó összes képezhető felosztási tábla látható. A táblák peremezésen olyan elrendeződésben ábrázoltuk, hogy az oszlopfajták számát könnyű legyen ellenőrizni. A táblákon bekarikázva ábrázoltuk azok-

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

a)

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

b)

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

c)

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

d)

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

e)

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

f)

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

g)

2.79. ábra. Az összes lehetséges felosztási tábla négyváltozós esetben

nak a mintermeknek a decimális indexeit, amelyeket a példánkban vizsgált függvény tartalmaz.

A 2.79. ábrán az *a)*, *b)*, *c)* és *d)* esetben a változókat úgy csoportosítottuk, osztottuk fel, hogy 1 db változó szabad, a többi (3 db) kötött változó legyen. Az első négy felosztási tábla tehát a változók 1—3 felosztásának lehetséges eseteit ábrázolja.

A táblázatok segítségével megállapítható, hogy csak a *b)* esetben van kétfajta oszlop, vagyis csak az ennek megfelelő 1—3 változófelosztású egyszerű, diszjunkt dekompozíció létezik.

Az *e)*, *f)* és *g)* esetben a logikai változók 2—2 típusú felosztása vizsgálható. Ennél a felosztásnál a felosztási tábla mindegyike tulajdonképpen két felosztási táblát képvisel, hiszen a szabad és kötött változók szerepének felcserélése ebben az esetben csupán az oszlopok és sorok felcserélését jelenti. Ezért a változók e 2—2 típusú felosztása mellett az oszlopfajták vizsgálatát úgy is el kell végezni, hogy a felosztási tábla sorait tekintjük oszlopoknak. Az *e)* esetben kétfajta oszlopot figyelhetünk meg, a sorokat tekintve oszlopoknak, viszont az oszlopfajták száma nagyobb kettőnél. Így az *e)* felosztási tábla alapján a C és D változók lehetnek kötött változók. Az *f)* felosztási táblán akkor adódik kétfajta oszlop, ha a sorokat tekintjük oszlopoknak, ezért az A és C változókat tekinthetjük kötött változóknak. Hasonló módon állapíthatjuk meg a *g)* felosztási táblán, hogy az A és D változók is lehetnek kötött változók.

Írjuk fel ezek után a példaként vizsgált függvény összes lehetséges egyszerű diszjunkt dekompozícióját. A *b)* tábla alapján tekintsük a $B=0$ sorhoz tartozó bekarikázott mintermek logikai összegét $\bar{B}\cdot f_1(A, C, D)$-nek, így a $B=1$ sorhoz $B\cdot\overline{f_1(A, C, D)}$ tartozik, vagyis a függvény felírható az alábbi alakban:

$$F(A, B, C, D) = \bar{B}\cdot f_1(A, C, D)+B\cdot\overline{f_1(A, C, D)}.$$

A $B=0$ sor alapján:

$$f_1(A, C, D) = A\bar{C}\bar{D}.$$

Az *e)* felosztási tábla alapján C és D tekinthető kötött változónak. Az $AB=00$ sorban nincs olyan minterm, amelyet a függvény tartalmaz, az $AB=01$ sorban pedig az összes mintermet tartalmazza a függvény. Tekintsük az $AB=10$ sorhoz tartozó mintermek logikai összegét $A\bar{B}\cdot f_1(C, D)$-nek. Így az $AB=11$ sorhoz $AB\cdot\overline{f_1(C, D)}$ tartozik. A függvény tehát felírható az alábbi alakban:

$$F(A, B, C, D) = \bar{A}\bar{B}\cdot 0+\bar{A}B\cdot 1+A\bar{B}\cdot f_1(C, D)+AB\cdot\overline{f_1(C, D)} =$$

$$= \bar{A}B+A\bar{B}\cdot f_1(C, D)+AB\cdot\overline{f_1(C, D)},$$

ahol $f_1(C, D) = \bar{C}\bar{D}$.

A függvény dekompozícióját az *f)* és *g)* táblák sorai alapján ugyanolyan gondolatmenettel írhatjuk fel, mint az *e)* tábla oszlopai alapján. Ezért részletezés nélkül csak a végeredményt adjuk meg.

Az *f)* tábla alapján:

$$F(A, B, C, D) = \bar{B}\bar{D}\cdot f_1(A, C)+BD+B\bar{D}\cdot\overline{f_1(A, C)},$$

ahol $f_1(A, C)=A\bar{C}$.

A *g)* tábla alapján:

$$F(A, B, C, D) = \bar{B}\bar{C}\cdot f_1(A, D)+BC+B\bar{C}\cdot\overline{f_1(A, D)},$$

ahol $f_1(A, D)=A\bar{D}$.

A bemutatott egyszerű diszjunkt dekompozíció a függvénynek két részfüggvényre történő felbontását jelenti. Természetesen a részfüggvényekre alkalmazott további egyszerű diszjunkt dekompozícióval további részfüggvényekre bonthatjuk a kiindulási függvényt. Az ilyen hierarchikusan egymásba ágyazott dekompozíciós alakzatokat eredményező eljáráson kívül más módon is elvégezhetjük a több részfüggvényt adó diszjunkt dekompozíciót. Ha a változók halmazát ugyanis kettőnél több diszjunkt részhalmazra osztjuk, akkor a hazárdmentesítéssel kapcsolatos előnyök megmaradnak és közvetlenül kettőnél több részfüggvényt kapunk.

2.5.2. Több szintű kombinációs hálózatok NAND vagy NOR kapukból történő felépítésének egy módszere

Az eljárás szemléltetéséhez vizsgáljuk meg a 2.98. ábrán látható háromszintű elvi logikai rajzot, amely NAND kapukból épül fel. A hálózat által megvalósított logikai függvény algebrai alakja a De Morgan-azonosság alkalmazásával az alábbi módon írható fel:

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

$$= A(\bar{C} + \bar{D}) + BC(\bar{C} + \bar{D}) + AB = A\bar{C} + A\bar{D} + BC\bar{D} + AB.$$

A kiadódott diszjunktív algebrai alak és a NAND kapukból felépülő elvi logikai rajz összevetéséből az alábbi törvényszerűségeket lehet megállapítani:

A $\left.\begin{matrix}\text{páros}\\ \text{páratlan}\end{matrix}\right\}$ szinteken levő kapuk a bemeneteikre jutó változók $\left.\begin{matrix}\text{ponáltjai}\\ \text{negáltjai}\end{matrix}\right\}$ közötti $\left.\begin{matrix}\text{ÉS}\\ \text{VAGY}\end{matrix}\right\}$ kapcsolatokat valósítják meg.

A fenti formális szabály alkalmazásával a 2.98. ábra alapján az alábbi függvényalakot írhatjuk fel:

$$F = A(\bar{C} + \bar{D}) + BC(\bar{C} + \bar{D}) + AB,$$

amely megegyezik a kiindulási átalakítás során kapott második közbenső kifejezéssel.

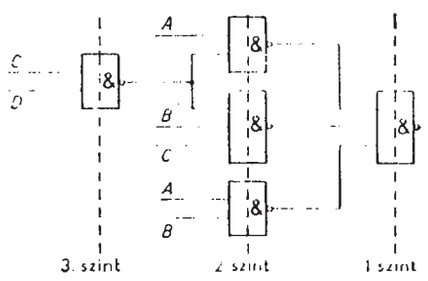
Könnyen belátható, hogy a fenti törvényszerűségek csak olyan több szintű hálózatokban érvényesek, amelyekben az egyes kapukimenetek csak a következő szintre kapcsolódnak, azaz amelyekben nem fordul elő, hogy egy kapukimenet több szintre kapcsolódik.

A NAND és NOR műveletek közötti dualitás következtében a fenti törvényszerűségek könnyen átfogalmazhatók NOR kapukból felépülő több szintű hálózatokra is. Az eljárás lényegének megértését ezért nem zavarja, ha a továbbiakban a NAND kapukból felépített hálózatokra szorítkozunk.

Példaként valósítsuk meg NAND kapukból felépülő több szintű hálózattal az

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

függvénnyel megadható KIZÁRÓ VAGY kapcsolatot.



2.98. ábra. Példa több szintű hálózat NAND kapukból történő felépítésének szemléltetésére

A függvény algebrai alakját az alábbi módon átalakítva a műveleteket hozzárendelhetjük az egyes szintekhez a fenti törvényszerűségek alapján:

$$F(A, B) = A \cdot (\bar{A} + \bar{B}) + B \cdot (\bar{A} + \bar{B}).$$

1. szint

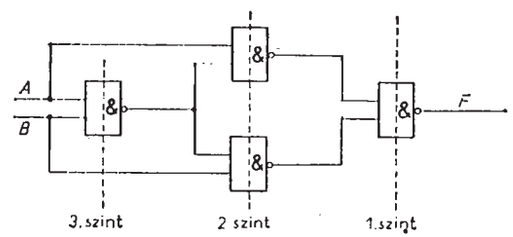
2. szint 2. szint

3. szint 3. szint

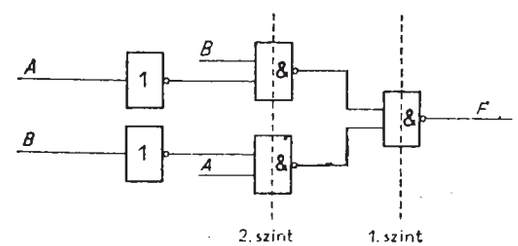
Az egyes változóknak megfelelő bemenőjelek bevezetésének helyét is meghatározhatjuk. Az így felépíthető elvi logikai rajz a 2.99. ábrán szerepel. Összehasonlítás céljából a 2.100. ábrán felrajzoltuk a kétszintű megoldást szintén NAND kapukból felépítve. Ha a szükséges INVERTER-eket is figyelembe vesszük, akkor a több szintű megvalósítás kevesebb építőelemet igényel.

A bemutatott egyszerű példa alapján is megállapítható, hogy a több szintű megvalósításhoz kedvező algebrai alak felírása alapvetően próbálgatásos jellegű feladat. A bemutatott törvényszerűségek nyújtanak ugyan ehhez segítséget, de sok változó esetén az eljárás rendkívül körülményessé válik. A hálózat felépítésére tett megkötés, amely a szintek egymás utáni kapcsolódására vonatkozik, kedvező a szintenkénti hazárdmentesítés szempontjából. Egy adott szinten levő kapuk kimenetét ugyanis csak egy másik szinten kell figyelembe venni.

Több kimenetű hálózatok tervezésekor az eljárás a kimenetenkénti algebrai alakokból indul ki. Az egyes kimeneti függvények megvalósításakor keletkező esetlegesen azonos felépítésű és bemenetű szinteket természetesen elegendő csak egyszer megvalósítani.



2.99. ábra. Példa NAND kapukból felépülő több szintű kombinációs hálózatra



2.100. ábra. A 2.99. ábrán több szintű módon megvalósított függvény összehasonlítás céljából kialakított kétszintű logikai rajza

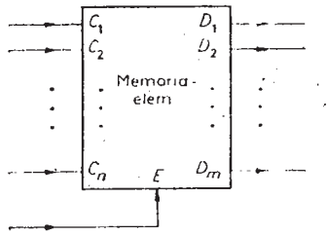
2.6. Kombinációs hálózatok megvalósítása memóriaelemek felhasználásával

A digitális berendezések memóriáit arra a célra alakították ki, hogy bináris információkat tároljanak. A tárolt bináris információk szükség esetén kiolvashatók a memóriából, azaz *hozzáférhetők*. A memóriaelemek fizikai felépítésével, áramköri megoldásaival, különböző típusaikkal és tulajdonságaikkal e helyen nem foglalkozunk, csupán működésük alapgondolatát foglaljuk össze. Egy adat a memóriában akkor hozzáférhető, ha tudjuk, hogy a memória melyik *helyén* található. Ahhoz tehát, hogy egy adatot a memóriából ki lehessen olvasni, tudni kell, hogy a szóban levő adat melyik helyen található, ezt a helyet egy ún. *címmel* azonosítjuk.

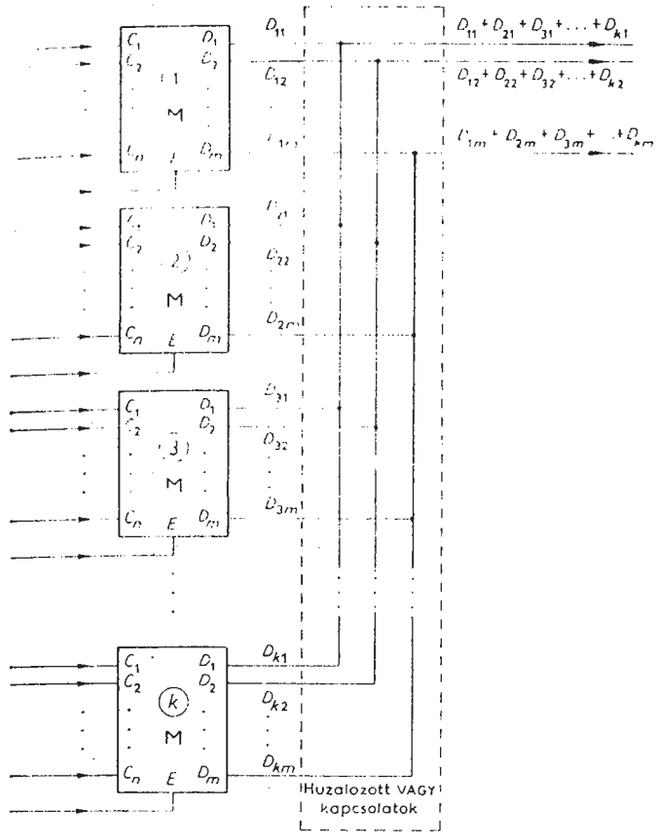
Ha tehát egy adatot a memóriában elhelyezünk, akkor az adat mellett azt a címet is közölni kell, amelyen az adatot el kell helyezni. Kiolvasáskor ugyanezt a címet kell közölnünk a memóriával ahhoz, hogy az illető adatot megkapjuk. Mind az adat, mind a cím egy-egy bináris kombináció, azaz bináris szám.

Fogalmazzuk meg ezek után a 2.101. ábrán levő általános vázlat alapján a memóriaelem alapvető működését kiolvasáskor:

A $C_1 \ldots C_n$ *címbemenetekre* érkező bináris kombináció, azaz a cím hatására a memóriaelem kiválasztja az általa tárolt m bites bináris kombinációk, az ún. *adatok* közül azt, amelynek helyét a cím jelöli ki. Az így kiválasztott adat a $D_1 \ldots D_m$ *adatkimeneteken* jelenik meg. A cím megérkezése és az általa kijelölt adat megjelenése között eltelt időt a memóriaelem *ciklusidejének* vagy *elérési idejének* nevezik. Az adatkimenetek áramköri megvalósítása általában lehetővé teszi, hogy több memóriaelem megfelelő adatkimeneteit közvetlenül összekössük és ezáltal ún. *huzalozott VAGY kapcsolatot* alakítsunk ki a megfelelő adatkimenetek között a 2.102. ábra szerint. Ennek a lehetőségnek a hasznosításához járul hozzá az E jelű *engedélyezőbemenet* funkciója. Az E jelű bemenetre juttatható jel két lehetséges logikai értéke közül az egyik — az engedélyező — azt eredményezi, hogy az illető memóriaelem adatkimeneti jelei jutnak a huzalozott VAGY kapcsolatok kimeneteire. Az E bemenetre adható másik logikai jelérték — az ún. tiltó — viszont megakadályozza, azaz letiltja az adatkimeneti jelek megjelenését és a memóriaelem adatkimenetei rendszerint olyan áramköri állapotba kerülnek, hogy nem zavarják a huzalozott VAGY kapcsolatok



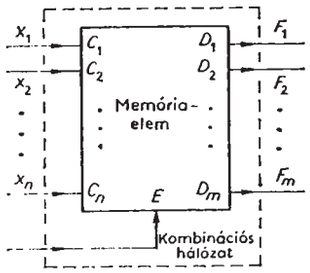
2.101. ábra. A memóriaelem általános értelmezése kombinációs hálózatok megvalósítása céljából



2.102. ábra. Memóriaelemek megfelelő adatkimeneteinek összekötésével megvalósítható huzalozott VAGY kapcsolatok szemléltetése

megvalósulását, vagyis a többi memóriaelem valamelyikének adatkimeneti jelei juthatnak a huzalozott VAGY kapcsolatok kimeneteire. Ha tehát a huzalozott VAGY kapcsolatban részt vevő memóriaelemek közül mindig csak egynek az E bemenetére adunk engedélyező jelértéket a többiére pedig tiltót, akkor ezáltal egyszerűen kiválaszthatjuk, hogy melyik memóriaelemnek az adatkimeneti jelei jussanak a huzalozott VAGY kapcsolatok kimeneteire.

A vázolt tulajdonságok és működési mód alapján a memóriaelem alkalmas kombinációs hálózatok megvalósítására. Ennek szemléltetése céljából tételezzük fel, hogy adott egy n bemenetű m kimenetű kombinációs hálózat igazságtáblája. Ennek ismert értelmezése szerint a 2^n számú bemeneti kombináció mindegyike kijelöl egyet az adott kimeneti kombinációk közül. Ennek a kimeneti kombinációnak kell megjelennie a kombinációs hálózat kimenetén az őt kijelölő bemeneti kombináció fellépésekor. Könnyen belátható tehát, hogy a memóriaelem megvalósíthatja a kombinációs hálózatot, ha a 2.103. ábra szerint a címbenemetekre juttatjuk a bemeneti jeleket és az adatkimeneteket tekintjük kimeneteknek. Ha ugyanis feltételezzük, hogy



2.103. ábra. A memóriaelem alkalmazása több kimenetű kombinációs hálózat megvalósítására

a memóriaelem által tárolt adatok minden egyes bemeneti kombinációnak megfelelő címen azonosak az illető bemeneti kombinációhoz tartozó kimeneti kombinációval, akkor a címek hatására megjelenő adatok éppen a megvalósítandó hálózat előírt működésének felelnek meg. A kombinációs hálózat működése tehát ily módon a memóriaelemből történő adatkiolvasási lépésekben valósul meg, miután az igazságtábla alapján a megfelelő adatokat bejuttattuk a memóriaelembe. Az adatokat a különböző memóriatípusok esetében más-más módon juttatják, ill. írják be, ezúttal csak azt tételezzük fel, hogy az adatok valamilyen módon a memóriába juttathatók.

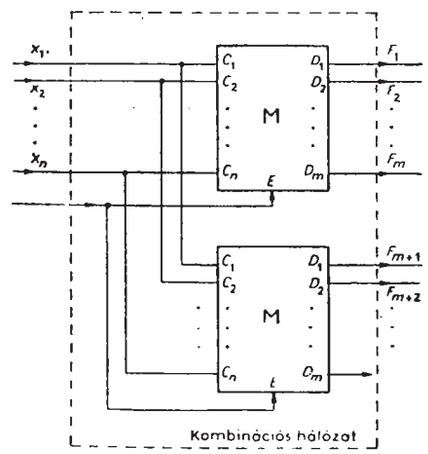
A memóriaelem bemutatott alkalmazásának az a fő előnye, hogy közvetlenül az igazságtábla alapján végezhetjük el az adatbeírást és így nem szükséges semmiféle minimalizálás. Az elvi logikai rajz és a logikai kapcsolási terv rendkívül egyszerű, kevés összeköttetést tartalmaz, így olcsón megvalósítható. Nagyon előnyös az is, hogy egy memóriaelem alkalmas sokfajta kombinációs hálózat megvalósítására attól függően, hogy milyen adatokkal töltjük fel az egyes címeken és így a megvalósított kombinációs hálózat megváltoztatása nem igényli az összeköttetések megváltoztatását.

Memóriaelemek alkalmazása esetén a statikus és dinamikus hazárd nem értelmezhető a megismert módon, hiszen a hálózatot nem a építőelem-készletben található kapukból építjük fel, hanem az igazságtábla alapján közvetlenül határozzuk meg beírandó adatokat. Így az ismert hazárdmentesítési eljárást nem alkalmazhatjuk. A kimenetek tranziens viselkedése az adott memóriaelem tulajdonságaitól függ. A memóriaelem funkcióit tokon belül típusonként más-más jellegű logikai hálózatokkal valósítják meg. A memóriaelemen belüli hazárdmentesítés módjáról és hatékonyságáról a katalógusok alapján általában nem szerezhetünk kellő mélységű ismereteket. A gyártó cégek legtöbbször csak azt garantálják, hogy a címváltozás időpontjától számított ciklusidő múlva az adat rendelkezésre áll a kimeneten. A ciklusidőn belül lezajló kimeneti változások káros hatását ezért általában valamilyen szinkronizációs vagy vezérlési megoldással kell kiküszöbölnünk. A nem szomszédos bemeneti (azaz cím-)változások hatása a kimeneti (azaz adat-)változásokra funkcionális hazárdként értelmezhető. A szinkronizáláshoz, a kimeneti változások ütemezésére előnyösen használható az E bemenet engedélyező, ill. letiltó hatása, hiszen az E bemenetre jutó jellel közvetlenül kapuzhatók a D kimenetek.

A nem szomszédos kimeneti változások memóriaelem alkalmazása esetén ugyan-

úgy funkcionális hazárdot okozhatnak a logikai rendszer érintett egységeiben, mint a kapukból felépített több kimenetű hálózatok esetében.

Az építőelem-készletekben rendelkezésre álló memóriaelemek címeinek száma és az azokon elhelyezhető adatok mérete (bináris jegyeinek száma) rögzített. Ez nyilvánvalóan korlátozza az egyetlen memóriaelemmel megvalósítható kombinációs hálózatok bemeneteinek és kimeneteinek számát. Ha a megvalósítandó hálózat kimeneteinek száma kisebb, mint a rendelkezésre álló memóriaelem adatkimeneteinek száma, akkor a felhasználatlan adatkimenetek nem igényelnek további figyelmet. Ha a megvalósítandó hálózat bemeneteinek száma kisebb, mint a memóriaelem címbemeneteinek száma, akkor a felhasználatlan címbemenetek állandó logikai értékűnek tekintendők. A legtöbb memóriaelem a szabadon hagyott címbemeneteknek eleve állandó értéket tulajdonít, zavarvédelmi okokból azonban ezeket általában huzalozással is logikai 1-re vagy 0-ra kapcsolják. Akár a bemenetek száma, akár a kimenetek száma kisebb a memóriaelem címbemeneteinek, ill. adatkimeneteinek számánál, tulajdonképpen nem használjuk ki a kombinációs hálózat megvalósításakor a memóriaelem teljes tárolási képességét, az ún. *kapacitását*. A felhasználatlan adatkimenetek ugyanis azt jelentik, hogy a tárolt m bináris számjegyű adatok egyes jegyeit nem használjuk, felhasználatlan címbemenetek esetén pedig a memóriaelem bizonyos címei feleslegesek. Vizsgáljuk meg ezek után, hogy miként lehet több memóriaelem összekapcsolásával megvalósítani olyan kombinációs hálózatokat, amelyeknek több bemenetük vagy kimenetük van, mint a rendelkezésre álló memóriaelemeknek. Tételezzük fel először, hogy a megvalósítandó kimenetek száma 2-vel nagyobb, mint a rendelkezésre álló memóriaelem adatkimeneteinek száma. A tárolandó bináris kombinációk tehát több bitből állnak, mint amennyi egyetlen memóriaelemben tárolható adatok bináris jegyeinek száma. A 2.104. ábra szerinti kapcsolásban megfigyelhetjük, hogy ilyenkor két memóriaelem elegendő, ha $m>2$. A bemeneti jelek mindkét memóriaelem címbemeneteire rájutnak, és a kimeneti kombináció pótlólagos két bitjét a második elem két adatkimenete szolgáltatja. Ha a meg-

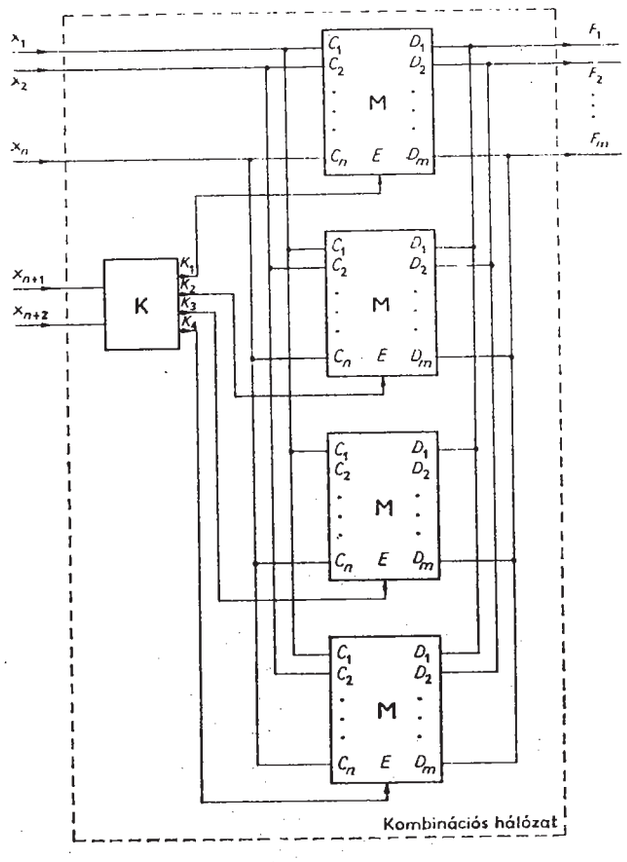


2.104. ábra. Kombinációs hálózat megvalósítása memóriaelemek összekapcsolásával, ha a kimenetek száma miatt egyetlen memóriaelem nem elegendő

valósítandó kimenetek száma meghaladja a két memóriaelem által összesen biztosított kimenetek számát, akkor a kapcsolás hasonló elven bővíthető több memóriaelemmel.

Ha több bemenetet kell megvalósítani, mint amennyi címbemenettel egy memóriaelem rendelkezik, akkor több bináris kombinációt kell tárolni, mint amennyi egyetlen memóriaelemben lehetséges. Tételezzük fel, hogy a megvalósítandó bemenetek száma kettővel nagyobb, mint a rendelkezésre álló memóriaelem címbemeneteinek száma. Ebben az esetben tehát 2^n számú bináris kombináció helyett 2^{n+2} számú kell tárolni, amihez $2^{n+2}=2^n \cdot 2^2$ miatt 4 db memóriaelem szükséges. Általában is megállapíthatjuk, hogy a bemenetek számának növelése 2 egész kitevőjű hatvány szerint növeli a szükséges memóriaelemek számát. A 2.105. ábrán szereplő kapcsolás $n+2$ számú bemenet esetén mutatja be a megoldást. Az x_{n+1} és x_{n+2} bemeneti jelek a K jelű, ún. *átkódoló kombinációs* hálózatra jutnak, amelynek igazságtáblája a 2.106. ábrán látható.

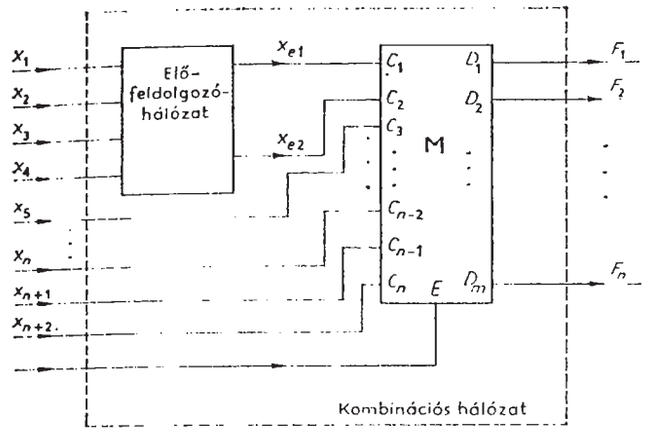
A $K_1 \ldots K_4$ kimeneteken ún. *„n-ből 1 kód"* jelenik meg, vagyis egyidejűleg egynél több kimeneten nem lehet „1". A $K_1 \ldots K_4$ kimenetek vezérlik a memóriaelemek



2.105. ábra. Kombinációs hálózat megvalósítása memóriaelemek összekapcsolásával, ha a bemenetek száma miatt egyetlen memóriaelem nem elegendő

x_{n+1}	x_{n+2}	K_1	K_2	K_3	K_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

2.106. ábra. Példa az engedélyező bemeneteket vezérlő kombinációs hálózat igazságtáblájára. (A 2.105. ábra K jelű átkódolóhálózata)



2.107. ábra. A memóriaelemre jutó bemeneti jelek számának csökkentése előfeldolgozással

jelű bemenetei. Feltételeztük, hogy $E=1$ esetén a memóriaelem működése engedélyezve van, vagyis az adatkimenetek huzalozott VAGY kapcsolatán keresztül az $F_1 \ldots F_m$ kimenetek értéke az $x_1 \ldots x_n$ bemeneteken fennálló bináris kombinációk hatására jön létre. A beírandó adatok összességét tehát fel kell osztani a négy memóriaelem között az x_{n+1}, x_{n+2} bemenetek értékkombinációja szerint. A K jelű hálózat a funkciójából következően szomszédos bemeneti változások hatására nem szomszédos kimeneti változásokat állít elő, amit a teljes kapcsolásban fellépő funkcionális hazárdjelenségek vizsgálatakor feltétlenül figyelembe kell vennünk. Mielőtt azonban a fentiekben leírt módon több memóriaelemet alkalmaznánk, célszerű megvizsgálni a megvalósítandó kombinációs hálózat igazságtábláját. Ha ugyanis az több bemeneti kombinációhoz azonos kimeneti kombinációt rendel hozzá, akkor előfordulhat, hogy a 2.107...2.109. ábrák szerinti ún. *előfeldolgozó* hálózatot alkalmazhatjuk. Ennek kevesebb kimenete van, mint ahány bemenete. A 2.107. ábrán feltételeztük, hogy az $x_1 \ldots x_4$ bemeneti jelek alapján kombinációs hálózattal előállíthatók az x_{e1} és x_{e2} előfeldolgozott bemeneti jelek, amelyek az $x_5 \ldots x_{n+2}$ bemeneti jelekkel együtt már csak egyetlen, n számú címbemenetű memóriaelemet igényelnek a kiindulási kombinációs hálózat megvalósítása céljából. Az eljárás alkalmazhatósága természetesen feladatfüggő és ezért általános érvényű szisztematikus módszer nem létezik sem az előfeldolgozásban résztvevő jelek kiválasztására, sem pedig az előfeldolgozó hálózat igazságtáblájának meghatározására. Az is előfordulhat, hogy a memória-

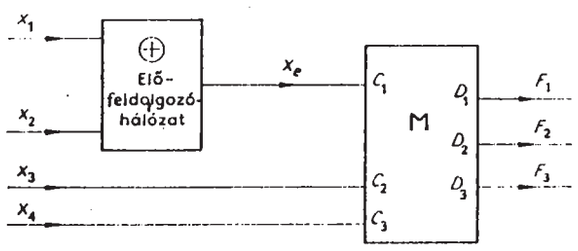
elem címbemeneteire jutó jeleknek vagy azok egy részének részt kell venniük az előfeldolgozásban is. Ilyen esetekben fokozottan kell ügyelnünk a funkcionális hazárd esetleges következményeire. A 2.108. és 2.109. ábrán egyszerű példákon szemléltettük az eljárás lényegét. A könnyebb áttekinthetőség céljából azt feltételeztük, hogy a memóriaelemnek három címbemenete van és négybemenetű kombinációs hálózatot kell megvalósítanunk. Figyeljük meg, hogy a 2.108. ábrán azért választhattuk le könnyen az előfeldolgozó hálózatot, mert egyrészt a megvalósítandó hálózat igazságtáblájában az $F_1F_2F_3=000$ kimeneti kombináció olyan bemeneti kombinációkhoz van hozzárendelve, amelyek $x_1 \oplus x_2=1$-gyel jellemezhetők az x_3 és x_4 értékének megadása nélkül. Másrészt pedig a többi előforduló kimeneti kombináció fellépését

x_1	x_2	x_3	x_4	F_1	F_2	F_3
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	1
0	0	1	1	1	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	0	1	0	1	0
1	1	1	0	0	1	1
1	1	1	1	1	0	0

a)

x_e	x_3	x_4	F_1	F_2	F_3
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

b)



c)

2.108. ábra. Egyszerű példa az előfeldolgozás alkalmazására

a) A megvalósítandó hálózat igazság táblája

b) A memóriaelem (M) által megvalósítandó függvény igazságtáblája

c) Kapcsolási vázlat. Az előfeldolgozó hálózat által megvalósítandó logikai függvény:

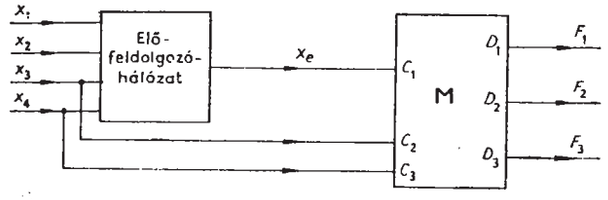
$x_e = x_1 \oplus x_2$

x_1	x_2	x_3	x_4	F_1	F_2	F_3
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	1	0	0

a)

x_e	x_3	x_4	F_1	F_2	F_3
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

b)



c).

2.109. ábra. Egyszerű példa az előfeldolgozás alkalmazására. Az előfeldolgozásban részt vevő bemeneti jelek (x_3, x_4) a memóriaelem címbemenetére (c_2, c_3) közvetlenül is rájutnak

a) A megvalósítandó hálózat igazság táblája

b) A memóriaelem (M) által megvalósítandó hálózat igazság táblája

c) Kapcsolási vázlat. Az előfeldolgozó hálózat által megvalósítandó logikai függvény:

$$x_e = [x_1 \oplus x_2 + E(x_1, x_2, x_3, x_4)] \cdot \overline{H(x_1, x_2, x_3, x_4)},$$

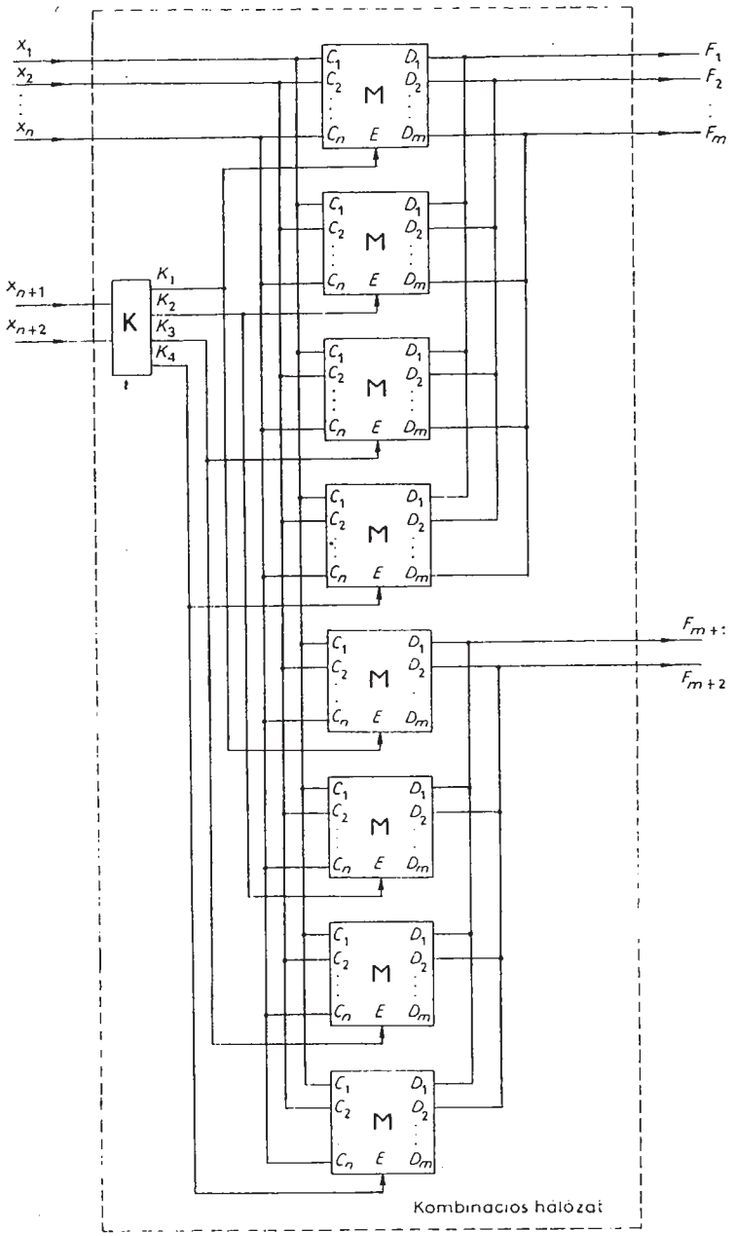
ahol

$$E(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2x_4 + \bar{x}_2x_3x_4,$$

$$H(x_1, x_2, x_3, x_4) = x_1x_3.$$

olyan bemeneti kombinációk okozzák, amelyek $x_1 \oplus x_2=0$ értéket állítanak elő, és ezen felül csak x_3 és x_4 értékének ismerete szükséges az egyes kimeneti kombinációk megkülönböztetéséhez.

A 2.109. ábrán bemutatott esetben is hasonló gondolatmenetet követhetünk, de az előfeldolgozó hálózat által megvalósítandó logikai függvény felismerése nehezebb, mert az x_3 és x_4 bemeneti jeleknek részt kell venniük az előfeldolgozásban is és a memóriaelemre közvetlenül is rá kell jutniuk. Az előfeldolgozó hálózat kimeneti függvényét (x_e) ezúttal is az $F_1F_2F_3=000$ kimeneti kombináció megkülönböztetése alapján írhatjuk fel. Most azonban $x_1 \oplus x_2$ értéke nem elegendő ehhez, így x_e kifejezését az elhagyott és a hozzávett mintermek figyelembevételével kell felírnunk. Az előfeldolgozás alkalmazásával tulajdonképpen egyfajta dekompozíciót hajtunk végre. Ennek során a memóriaelem csak az egyik részhálózatot állítja elő. Termé-



2.110. ábra. Kombinációs hálózat megvalósítása memóriaelemek összekapcsolásával, ha a bemenetek és a kimenetek száma miatt egyetlen memóriaelem nem elegendő

szetesen nincs akadálya annak, hogy az előfeldolgozó hálózatot vagy akár a K jelű hálózatot is memóriaelemekkel valósítsuk meg. Ilyenkor a sorba kapcsolt memóriaelemek miatt csökken az eredő hálózat működési sebessége. Ha a megvalósítandó kombinációs hálózat mind a bemenetek, mind a kimenetek száma szempontjából túlépi a rendelkezésre álló memóriaelem címbemeneteinek és adatkimeneteinek számát, akkor a 2.104. és 2.105. ábrákon bemutatott kapcsolási módokat együttesen alkalmazhatjuk. Ezt szemlélteti a 2.110. ábra $n+2$ számú bemenet és $m+2$ számú kimenet esetére.

A memóriaelemekkel megvalósított kombinációs hálózatok működési sebességét az alkalmazott memóriaelemek ciklusideje szabja meg.

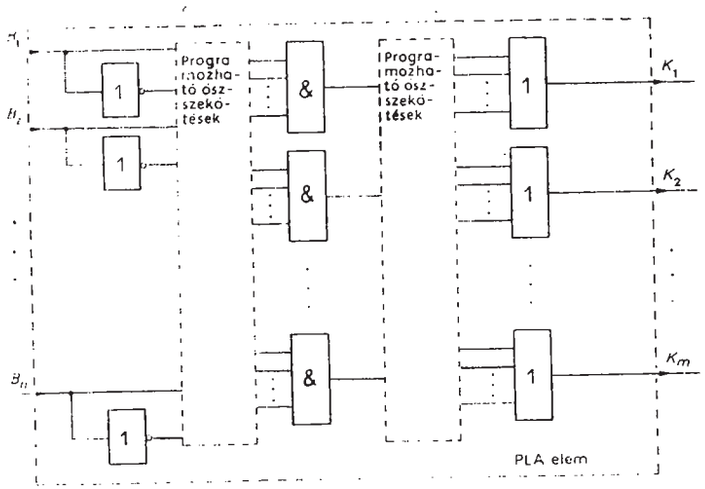
A memóriaelemekkel történő megvalósításhoz alkalmazhatjuk az ismert diszjunkt dekompozíciós eljárást. Ilyenkor a kimenetenként keletkező részfüggvényeket építhetjük fel memóriaelemekkel. A kimenetenkénti felbontást addig érdemes folytatni, amíg a keletkező részfüggvények mindegyike egyetlen memóriaelemmel meg nem valósítható. Ilyenkor a kimenetenként keletkező részfüggvényeket építhetjük fel memóriaelemekkel. Természetesen célszerű arra törekedni, hogy egy memóriaelemmel több kimeneti függvény részfüggvényeit valósíthassuk meg. Ez a törekvés rendszerint visszahat a változók felosztására is, és így alapvetően próbálgatásos, intuitív jellegű feladat. A PLA elemekkel kapcsolatban a dekompozíció keresésének egy további lehetőségét ismerjük meg, amely memóriaelemek esetén is alkalmazható.

2.7. Kombinációs hálózatok megvalósítása PLA elemek felhasználásával

Az ún. *PLA* (Programmable Logic Array), azaz *programozható logikai elemek* típusait és tulajdonságait nem ismertetjük részletesen. Csak az a célunk, hogy bemutassuk felépítésük és felhasználásuk alapgondolatát.

A több kimenetű kombinációs hálózatok minimalizálási módszereinek tárgyalásakor a 2.42. ábrán bemutattuk a kétszintű ÉS—VAGY elvi logikai rajz általános felépítését. Ennek alapján adhatjuk meg legegyszerűbben a PLA elem belső felépítését. A 2.111. ábrán vázolt belső felépítés szerint a PLA elemben adott számú bemenetű adott számú ÉS kapu, ill. VAGY kapu és adott számú INVERTER van megvalósítva. Ezeknek a belső elemeknek az összekötését a kétszintű, több kimenetű kombinációs hálózatok általános belső felépítésének megfelelően a megvalósítandó hálózat kétszintű elvi logikai rajzának ismeretében hozhatjuk létre. Az összeköttetések kialakításának, esetleges változtatásának, azaz programozásának különböző módjaival nem foglalkozunk.

Ha feltételezzük az összekötések programozhatóságát, akkor a PLA elem használatakor a megvalósítandó kombinációs hálózat kétszintű elvi logikai rajzával



2.111. ábra. PLA elem belső felépítésének általános vázlata

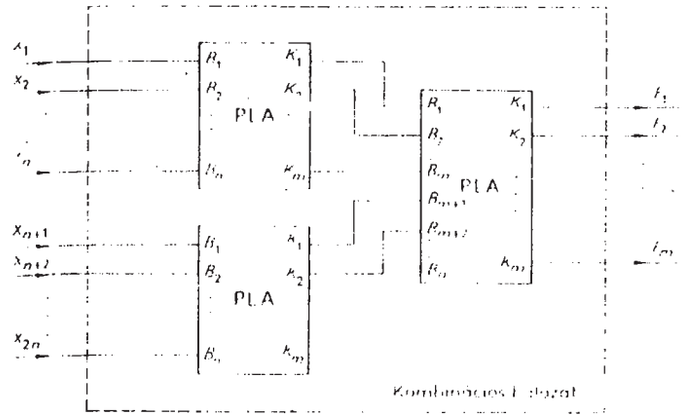
egyértelműen megadhatjuk az összekötések programját, valamint azt, hogy az egyes bemeneti jelek mely PLA bemenetekre jussanak. A kiindulási elvi logikai rajz létrehozásához természetesen alkalmazhatjuk a megismert minimalizáló és hazárdmentesítő eljárásokat.

Az építőelem-készletekben rendelkezésre álló PLA elemek jellemző adatai szabják meg, hogy egy adott kombinációs hálózat megvalósításához elegendő-e egyetlen PLA elem felhasználása. Ezek a jellemzők az alábbiak:

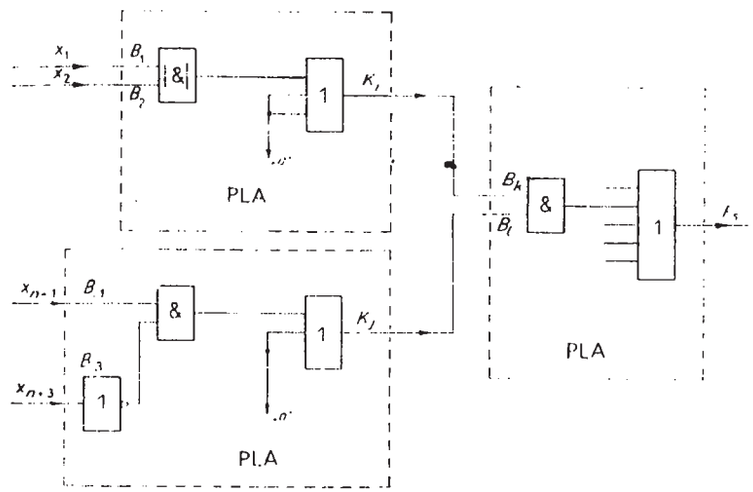
— a PLA elem bemeneteinek száma,
— a PLA elem kimeneteinek száma,
— a belső ÉS kapuk száma.

A belső ÉS kapuk mindegyikének általában annyi bemenete van, amennyi a PLA elem bemeneteinek száma. A belső VAGY kapuk általában annyi bemenetűek, amennyi a belső ÉS kapuk száma. Vannak olyan PLA elemek is, amelyek nem tartalmazzák a bemeneti INVERTER-eket, ezért a megvalósítandó hálózat elvi logikai rajzán ponáltan is és negáltan is szereplő bemeneti jelek mindegyike két PLA bemenetet vesz igénybe, hiszen az invertálást a PLA elemen kívül kell megvalósítani. Ha a megvalósítandó több kimenetű kombinációs hálózat kimeneti függvényeinek minimalizálása és hazárdmentesítése során olyan elvi logikai rajznoz jutottunk, amelyről megállapítható, hogy egyetlen PLA elemmel megvalósítható, akkor a további minimalizálás nyilvánvalóan felesleges és elvégezhetjük a programozást.

Ha a megvalósítandó hálózat elvi logikai rajza nem minimalizálható tovább és egyetlen PLA elemmel nem valósítható meg, akkor több PLA elem megfelelő összekapcsolásával építhetjük fel a hálózatot. A PLA elemek összekapcsolásának módja attól függ, hogy melyik PLA jellemző miatt nem valósítható meg a hálózat egyetlen PLA elemmel.



2.112. ábra. Kombinációs hálózat megvalósítása PLA elemek összekapcsolásával, ha csak a bemenetek száma miatt nem elegendő egyetlen PLA elem



2.113. ábra. Az $x_1 x_2 x_{n+1} \overline{x_{n+3}}$ szorzat előállításának szemléltetése a 2.112. ábra szerinti kapcsolás esetén

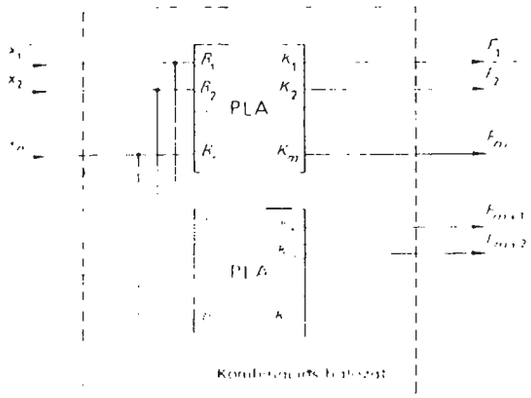
Ha csupán a PLA elem bemeneteinek száma nem elegendő, akkor a 2.112., 2.113. ábra szerinti kapcsolást alakíthatjuk ki. Az ábrán feltételeztük, hogy a megvalósítandó hálózatnak kétszer annyi bemenete van, mint amennyi egyetlen PLA elemnek. A bemeneti jeleket fogadó két PLA elem kimeneti jeleiből újabb PLA segítségével állíthatjuk elő a megvalósítandó hálózat kimeneti jeleit. A feladattól függően ugyanis előfordulhat, hogy olyan ÉS kaput kell megvalósítani, amelynek bemeneti jeleit különböző PLA elemekhez rendeltük. A 2.113. ábrán példaképpen az $x_1 x_2 x_{n+1} \overline{x_{n+3}}$

ÉS kapcsolat előállítását mutattuk be. Látható, hogy a harmadik PLA elem azért szükséges, hogy az ÉS kapcsolat két részéből előállítsuk a teljes megvalósítandó műveletet és VAGY kapcsolatba hozzuk a megfelelő kimenetet előállító többi ÉS kapu kimenettel. (Az i, j, k, l, s indexeknek csak megkülönböztető szerepük van.) Így a különböző bemeneti PLA elemekben megvalósított ÉS kapu kimeneteket is programozható módon hozhatjuk VAGY kapcsolatba. A tervezéskor nyilvánvalóan célszerű arra törekedni, hogy a bemeneti jeleknek a PLA-khoz történő megfelelő felosztású hozzávezetésével minden megvalósítandó ÉS kapu összes bemeneti jelei egyazon PLA-n rendelkezésre álljanak. Sok esetben ezt úgy is elérhetjük, hogy bizonyos bemeneti jeleket több PLA elemre vezetünk, ha azoknak maradnak kihasználatlan bemeneteik. A 2.112. ábrán feltételeztük, hogy erre nem volt lehetőségünk. Nyilvánvaló, hogy nem kell rávezetnünk a kimeneti PLA-ra azokat a kimeneti jeleket, amelyek előállításához csak egyetlen PLA elemen belüli ÉS kapuk szükségesek. A 2.112. ábrán az általános jelölések miatt vezettük rá az összes kimeneti jelet a kimeneti PLA elemre. Az ábrán így azt is feltételeztük, hogy a két bemeneti PLA elem felhasználandó kimeneteinek együttes száma kisebb, mint a kimeneti PLA elem bemeneteinek száma. Az alsó bemeneti PLA elemnek csak az első két kimenetét használtuk fel, miáltal a kimeneti PLA elemre $m+2$ számú jelet kellett vezetnünk és feltételeztük, hogy $n \geq m+2$. Ha adott esetben feltételezéseink nem teljesülnek, akkor a kapcsolás hasonló elven bővíthető és a működés szintén egyszerűen magyarázható a PLA elem belső felépítése alapján. A memóriaelemekkel kapcsolatban bemutatott előfeldolgozás elve szintén alkalmazható, ha a rendelkezésre álló PLA elemnek kevés bemenete van az adott kombinációs hálózat megvalósításához. Sok esetben előfeldolgozó hálózatként is célszerű pótlólagos PLA elemet alkalmazni.

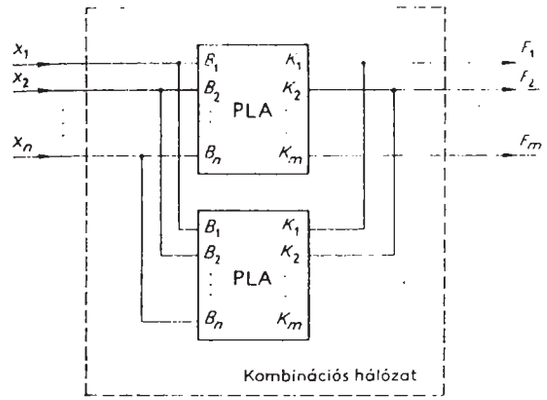
Megfigyelhetjük, hogy a 2.112. ábra szerinti elrendezésben a PLA elemek helyére memóriaelemeket is képzelhetünk. Így a bemutatott gondolatmenettel memóriaelemekkel történő megvalósításkor is megkísérelhetjük a bemeneti jelek felosztását és a dekompozíciót, ha a bemenetek száma miatt nem elegendő egyetlen memóriaelem. Ezáltal sok esetben elkerülhetjük a szükséges memóriaelemek számának a 2.105. ábra szerinti exponenciális növekedését.

Ha egy adott kombinációs hálózat csak azért nem valósítható meg egyetlen PLA elemmel, mert annak nincs elegendő kimenete, akkor a 2.114. ábra szerint kapcsolhatunk össze több PLA elemet. Az ábrán feltételeztük, hogy a megvalósítandó hálózat kettővel több kimenetet igényel, mint amennyivel egyetlen PLA elem rendelkezik. Így az F_{m+1} és F_{m+2} kimeneteket pótlólagos PLA elemmel kell előállítani, miáltal előfordulhat, hogy bizonyos közös prímimplikánsokat mindkét PLA elemben meg kell valósítanunk.

Előfordulhat, hogy a PLA elemnek elegendő bemenete és kimenete van egy adott hálózat megvalósításához, de a belső ÉS kapuk száma nem elegendő ahhoz, hogy az elvi logikai rajz összes ÉS kapuját megvalósítsuk. Ilyen esetben a 2.115. ábra szerinti kapcsolásban alkalmazhatunk több PLA elemet. Az ábrán feltételeztük, hogy csak az F_1 és F_2 kimeneti függvények megvalósításához szükségesek pótlólagos ÉS kapuk,



2.114. ábra. Kombinációs hálózat megvalósítása PLA elemek összekapcsolásával, ha csak a kimenetek száma miatt nem elegendő egyetlen PLA elem



2.115. ábra. Kombinációs hálózat megvalósítása PLA elemek összekapcsolásával, ha csak a belső ÉS kapuk száma miatt nem elegendő egyetlen PLA elem

amelyeket például az alsó PLA elem K_1 és K_2 kimenetén keresztül vehetünk figyelembe. Mivel a pótlólagos ÉS kapuk kimeneteit a megfelelő kimeneti függvények megvalósításakor VAGY kapcsolatban kell figyelembe vennünk, ezért az ábrán a két PLA elem K_1, K_2 kimenetei között huzalozott VAGY kapcsolatot jelöltünk. Ha a rendelkezésre álló PLA elemek kimeneteinek áramköri megvalósítása nem teszi lehetővé a huzalozott VAGY kapcsolat kialakítását, akkor külső VAGY kapukkal, vagy egy további PLA elem alkalmazásával kell a VAGY kapcsolatokat létrehozni. A huzalozott VAGY kapcsolat lehetőséget nyújt arra is, hogy a 2.112. ábrán vázolt esetekben néha elhagyhassuk a kimeneti PLA elemet, ha annak egyébként csak a VAGY kapuit használnánk.

A 2.112...2.116. ábrákon bemutatott kapcsolási módok értelemszerűen együttesen is alkalmazhatók, ha a rendelkezésre álló PLA elemnek több jellemzője is akadályozza a megvalósítást egyetlen PLA elemmel.

Az egyetlen PLA elemmel megvalósított kombinációs hálózatokban fellépő hazárdjelenségekkel nem kell külön foglalkoznunk, hiszen a programozáskor a kétszintű elvi logikai rajzból kell kiindulnunk, amelynek hazárdmentesítését az ismert

eljárásokkal elvégezhetjük. Ha több PLA elem összekapcsolásával alakítjuk ki a kombinációs hálózatot, akkor természetesen nem szabad megfeledkeznünk az ezáltal esetleg fellépő hazárdjelenségek következményeinek ellenőrzéséről.

A dekompozíció elvének alkalmazhatóságáról PLA elemekből történő felépítés esetén ugyanaz állapítható meg, mint a memóriaelemek esetében.

Összehasonlítva a memóriaelemek és a PLA elemek alkalmazásának hatását a megvalósított kombinációs hálózat működési sebességére, megállapíthatjuk, hogy PLA elemek alkalmazása esetén nagyobb működési sebességet tudunk elérni. Ennek oka egyrészt az, hogy a PLA elemek tulajdonképpen kétszintű kombinációs hálózatoknak tekinthetők, másrészt pedig az, hogy technológiai okokból a PLA elemek eleve gyorsabb működésűek, mint a memóriaelemek.

A hazárdmentességet PLA elemek esetében a tervezés során az ismert módszerekkel biztosíthatjuk, ellentétben a memóriaelemek ciklusidőn belüli nem definiált viselkedésével.

A tervezés menete memóriaelemek alkalmazása esetén egyszerűbb, hiszen közvetlenül az igazságtáblából olvashatjuk ki a beírandó adatokat, míg PLA elemek esetén a programozáshoz létre kell hozni a megismert eljárásokkal az egyszerűsített hazárdmentes, kétszintű elvi logikai rajzot.

Mind a memóriaelem, mind a PLA elem univerzális építőelemnek tekinthető, mert a tárolt adatok, ill. az összekötési program és a bemeneti jelek kapcsolódási pontjainak meghatározásával megadható az elem által megvalósított kombinációs hálózat. Az ilyen univerzális építőelemek egyes típusai viszonylag egyszerűen átprogramozhatók, vagyis a megvalósított kombinációs hálózat megváltoztatható a logikai kapcsolási terv megváltoztatása nélkül.

3. Sorrendi hálózatok tervezése

Ebben a fejezetben először a sorrendi hálózatok működési modelljeivel foglalkozunk, majd bemutatjuk működésük leírásának legfontosabb módjait: az állapottáblát és az állapotgráfot. Ezután megismerkedünk a legismertebb elemi sorrendi hálózatok (flip-flopok) elvi működésével és felépítésével. A szinkron és az aszinkron sorrendi hálózatok tervezési lépéseit egy-egy egyszerű példán követjük végig az elvi logikai rajz meghatározásáig. Részletesen megvizgáljuk az integrált áramköri építőelemek szinkron flip-flopjainak logikai felépítését, majd bemutatjuk a sorrendi hálózatok analízisének lépéseit.

A bemutatott tervezési lépések közül az állapotösszevonásra és az állapotkódok megválasztására megfogalmazható legismertebb eljárásokat foglalja össze a fejezet további része. Ezt követően a sorrendi hálózatok dekompozíciójának alapgondolatát és a felmerülő nehézségeket vázoljuk, majd a sorrendi hálózatok kiindulási állapotának beállítási lehetőségeivel foglalkozunk. A fejezet anyagában nagy súllyal szerepel a sorrendi hálózatokban is meglevő jelkésleltető hatások által okozott működési hibák vizsgálata és elkerülésük módja.

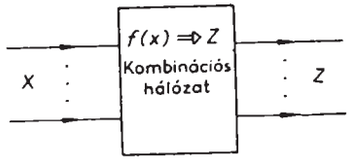
3.1. A sorrendi hálózatok fogalma és csoportosítása

Mint ismeretes, a kombinációs logikai hálózatok csak olyan logikai feladatok megoldására valók, amelyekben az egyes következmények kizárólag a mindenkori, éppen teljesülő feltételektől függenek. Így a kombinációs hálózat minden egyes bemeneti kombinációhoz egyértelműen egy-egy kimeneti kombinációt rendel hozzá:

$$f(X) \Rightarrow Z,$$

ahol X a bemeneti kombinációk halmaza, Z a kimeneti kombinációk halmaza, f a hozzárendelést megvalósító leképezés, amely — mint tudjuk — annyi logikai függvénnyel adható meg, ahány kimenetű a kombinációs hálózat (3.1. ábra).

Könnyen belátható, hogy az f leképezés nem kölcsönösen egyértelmű, hiszen a kimeneti kombináció ismeretében általában nem tudjuk meghatározni az azt előidéző bemeneti kombinációt. Ugyanis nem szükségszerű, hogy különböző bemeneti kom-



3.1. ábra. A kombinációs hálózatok által megvalósított leképezés szemléltetése

binációk más-más kimeneti kombinációt hozzanak létre. (Pl. ÉS kapu mint elemi kombinációs hálózat.)

Ha a megoldandó logikai feladat az egyes következményeket nem kizárólag a mindenkori, éppen teljesülő feltételek alapján írja elő, hanem az a megelőzően teljesült feltételektől is függ, akkor erre a célra *sorrendi (szekvenciális)* logikai hálózatot kell terveznünk. A sorrendi hálózat a kimeneti kombináció előállításához a pillanatnyi bemeneti kombináción kívül a korábban fennállt bemeneti kombinációkat is — sőt általában azok sorrendjét is — figyelembe veszi. A sorrendi hálózatok — ellentétben a kombinációs hálózatokkal — képesek arra, hogy ugyanahhoz a bemeneti kombinációhoz esetenként más-más kimeneti kombinációt állítsanak elő. (Megjegyezzük, hogy a kombinációs és a sorrendi hálózatok összehasonlításához célszerű átismételni az 1.3. fejezetet.)

A sorrendi hálózatok pillanatnyi kimeneti kombinációit tehát általában csak akkor tudjuk egyértelműen meghatározni, ha ismerjük:

— a pillanatnyi bemeneti kombinációt,

— a pillanatnyi bemeneti kombinációt megelőző bemeneti kombinációkat és azok sorrendjét.

Mivel a sorrendi hálózat ugyanarra a bemeneti kombinációra különbözőképpen reagálhat attól függően, hogy milyenek voltak a hálózatot ért megelőző hatások, ezért ezeket ismerni kell. A hálózatot ért megelőző hatásoktól való függés megvalósítása céljából a sorrendi hálózatnak minden egyes bemeneti kombináció fellépésének hatására elő kell állítania egy olyan ún. szekunder kombinációt, amely a soron következő bemeneti kombináció mellett a hálózat előéletét (tehát a hálózatra hatott korábbi bemeneti kombinációkat és azok fellépésének sorrendjét) hivatott képviselni. A szekunder kombinációk tehát a sorrendi hálózat „emlékező" jellegét valósítják meg. Így egy adott időpontban az aktuális bemeneti kombináció a fellépésekor fennálló szekunder kombinációval együtt egyrészt létrehozza a kimeneti kombinációt, másrészt pedig előállítja az új szekunder kombinációt, amely ezáltal az aktuális bemeneti kombináció hatását is „előéletként" képviseli majd a soron következő bemeneti kombináció fellépésekor. A sorrendi hálózatnak tehát ún. rekurzív módon kell gondoskodnia a szekunder kombinációk módosításáról. A szekunder kombinációkat — a fentiekben leírt szerepükből következően — a sorrendi hálózat *állapotainak* nevezik, és ún. *szekunder (vagy állapotjellemző)* logikai változók értékkombinációjaként jönnek létre.

Mint láttuk a szekunder változók aktuális értékei függenek megelőző értékeiktől is, ezért valójában a szekunder változók értékeinek *visszacsatolása* érvényesül a sorrendi hálózatban. Minden kialakuló szekunder kombináció a hálózat bemenetére visszahatva teszi lehetővé az előélet folyamatos aktualizálását, vagyis az állapotváltozások révén ily módon válik képessé a sorrendi hálózat arra, hogy azonos bemeneti kombinációhoz annak fellépési időpontjától függően más-más kimeneti kombinációt állítson elő. A hálózat által megoldandó logikai feladattól függ az, hogy az előírt működéshez minimálisan hány szekunder kombináció, azaz állapot szükséges. A működés során adott bemeneti kombináció mellett létrejövő szekunder kombinációk a soron következő bemeneti kombinációval együtt hozzák létre a soron következő kimeneti- és szekunder kombinációt. Emiatt úgy képzelhetjük, hogy a létrejövő szekunder kombinációk visszajutnak a hálózat bemenetére a szekunder változók értékeinek visszacsatolása révén.

Így a sorrendi hálózat működését az alábbi két leképezéssel adhatjuk meg:

$$f_z(X, y) \Rightarrow Z,$$

$$f_y(X, y) \Rightarrow Y,$$

ahol X a bemeneti kombinációk halmaza; Z a kimeneti kombinációk halmaza; y a bemenetre pillanatnyilag visszajutott szekunder kombinációk, azaz az ún. pillanatnyi állapotok halmaza, Y az X és y által létrehozott soron következő szekunder kombinációk, azaz a következő állapotok halmaza, f_Z a kimeneti kombinációt előállító leképezés, f_y a szekunder kombinációt előállító leképezés.

Mivel minden kialakuló szekunder kombinációról feltételeztük, hogy visszajut a bemenetre, ezért a pillanatnyi és a következő állapotok, azaz szekunder kombinációk halmaza tulajdonképpen ugyanaz a halmaz, amelyet a továbbiakban *állapothalmaznak* nevezzük. A y és Y jelölésbeli megkülönböztetésének csupán az a szerepe, hogy a sorrendi hálózat működésének egyes fázisait, vagyis az állapotváltozások menetét szemléltesse.

A kimeneti kombinációt előállító leképezést az alábbi alakban is definiálhatjuk:

$$f'_z(y) \Rightarrow Z.$$

Ilyenkor a kimeneti kombinációt a bemeneti változások csak látszólag nem befolyásolják, mert az

$$f_y(X, y) \Rightarrow Y$$

leképezés Y-t ezúttal is X-től függően állítja elő és Y a visszacsatolás révén előbb-utóbb y-ná válik. Így az $f'_z(y) \Rightarrow Z$ leképezés esetén a kimeneti kombináció előállításában a bemeneti változások hatását is y megváltozása képviseli közvetve.

A kimeneti kombinációt előállító leképezés mindkét említett definíciója alapján tervezhetünk sorrendi hálózatot egy adott logikai feladat megoldására. Természetesen a tervezés menete és a kialakuló hálózat felépítése általában különbözik a kétfajta definíció szerint. Mint később látni fogjuk a sorrendi hálózat felhasználásának van néhány olyan szempontja, amelynek alapján eldönthetjük, hogy a kimeneti kombinációt előállító leképezésnek melyik definíciója előnyösebb esetenként. Annyit már most

is előrebocsátunk, hogy az $f_z'(y) \Rightarrow Z$ leképezés esetén általában pótlólagos szekunder kombinációkra (állapotokra) van szükség, hiszen ekkor a megoldandó feladatban előírt kimeneti kombinációk megkülönböztetéséhez csak az y kombinációk állanak rendelkezésre. (Az $f_z(X, y) \Rightarrow Z$ leképezés esetén ugyanis a Z kombinációk megkülönböztetésére az X kombinációk is felhasználhatók.)

A kimeneti kombinációk kétfajta előállítási módja szerint a sorrendi hálózatokat két csoportra osztják:

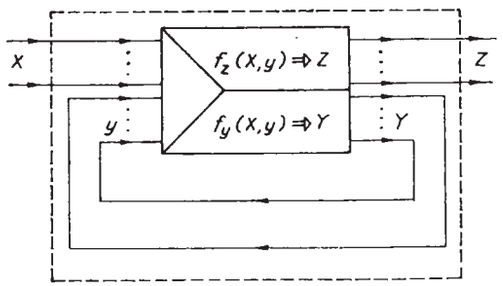
$$f_z(X, y) \Rightarrow Z$$

esetén *Mealy-modell* szerint működőnek,

$$f_z'(y) \Rightarrow Z$$

esetén *Moore-modell* szerint működőnek nevezik a sorrendi hálózatot.

Vizsgáljuk meg a sorrendi hálózatok működését f_z és f_y leképezések feltételezésével először a 3.2. ábra alapján.



3.2. ábra. A sorrendi hálózatok által megvalósított leképezések szemléltetése

Egy adott X bemeneti kombináció és az éppen fennálló y kombináció hatására létrejön egy Z és Y kombináció, ahogyan azt az f_z és f_y leképezések előírják. Ha változatlan marad is az X bemeneti kombináció, általában akkor sem biztos, hogy a hálózat rögtön nyugalomba kerül. Az Y kombináció ugyanis a visszacsatolás következtében y-ként visszajut a bemenetre és f_z, ill. f_y leképezésektől függően — állandó X mellett is — újabb Z és Y értékeket hozhat létre. Az így kialakult Y új y kombinációt hoz létre a bemeneteken és így tovább. Nyugalmi állapot egy adott X bemeneti kombináció mellett csak akkor jöhet létre, ha egy kialakult Y kombináció a bemenetre y-ként visszajutva az $f_y(X, y)$ leképezés alapján változatlan Y kombinációt hoz létre. Ez természetes, hiszen a 3.2. ábra alapján is könnyen beláthatjuk, hogy állandósult állapotban csakis $Y=y$ állhat fenn a közvetlen visszacsatolások miatt. A hálózatnak ezt az állapotát a továbbiakban *stabil állapotnak* nevezzük. Egy adott X bemeneti kombináció hatására tehát az Y és y értékek addig változnak, míg az illető X mellett a stabil állapot ki nem alakul. Természetesen ezen változások során minden egyes $Y \neq y$ szekunder kombináció csak átmenetileg állhat fenn. Ezeknek az ún. *instabil állapotoknak* megfelelő szekunder kombinációk fennállási idejét az szabja meg, hogy egy Y kombináció mennyi idő alatt jut vissza a visszacsatoló ágakon a

bemenetre, valamint az, hogy az f_y leképezést megvalósító hálózat mennyi idő alatt hozza létre az új Y kombinációt. Az instabil állapotok fennállási idejét tehát a visszacsatoló ágak jelterjedési késleltetése, valamint az f_y leképezést megvalósító hálózat megszólalási ideje határozza meg. Nyilvánvaló, hogy a késleltető hatások mértéke általában az Y, y kombinációk értékétől is függ. Ezért nem mondhatjuk meg általánosságban, hogy valamely stabil állapotot megelőző instabil állapotok egyenként mennyi ideig állnak fenn.

Ha az instabil állapotváltozások ideje alatt éri a sorrendi hálózatot a bemeneti változás, akkor az ennek hatására kialakuló Z és Y kombináció attól fog függeni, hogy a bemeneti változás pillanatában éppen melyik instabil állapot állt fenn. Ez pedig az említett késleltetési hatások kézbentarthatatlansága miatt teljesen véletlenszerű. Emiatt az X bemeneti kombináció nem változhat tetszőlegesen gyorsan. Természetesen olyan f_z és f_y leképezések is elképzelhetők, amelyek nem minden X bemeneti kombináció mellett hoznak létre stabil állapotot. Ekkor létezhetnek tehát olyan bebemeneti kombinációk is, amelyek fennállása idején nem alakul ki stabil állapot, hanem az instabil állapotok valamilyen ciklus szerint ismétlődnek. Ennek következtében az Y és általában a Z kombinációk is ciklikusan változnak. A változás periódusideje természetesen csak egy adott kombináció értékére vonatkozhat és nem jelenthet állandó ismétlődési időt, hiszen az instabil állapotok időtartamát meghatározó késleltetési hatások időben is változhatnak. Ha egy bemeneti kombináció mellett nem alakul ki stabil állapot, hanem hatására az említett módon az instabil állapotok állandóan váltják egymást, akkor — az elterjedt szóhasználat szerint — a sorrendi hálózat *oszcillál* (l. később).

Az eddig elmondottak szerint működő sorrendi hálózatokat *aszinkron* sorrendi hálózatoknak nevezik. Megfigyelhetjük, hogy az f_z leképezés megvalósítható kombinációs hálózattal, hiszen az X és y kombinációk függetlenek a Z kombinációktól. Az f_z leképezés így annyi logikai függvényt jelent, ahány kimenetű a sorrendi hálózat. Ha a visszacsatoló ágakat felnyitottnak képzeljük, akkor az f_y leképezés is kombinációs hálózatnak tekinthető, hiszen visszacsatolások nélkül a bemenetre jutó y kombinációk függetlenek az Y kombinációktól. Ennek a gondolatmenetnek az alapján kimondhatjuk, hogy *az aszinkron sorrendi hálózatok minden esetben megvalósíthatók visszacsatolt kombinációs hálózattal.*

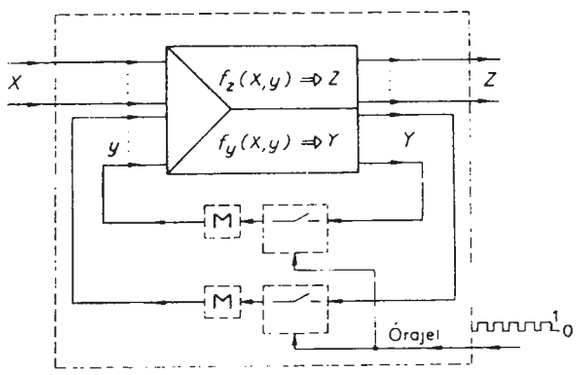
Az aszinkron sorrendi hálózatok tervezése során tehát a megoldandó logikai feladattól függően meg kell határoznunk az f_z és f_y leképezéseket és realizálnunk kell azokat. A realizálás során az elmondottak szerint tervezhetünk kombinációs hálózatot, amelyet megfelelő módon visszacsatolunk.

A megoldandó logikai feladattól függ az, hogy a sorrendi hálózatnak minimálisan hány állapottal kell rendelkeznie az előírt működéshez. Az aszinkron sorrendi hálózat leírt működési módjából következően tehát a megoldandó logikai feladat határozza meg, hogy legalább hány stabil állapottal kell rendelkeznie a hálózatnak. Ha ugyanis vizsgálatunkból kizárjuk az oszcillációt, akkor új bemeneti kombináció csak azután juthat a hálózatra, miután a megelőző bemeneti kombináció mellett a stabil

állapot már kialakult. A tervezés során azonban azt is biztosítanunk kell, hogy az előírt stabil állapotok valóban létre is jöjjenek. Mint láttuk új stabil állapot — feltéve, ha kialakul — mindig átmenetileg fennálló instabil állapot(ok) után jön létre. Az instabil állapotokat tehát szintén létre kell hoznia a hálózatnak, hiszen éppen ezek segítségével juthat az előírt stabil állapotba. Az instabil állapotokat ezért úgy kell megterveznünk, hogy mindig a megfelelő stabil állapotba vezessék a hálózatot. Ennek során természetesen törekednünk kell arra, hogy egyik stabil állapotból a másikba való átmenet a lehető legkevesebb instabil állapot közbeiktatásával valósuljon meg. Így hozhatjuk létre ugyanis a lehető leggyorsabb aszinkron sorrendi hálózatot, hiszen az ismertetett működési mód szerint ebben az esetben kell az egyes bemeneti kombinációknak a lehető legrövidebb ideig állandóknak maradniuk.

A hálózatot *normál aszinkron* hálózatnak nevezik, ha bármely két stabil állapota közötti átmenet során legfeljebb egy instabil állapota alakul ki. Az instabil állapotok szükségessége miatt aszinkron hálózatok esetén általában több szekunder változót kell alkalmaznunk, mint amennyit a megoldandó logikai feladat alapján megkülönböztetendő állapotok száma igényelne.

Bemutatjuk, hogy ez a szekunderváltozó-többlet nem szükséges, ha a 3.3. ábra szerint a visszacsatoló ágakat megfelelő módon periodikusan nyitjuk, ill. zárjuk a sorrendi hálózat működése során. Az ábrán a visszacsatoló ágakban jelképesen olyan kapcsolókat ábrázoltunk, amelyek periodikusan ismétlődő négyszögimpulzusok (órajel) hatására létrehozzák, ill. megszüntetik a visszacsatolást.



3.3. ábra. A visszacsatoló ágak periodikus nyitásának modellje

Az egyes kapcsolók után rajzolt M jelű elemekről tételezzük fel, hogy kimeneteiken azt az értéket jelenítik meg, amely a kapcsoló zárásainak pillanatában a bemenetükre jutott. Tételezzük fel továbbá, hogy ezt a kimeneti értéket mindaddig fenntartják, amíg egy újabb kapcsolózárás be nem következik. Így az M jelű elemek kimeneti értéke a kapcsolók nyitásakor vagyis a visszacsatoló ágak megszakítása alatt nem változik. Az M jelű elemek tehát tároló (memória) tulajdonságúak is.

A visszacsatoló ágban feltételezett, jelképesen ábrázolt elemek és hatások konkrét megvalósítási módjaival később foglalkozunk. Egyelőre azt vizsgáljuk meg, hogy

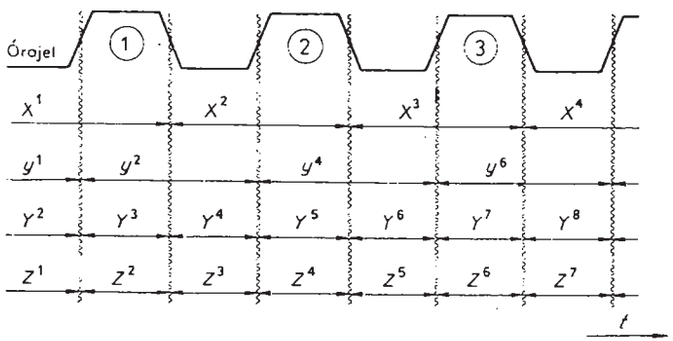
jelenlétük milyen változásokat okozhat a sorrendi hálózat működésében. Tételezzük fel, hogy az órajel logikai 1 szintjének időtartama alatt a visszacsatoló ágak zártak, a 0 szint időtartama alatt pedig nyitottak. Kövessük ilyen esetben a sorrendi hálózat működését.

Induljunk ki abból, hogy vizsgálatunk kezdetekor két óraimpulzus között a hálózat éppen nyugalomban van. Ekkor tehát a visszacsatoló ágak nyitottak, az éppen jelenlevő X és az előző óraimpulzus hatására a bemenetre visszajutott y alakítja ki a Z és Y kombinációt. Ha ezek után megérkezik az óraimpulzus, akkor ennek hatására az éppen fennálló Y kombináció y-ként a bemenetre jut az M jelű elemek közvetítésével. Az így kialakuló új y kombináció az éppen jelenlevő X-szel együtt hatva új Y és Z kombinációkat hozhat létre. E jelenség minden újabb óraimpulzus hatására megismétlődik mindaddig, amíg a már megismert módon stabil állapot ki nem alakul. Ha a mindenkori X megváltoztatásával mindig megvárjuk a stabil állapot kialakulását, akkor a hálózat működése lényegében aszinkron jellegű. A megismert aszinkron működéshez képest csupán annyi az eltérés, hogy az instabil állapotok az órajel ütemezésében következnek egymás után. Fennállási időtartamuk ezért már nem véletlenszerű, hanem pontosan egy órajel periódusideje. Következésképpen a hálózat működési sebességét az órajel frekvenciája korlátozza, hiszen két óraimpulzus között a visszahatások meg vannak szüntetve.

Az így működő hálózatokat *ütemezett aszinkron* sorrendi hálózatoknak nevezzük. Könnyen belátható, hogy a szükséges szekunder változók számára és a tervezés menetére vonatkozólag az ütemezett működéstől függetlenül érvényesek az aszinkron hálózatok esetén elmondottak. Az ütemezett aszinkron hálózatok azonban mindenképpen lassúbb működésűek, mint az ütemezés nélkülick. Az ütemezés ugyanis a jelvisszahatást az órajel frekvenciájától függő mértékben lassítja, vagyis egy állapotváltozás ideje azonos építőelemek esetén is megnövekszik az ütemezés nélküli esethez képest. Az ütemezés egyetlen előnye a sebességcsökkenés árán az, hogy az instabil állapotok fennállási időtartamát az órajel periódusideje rögzíti. Az ütemezett aszinkron működés tehát szintén igényli az instabil állapotok megvalósítását.

Az aszinkron jellegű működés helyett újfajta működési módhoz jutunk, ha a visszacsatoló hatások leírt ütemezése esetén megengedjük X változását instabil állapotban is. Így minden kialakuló állapot — akár stabil, akár instabil — a logikai feladatban előírt bemeneti kombinációkkal együtt létrehozhat új kimeneti kombinációt (Z) és új állapotot (Y). Az instabil állapotoknak tehát megszűnik az a kizárólagos szerepük, hogy stabil állapotba vezessék a hálózatot. Fennállásuk ideje az órajel ismeretében jól meghatározható, így X megváltozásának időpontja egyértelműen kijelölhető. Ha megengedjük, hogy minden órajelperiódusban új X kombináció juthasson a bemenetre, akkor ún. *szinkron sorrendi hálózatot* kapunk. Ilyenkor ugyanis a bemeneti változások szinkronban vannak az órajellel. Nyilvánvaló, hogy az ilyen szinkron működés esetén nincs jelentősége annak, hogy egy állapot stabil vagy instabil, hiszen minden y kombináció — ellentétben az aszinkron működéssel — új X kombinációval találkozik. Ezáltal egy X kombináció fennmaradása az ismétlődéstől nem is külön-

böztethető meg. Szinkron sorrendi hálózatok esetében ezért nem szükséges több szekunder változót alkalmazni, mint amennyit a megoldandó logikai feladatban megkülönböztetendő állapotok igényelnek. Az aszinkron működéssel ellentétben ugyanis az instabil állapotok is hasznosíthatók a megoldandó logikai feladat szempontjából. Hátrányos következmény viszont, hogy a bemeneti változásoknak követniük kell az órajel ütemét, és ezt a szinkronizációt a szinkron hálózat bemeneti jeleit előállító megelőző hálózatban kell megoldani, vagy külön szinkronizáló hálózatot kell tervezni. Sőt, a kimeneti kombináció értelmezése sem olyan egyszerű, mint aszinkron esetben. Ennek szemléltetésére a 3.4. ábrán az óraimpulzushoz képest bejelöltük az X, y, Y és Z változásokat Mealy-modell szerinti működés esetén, azzal a feltételezéssel, hogy az X változások az óraimpulzus eltűnésekor (a lefutó él hatására) játszódnak le. A változások időpontját jelölő hullámos vonal arra utal, hogy az időpontok a késleltetési viszonyoktól függően változhatnak. A hálózat kiindulási állapotában a bemeneten levő kombinációkat X^1 és y^1 jelöli. Feltételeztük, hogy a következő állapot: $f_y(X^1, y^1) \Rightarrow Y^2$.



3.4. ábra. Mealy-modell szerinti szinkron sorrendi hálózat működésének szemléltetése

A további kombinációváltozásokat azáltal szemléltettük, hogy minden tartományban más felső indexet alkalmaztunk. A felső indexek X és Z esetében csak az eltérő kombinációértékek megkülönböztetésére szolgálnak, Y és y esetében viszont emellett még a visszacsatoló ágakon történő jelvisszahatást is kifejezik. Így például y^2 azért alakul ki, mert az órajel megjelenésének pillanatában Y^2 állt fenn. Az y, Y kombinációkra vonatkozóan tehát az azonos felső index azonos kombinációt jelent. Az y kombináció megváltozása természetesen új Y kombinációt hozhat létre. Például y^2 megjelenésének hatására Y^3 kialakulását tételeztük fel.

Ha az ábrázolt első óraimpulzus lefutó élének hatására a bemeneti kombináció X^1-ről X^2-re változik, akkor az ábrázolt kombinációváltozások a leképezések alapján:

$$f_y(X^2, y^2) \Rightarrow Y^4; \quad f_z(X^2, y^2) \Rightarrow Z^3$$

A második óraimpulzus felfutó élének hatására Y^4-ből y^4 lesz és

$$f_y(X^2, y^4) \Rightarrow Y^5; \quad f_z(X^2, y^4) \Rightarrow Z^4.$$

A második óraimpulzus lefutó éle után már X^3 jut a bemenetre, amelynek hatására

$$f_y(X^3, y^4) \Rightarrow Y^6; \quad f_z(X^3, y^4) \Rightarrow Z^5$$

áll elő.

A harmadik óraimpulzus felfutó éle Y^6-ot juttatja vissza a bemenetre y^6-ként, azaz

$$f_y(X^3, y^6) \Rightarrow Y^7; \quad f_z(X^3, y^6) \Rightarrow Z^6.$$

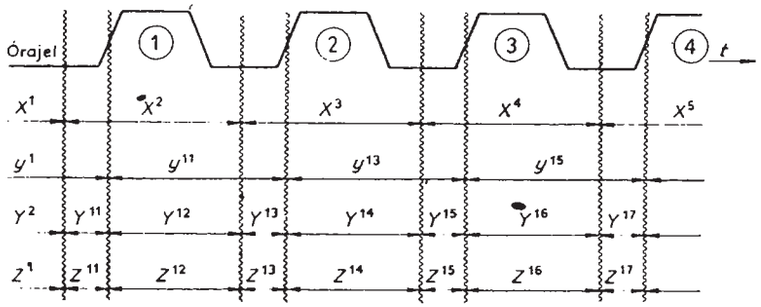
Az ábra alapján hasonló módon folytatható a többi ábrázolt bemeneti változás vizsgálata. Megfigyelhetjük, hogy egy bemeneti kombináció fennállásának idején két kimeneti kombináció keletkezik. (Pl. X^3 idején Z^5 és Z^6.) Ezt az okozza, hogy X és y különböző időpontokban változik és mindkét változás befolyásolja a leképzések eredményét. Ezért kell rögzíteni szinkron sorrendi hálózatok esetén a kimeneti kombináció értelmezésének időtartományát. A kimenetekre kapcsolódó további logikai hálózatok megvalósításakor ugyanis döntő szempont, hogy például X^3 esetén a Z^5 vagy a Z^6 kimeneti kombinációt kell-e bemeneti kombinációnak tekinteniük a további feldolgozás szempontjából. Ennek eldöntése vagy a további logikai hálózatok által megoldandó logikai feladatok definiálásának pótlólagos feltétele, vagy külön hálózatot kell erre a célra tervezni. Az ilyen ún. kimeneti szinkronizálás elve lehet például az, hogy a kimeneti kombinációt az órajellel vagy annak negáltjával hozzuk ÉS kapcsolatba attól függően, hogy a kettő közül melyik kimeneti kombinációt akarjuk értelmezni a következő hálózat bemeneti kombinációjaként.

A 3.4. ábra alapján azt is megállapíthatjuk, hogy a feltételezett működés során keletkeznek olyan kombinációk is, amelyeknek a vizsgált bemeneti kombinációsorozat hatására nincs alkalmuk a bemenetre visszahatni, vagyis y-ná válni. Ilyenek példánkban a páratlan felső indexű Y-ok, mert mielőtt a visszahatást lehetővé tevő óraimpulzus felfutó éle megjelenne, a bemeneti változás megváltoztatja azokat.

Az ilyen Y kombinációknak tehát a vizsgált bemeneti kombinációsorozat mellett a szinkron működésre nincs semmiféle hatásuk, ezért figyelmen kívül hagyhatók.

Megjegyezzük azonban, hogy más bemeneti kombinációsorozat hatására ezeket az ezúttal figyelmen kívül hagyható Y kombinációértékeket előállíthatják a leképezések olyan időtartományokban is, amelyekben y-ként történő visszahatásuk létrejön. Így egy hálózat teljes működési előírása szempontjából csak azok az Y kombinációk hagyhatók figyelmen kívül, amelyek a megoldandó logikai feladatból eleve specifikálatlanoknak adódnak.

A 3.4. ábrán feltételeztük, hogy a bemeneti változások az óraimpulzus lefutó éleinek hatására történnek. Lényegében változatlan marad a működés, ha a bemeneti változás az óraimpulzus két felfutó éle között bárhol történik (természetesen minden órajel-periódusban csak egyszer). Ha a bemeneti változások időpontját nem tudjuk



3.5. ábra. Mealy-modell szerinti szinkron sorrendi hálózat működésének szemléltetése, ha a vizsgálat kezdetének időpontjához képest az X változás megelőzi y megváltozását

annyira pontosan beállítani a két felfutó él között, mint azt a 3.4. ábrán tettük, akkor a változás időpontjának bizonytalansága megnehezíti a kimeneti kombináció értelmezését. Ilyenkor ugyanis például az órajellel való kapuzás említett módja kevésbé alkalmas az értelmezendő kimeneti kombinációk kijelölésére, hiszen a bemeneti változások időpontja (és így a kimeneti változásoké is) bizonyos határok között órajel-periódusként változhat.

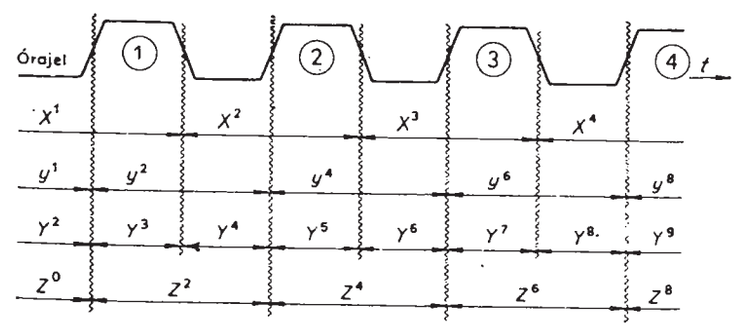
A 3.4. ábrán bemutatott működés módosulhat, ha a bemeneti változások a vizsgálat kezdetének időpontjához képest minden órajel-periódusban megelőzik az y változásokat. Ilyenkor a 3.4. ábrán feltételezett változatlan kiindulási állapot után következő változatlan bemeneti kombinációsorozat eltérő kimeneti kombinációsorozatot okozhat. Ezt szemlélteti a 3.5. ábra, amelyen feltételeztük, hogy a vizsgálat kezdetekor (például a hálózat bekapcsolásakor) fennálló bemeneti kombináció az első óraimpulzus felfutó éle előtt megváltozik. Megfigyelhetjük, hogy esetünkben már X^2 fellépésekor eltérés mutatkozik a 3.4. ábrához képest, hiszen nem biztos, hogy Y^4 jön létre. Ennek oka, hogy X^2-vel nem y^2, hanem még y^1 hat együtt a bemeneten, ami a leképezésektől függően Y^4-től eltérő kombinációt hozhat létre a szekunder kombináció következő értékében. Az eltérő érték lehetőségének szemléltetésére jelöltük ezt az Y kombinációt a 3.5. ábrán Y^{11}-gyel, mivel a 11-es felső index a 3.4. ábra működésrészletében nem fordul elő. A további működést is erősen befolyásolhatja Y^{11} kialakulása, hiszen visszahatva a bemenetre y^{11}-et hozhat létre, ami szintén eltérő a 3.4. ábrán szereplő y kombinációktól. Ezáltal a további létrejövő Y kombinációk (Y^{12}, Y^{13}, Y^{14}, Y^{15}, Y^{16}, Y^{17}) is eltérhetnek a 3.4. ábrán keletkezőktől. A kimeneti kombinációsorozat eltérésének lehetősége fentiekből már következik (Z^{11}, Z^{12}, Z^{13}, Z^{14}, Z^{15}, Z^{16}, Z^{17}).

Ha a bemeneti változások időpontjában akkora a bizonytalanság, hogy a működés során a 3.4. és 3.5. ábrán vázolt helyzetek órajel-periódusonként véletlenszerűen váltakozva fordulhatnak elő, akkor a hálózat működése nem adható meg egyértelműen. Ezért a továbbiakban az ilyen eseteket nem vizsgáljuk és feltételezzük, hogy a bemeneti változások időpontjának bizonytalansága a megengedhető határok között marad. A tervezés előtt azt is el kell döntenünk, hogy a hálózat működésének jellege a 3.4. vagy a 3.5. ábrának felel-e meg. Láttuk, hogy a két esetben ugyanaz a hálózat ugyanarra a bemeneti kombinációsorozatra eltérő módon reagálhat. A két lehetséges működési változat közül elterjedtebb a 3.4. ábra szerinti. A továbbiakban ezt a működési módot tételezzük fel minden szinkron hálózatról. Eszerint kiinduláskor (például bekapcsoláskor, alaphelyzetben) feltételezzük, hogy az éppen fennálló bemeneti kombináció csak azután változik meg, miután az óraimpulzus hatására y megváltozott. Ezek után azt is beláthatjuk, hogy nem szükségszerű az y változásokat az óraimpulzusnak éppen a felfutó éléhez rendelni. Azt is megtehetnénk például, hogy a lefutó élhez rendeljük a visszacsatoló ágak zárását. Ez ugyanis működésben nem okoz változást, ha az X változások időpontjait is úgy módosítjuk, hogy az y változások időpontjaihoz viszonyított helyzetük a 3.4. ábra szerinti maradjon. Ilyenkor természetesen a visszacsatoló ágakban feltételezett jelképes órajelvezérlésű kapcsolók és tárolóelemek működését az eddigiektől eltérő módon kellene definiálnunk.

A bemeneti változások időtartamának bizonytalansága — ha nem is akkora, hogy a működés jellegét befolyásolja — mint láttuk, a kimeneti kombinációk értelmezését megnehezíti. Ebből a szempontból jelentős gyakorlati előnnyel rendelkeznek a Moore-modell szerint működő szinkron sorrendi hálózatok.

A 3.6. ábra a Moore-modell szerinti működését szemlélteti. Megfigyelhetjük, hogy Z ritkábban változik, mert csak y változásai hatnak rá közvetlenül. (Ezért van szüksége a hálózatnak több y kombinációra, azaz több szekunder változóra a Mealy-modellhez képest, ha a megoldandó logikai feladat több kimeneti kombinációt igényel, mint ahány állapotot.) Az ábrán feltételeztük, hogy az $f'_z(y)$ leképezés egy adott y kombináció hatására rendre ugyanazt a kimeneti kombinációt szolgáltatja, mint a 3.4. ábrához tartozó $f_z(X, y)$ leképezés az adott y kombináció megjelenésekor. Például:

$$f'_z(y^4) \Rightarrow Z^4 \quad \text{és} \quad f_z(X^2, y^4) \Rightarrow Z^4.$$



3.6. ábra. Szinkron sorrendi hálózat Moore-modell szerinti működésének szemléltetése

így Moore-modell esetén a Z^1, Z^3, Z^5 és Z^7 kimeneti kombinációk nem lépnek fel, továbbá az X változások időpontjának megengedett bizonytalanságai a kimeneti változások időpontját nem befolyásolják. A leképezések fenti értelmezéséből következik, hogy a 3.6. ábrán a kiinduláskor jelenlevő Z^0 kimeneti kombináció nem szerepel a 3.4. ábrán vázolt működési tartományban.

Foglaljuk össze, hogy a definiált szinkron működéshez milyen ún. *szinkronizációs feltételeket* kell biztosítani a hálózat megvalósításakor:

— A visszacsatoló ágak órajelvezérléssel periodikusan megszakíthatók legyenek, és tartalmazzák a definiált működésű tárolóelemeket.
— A bemeneti változások az órajellel szinkronban történjenek (3.4., 3.6. ábra).
— Meg kell oldani a kimeneti kombinációk értelmezési időtartományának kijelölését.

Az utóbbi két szinkronizációs feltétel alapján megállapíthatjuk, hogy a helyes szinkron működés biztosítása megkötést jelent a szinkron hálózatot megelőző és követő hálózatok működésére is.

A továbbiakban szinkron hálózatok esetén mindig feltételezzük a szinkronizációs feltételekkel megfogalmazott működést. A szinkronizációs feltételek teljesítésének tényleges módját integrált áramköri építőelem-készlet esetén csak az aszinkron hálózatok tervezési lépéseinek megismerése után célszerű részletezni. Mivel azonban a szinkron hálózatok tervezése logikailag egyszerűbb, ezért először ezzel foglalkozunk. Ezt az is indokolja, hogy a szinkron hálózatok tervezésének számos lépése részét képezi az aszinkron hálózatok tervezési eljárásának. A szinkron hálózatok tervezésekor tehát a szinkronizációs feltételek teljesülését mindig feltételezzük.

Hasonlítsuk össze ezek után az aszinkron és a szinkron sorrendi hálózatok főbb tulajdonságait:

Aszinkron sorrendi hálózatok

— az instabil állapotok miatt a szükséges szekunder változók száma általában nagyobb, mint szinkron esetben, többek között ezért logikai tervezésük bonyolultabb;
— a bemeneti változások gyakoriságát, vagyis a működési sebességet csak az építőelemek működési sebessége és a jelterjedési késleltetések korlátozzák;
— logikai tervezésük után nem kell szinkronizációs feltételeket biztosítani, megépíthetők visszacsatolt kombinációs hálózattal is.

Szinkron sorrendi hálózatok

— stabil és instabil állapotok nincsenek értelmezve, a szükséges szekunder változók száma ezért általában kisebb, mint aszinkron esetben, logikai tervezésük egyszerűbb;
— működési sebességet az órajel frekvenciája korlátozza, ezért általában lassúbbak, mint az ütemezés nélküli aszinkron hálózatok;

— a bemeneti változásokra és a kimeneti kombináció értelmezésére szinkronizációs feltételeknek kell teljesülniük;
— logikai tervezésük után, megvalósításuk során biztosítani kell a szinkronizációs feltételeket.

3.2. A sorrendi hálózatok működésének leírása

3.2.1. Az állapottábla

Egy sorrendi hálózat működésének egyértelmű megadásához szükséges, hogy ismerjük az f_z és f_y leképezések eredményét, vagyis az Y és Z kombinációkat minden egyes X és y kombináció esetén. A leképezéseket áttekinthetően ábrázolhatjuk olyan táblázatban, amelyeknek minden egyes rovatához egy-egy X és y kombinációt rendelünk hozzá. Az így kialakított táblázat egyes rovataiba azokat a Z és Y kombinációkat írjuk be, amelyeket a leképezések a rovatokhoz rendelt X és y kombinációk fennállása esetén hoznak létre. Az így kialakított táblázatot ***állapottáblának*** nevezik.

Példaként írjuk fel egy olyan sorrendi hálózat állapottábláját, amelynek 4 db bemeneti kombinációja, 4 db szekunder változókombinációja (állapota) és 4 db kimeneti kombinációja van. Ez természetesen azt jelenti, hogy a hálózatnak legalább két bemenete, két kimenete és két szekunder változója van, hiszen négy kombináció előállításához legalább két változóra van szükség. A tömör jelölés kedvéért az egyes kombinációkat a bináris számok helyett felső indexekkel különböztessük meg egymástól. Így a bemeneti kombinációkat:

$$X^1, X^2, X^3, X^4,$$

a szekunder kombinációkat:

$$Y^1y^1, Y^2y^2, Y^3y^3, Y^4y^4,$$

a kimeneti kombinációkat:

$$Z^1, Z^2, Z^3, Z^4$$

jelöli. A felső indexek értelmezése ugyanaz, mint a 3.1. fejezetben. Tehát X és Z esetében ezúttal is csak a kombinációk megkülönböztetésére szolgálnak. Y és y esetében pedig az azonos felső index azonos kombinációt is jelöl.

Az f_z és f_y leképezéseket egyértelműen megadhatjuk, ha elkészítjük az állapottáblát. Az ismertetett jelölésekkel a 3.7. ábrán egyfajta módon kitöltöttük az állapottáblát, miáltal megadtunk egy f_z és egy f_y leképezést. A vízszintes fejlécben a bemeneti kombinációkat, a függőleges fejlécben pedig az y kombinációkat soroltuk fel. Például a

$y \backslash X$	X^1	X^2	X^3	X^4
y^1	Y^1Z^1	Y^2Z^4	Y^4Z^2	Y^4Z^4
y^2	Y^1Z^4	Y^2Z^3	Y^2Z^2	Y^3Z^1
y^3	Y^4Z^3	Y^3Z^1	Y^3Z^3	Y^1Z^2
y^4	Y^4Z^2	Y^2Z^2	Y^3Z^3	Y^2Z^3

3.7. ábra. Az állapottábla kitöltési módjának szemléltetése

bal oldali legfelső rovatban szereplő Y és Z kombinációértékekből azt olvashatjuk ki, hogy

$$f_z(X^1, y^1) \Rightarrow Z^1,$$

$$f_y(X^1, y^1) \Rightarrow Y^1.$$

Az ily módon egyértelműen megadott leképezések alapján azonban csak akkor követhetjük lépésről lépésre egyértelműen a hálózat működését, ha adott, hogy az aszinkron vagy szinkron. Először tételezzük fel, hogy a 3.7. ábra állapottáblájával megadott hálózat aszinkron működésű. Vizsgáljuk meg a hálózat működését, ha bemeneteire egymás után az

$$X^1, X^2, X^3, X^1, X^3, X^4, X^3$$

bemeneti kombinációk jutnak. Ha aszinkron működést tételezünk fel, akkor az állapottábla rovataiban célszerű megjelölni a stabil állapotokat. Mint ismeretes, minden olyan rovat stabil állapotot jelöl, amelyben szereplő Y kombináció azonos a rovathoz tartozó y kombinációval. A 3.8. ábrán bekarikáztuk a stabil állapotokat jelölő rovatokban az Y kombinációkat. Az adott bemeneti kombinációsorozat vizsgálatakor feltételezzük, hogy a hálózat az

$$X = X^1,$$

$$y = y^1,$$

$$Y = Y^1,$$

$$Z = Z^1,$$

$y \backslash X$	X^1	X^2	X^3	X^4
y^1	Y^1Z^1	Y^2Z^4	Y^4Z^2	Y^4Z^4
y^2	Y^1Z^4	Y^2Z^3	Y^2Z^2	Y^3Z^1
y^3	Y^4Z^3	Y^3Z^1	Y^3Z^3	Y^1Z^2
y^4	Y^4Z^2	Y^2Z^2	Y^3Z^3	Y^2Z^3

3.8. ábra. Az aszinkron működés követése az állapottábla alapján

stabil állapotból (az állapottábla bal oldali legfelső rovatából) indul. Természetesen kiindulhatnánk az

$$X = X^1,$$

$$y = y^4,$$

$$Y = Y^1,$$

$$Z = Z^2,$$

stabil állapotból is, hiszen a hálózat előéletétől függően X^1 esetén ez is stabilizálódhat. Mivel a hálózatra a vizsgálat kezdetét megelőzően jutott bemeneti jelsorozatokról nem rendelkeztünk, ezért a kiindulási stabil állapot megválasztása teljesen önkényes. A 3.8. ábrán bejelölt nyilak mentén követhetők az adott bemeneti kombinációsorozat hatására lejátszódó változások. A választott kiindulási stabil állapotból X^2 bemeneti kombináció hatására olyan instabil állapotba jut a hálózat, amelyhez tartozó Y kombinációt az állapottábla X^2 jelű oszlopából olvashatjuk ki az y^1 jelű sorban, hiszen ez utóbbi csak a keletkező új (következő) Y kombináció visszahatásakor fog megváltozni. Ebben a rovatban Y^2 és Z^4 kombinációkat találunk, vagyis ezeket hozzák létre X^2 és y^1 esetén a leképezések. Ennek következtében Y^2 visszahatása után y^2 kombináció alakul ki a hálózat bemenetén. Ennek hatására a leképezések által előállított Y és Z kombinációk az állapottábla X^2 jelű oszlopának y^2 jelű sorából olvashatók ki, hiszen a leképezések argumentuma már X^2, y^2. Ebben a rovatban viszont Y^2Z^3 kombinációt találunk, aminek következtében a kialakult állapot stabil, hiszen $y^2 = Y^2$. Ezek után mindaddig nem történik változás a hálózatban, amíg X^2 meg nem változik.

Az adott bemeneti kombinációsorozat következő értéke X^3. Ha ez fellép, akkor az előbbiek szerint a kialakuló új Y és Z kombinációk az állapottábla X^3 jelű oszlopának y^2 jelű sorából olvashatók ki. Ebben a rovatban Y^2Z^2-t találunk, tehát az állapot változatlan és stabil, csupán a kimeneti kombináció változott Z^3-ról Z^2-re.

A további bemeneti változások hatására lejátszódó jelenségek hasonló módon olvashatók ki a bejelölt nyilak mentén. Figyeljük meg, hogy az X^1-et követő X^3 bemeneti kombináció két instabil állapoton keresztül vezeti stabil állapotba a hálózatot. Így a vizsgált állapottáblával leírt aszinkron hálózat nem normálműködésű.

Vizsgáljuk meg részletesebben X^4 létrejöttének hatását. A vizsgálandó bemeneti kombinációsorozatban X^3-at követi X^4 az $X = X^3$, $y = y^3$, $Y = Y^3$, $Z = Z^3$ stabil állapotból kiindulva. X^4 fellépésekor az új Y-t az állapottábla Y^4 jelű oszlopának y^3 jelű sora jelöli ki. Ennek értéke jelen esetben Y^1, amelynek visszahatása után ugyanennek az oszlopnak az y^1 jelű sora tartalmazza a következő Y kombinációt, amely Y^4. Ennek visszahatása után hasonló módon megállapítható, hogy Y^2 áll elő, amely visszahatva Y^3-at hoz létre. Ha Y^3 visszajut a bemenetre, akkor újra Y^1 keletkezik és az egész változási ciklus kezdődik elölről. A leírt periodikus állapotváltozás mindaddig folytatódik, amíg X^4 fennáll a bemeneten. Az állapottáblán szembetűnő, hogy

az X^4 jelű oszlopban nem jelölhető be a stabil állapot. Ez azt jelenti, hogy a vizsgált sorrendi hálózat f_z és f_y leképezései $X=X^4$ esetén nem hoznak létre olyan szekunder kombinációt, amely megegyezne az őt létrehozó y kombinációval. A hálózat tehát X^4 fennállása esetén oszcillál, vagyis X^4-re nem teljesíthető az a feltétel, amely szerint a stabil állapot kialakulásáig nem változhat a bemeneti kombináció. Ezért az X^4-et követő bemeneti kombinációváltozás csakis instabil állapotban érheti a rendszert. Mint láttuk, az instabil állapotok fennállásának idejét ütemezés nélküli aszinkron hálózatban nem tarthatjuk kézben, ezért, ha a vizsgált bemeneti sorozatban X^4 után X^3 fellép, nem tudjuk megmondani, hogy melyik stabil állapotba kerül a hálózat. Ez ugyanis attól függ, hogy milyen y kombináció van éppen a bemeneten, amikor az instabil változások ideje alatt a bemeneti változás fellép. Ha ekkor y értéke pl. y^1, akkor X^3 hatására az y^3 stabil állapot alakul ki, ha viszont a bemeneti változás pillanatában y értéke pl. y^2, akkor X^3 az $y^2=Y^2$ stabil állapotot hozza létre. Az ilyen véletlenszerű működés elkerülése céljából a továbbiakban csak olyan aszinkron hálózatokat vizsgálunk és tervezünk, amelyekben nem léphet fel oszcilláció. Ilyenkor minden egyes bemeneti kombinációváltozás hatására előbb-utóbb stabil állapotba kerül a hálózat. Ennek szükséges feltétele, hogy az állapottábla minden egyes oszlopában legalább egy stabil állapot bejelölhető legyen. Ez a feltétel azért szükséges csupán, mert az elégségességhez még annak is teljesülnie kell, hogy a hálózat minden egyes bemeneti kombináció hatására, vagyis minden oszlopban el is jusson a stabil állapotba, ami természetesen további megkötés a leképezésekre, vagyis az állapottábla felépítésére vonatkozóan.

Az eddigiek alapján az alábbi egyszerű, formális módon követhetjük az aszinkron hálózatok működését az állapottábla alapján.

A bemeneti változást megelőző bemeneti kombináció mellett fennálló stabil állapot által kijelölt sorban haladva átjutunk az új bemeneti kombináció által kijelölt oszlopba. A további változások attól függenek, hogy abban a rovatban, amelybe így eljutottunk, milyen Y szerepel. Ha megegyezik a sornak megfelelő y-nal, akkor az állapot stabil és további változás nem történik az újabb bemeneti változásig. Ha viszont a rovatban szerepiő Y kombináció nem azonos a sornak megfelelő y kombinációval, akkor az oszlopon belül átkerülünk abba a sorba, amelyet a rovatban szereplő Y-nal azonos y kombináció jelöl ki. A további változasok ezután ismét attól függenek, hogy az oszlop így kijelölt új sorában milyen kombináció szerepel a rovatban.

Ezek után olvassuk ki az állapottáblából, hogy a vizsgált bemeneti kombinációsorozat hatására milyen kimeneti kombinációsorozatot szolgáltat az adott aszinkron sorrendi hálózat (az X^4 bemeneti kombináció hatását az említett indokok alapján ne vizsgáljuk).

$$\begin{array}{c|c|c|c|c|}
X^1 & X^2 & X^3 & X^1 & X^3 \\
\underline{Z}^1 & Z^4\,\underline{Z}^3 & \underline{Z}^2 & Z^4\,\underline{Z}^1 & Z^2\,\underline{Z}^3
\end{array}$$

(Aláhúzással jelöltük a stabil állapotokhoz tartozó kimeneti kombinációkat.)

$y \backslash X$	X^1	X^2	X^3	X^4
y^1	$Y^1 Z^1$	$Y^2 Z^4$	$Y^4 Z^2$	$Y^4 Z^4$
y^2	$Y^1 Z^4$	$Y^2 Z^3$	$Y^2 Z^2$	$Y^3 Z^1$
y^3	$Y^4 Z^3$	$Y^3 Z^1$	$Y^3 Z^3$	$Y^1 Z^2$
y^4	$Y^4 Z^2$	$Y^2 Z^2$	$Y^3 Z^3$	$Y^2 Z^3$

3.9. ábra. A szinkron működés követése az állapottábla alapján

Tételezzük fel ezután, hogy a 3.7. ábrán szereplő állapottábla szinkron működésű hálózatot ír le. Vizsgáljuk a működést ugyanannak a bemeneti kombinációsorozatnak a hatására, mint aszinkron esetben. A lejátszódó változásokat a 3.9. ábrán bejelölt nyilak mentén követhetjük.

Kiindulásképpen tételezzük fel, hogy a bemeneten y^1 és X^1 áll fenn. Választásunk — az aszinkron esethez hasonlóan — itt is önkényes, hiszen az X^1-t megelőző hatásoktól függően bármelyik y kombinációt választhattuk volna. A leképezések a választott kiindulási helyzetben a bal oldali legfelső rovatban ábrázolt módon Y^1-t és Z^1-t állítanak elő. Az első óraimpulzus hatására ez az Y^1 válik y^1-gyé, ami az y kombinációban nem jelent változást. Az ezután bekövetkező X^2 ezért továbbra is az állapottábla első sorában jelöli ki Y és Z kombinációk értékét. Az első sor X^2-höz tartozó rovatában Y^2 és Z^4 szerepel. A következő óraimpulzus hatására tehát y^2 jut vissza a bemenetre, ami az állapottáblán kijelöli a második sort. Az y^2 visszahatása után fellépő X^3 jelöli ki a második sor rovatát, ahol a leképezések alapján beírt új Y és Z kombinációkat találjuk. Ezek értéke Y^2 és Z^2, vagyis a következő óraimpulzus nem hoz létre y változást. Emiatt a megjelenő X^1 szintén a második sorban jelöli ki a megfelelő rovatot, amelyben Y^1 és Z^4 szerepel. Így a következő óraimpulzus y^1-t hozza létre a bemeneten. A további működés követése hasonló módon folytatható. A 3.9. ábrán nyilakkal jelöltük az adott bemeneti kombinációsorozatra bekövetkező változásokat. A szinkron működés az állapottábla alapján formálisan is megfogalmazható. Egy adott rovatból kiindulva mindig abba a rovatba jutunk, amelynek sorát a kiindulási rovatban szereplő Y kombináció, oszlopát pedig az új bemeneti kombináció jelöli ki.

Olvassuk ki, hogy a vizsgált bemeneti kombinációsorozat hatására milyen kimeneti kombinációsorozatot szolgáltat az adott szinkron sorrendi hálózat. A 3.4. ábrán bemutatott módon minden bemeneti kombinációhoz két kimeneti kombináció tartozik:

$$\begin{array}{c|c|c|c|c|c|c|}
X^1 & X^2 & X^3 & X^1 & X^3 & X^4 & X^3 \\
Z^1 Z^1 & Z^4 Z^3 & Z^2 Z^2 & Z^4 Z^1 & Z^2 Z^3 & Z^3 Z^1 & Z^2 Z^2
\end{array}.$$

Látható, hogy a hálózat által előállított kimeneti kombinációsorozat szinkron működés feltételezésével eltér az aszinkron esettől annak ellenére, hogy az állapottábla és a bemeneti kombinációsorozat változatlan volt. A szinkron működési módból következően X^4 fellépése nem okozhat oszcillációt.

A kimeneti kombinációk értelmezését megvalósító kimeneti szinkronizációnak kell kiválasztania a bemenetenkénti két kimeneti kombináció közül a további feldolgozás során figyelembe veendőt.

Ha formálisan összehasonlítjuk a vizsgált állapottáblán bejelölt állapotváltozásokat, akkor megállapíthatjuk, hogy adott méretű állapottábla esetén a szinkron „bejárási" mód általában többfajta állapotátmenetet tesz lehetővé, mint az aszinkron. Aszinkron esetben ugyanis az állapotátmenetekre szigorúbb formai követelmények érvényesek (a bemeneti változások és az instabil állapotváltozások által megszabott L alakú utak). A szinkron működésre vonatkozó szabályok viszont kevésbé rögzített bejárási utakat engednek meg.

Ez a formai összehasonlítás is alátámasztja azt a korábbi megállapításunkat, miszerint a szinkron hálózatok jobban hasznosítják az állapotokat a megoldandó logikai feladat szempontjából. A bejárási lehetőségek növelése következtében ugyanis adott számú szekunder kombináció, azaz állapot esetén többfajta állapotváltozás lehetséges, mint aszinkron esetben.

Láttuk, hogy azonos állapottábla esetén azonos bemeneti kombinációsorozatra a kimeneti kombinációsorozat eltérő lehet attól függően, hogy aszinkron vagy szinkron módon értelmezzük a működést. A későbbiekben bemutatjuk, hogy szerkeszthetők olyan állapottáblák, amelyekből ugyanazt a működést olvashatjuk ki mind aszinkron, mind szinkron esetben. Megállapodásunk értelmében a továbbiakban csak azokat az állapottáblákat értelmezhetjük aszinkron módon, amelyeknek egyetlen oszlopában sem alakulhat ki oszcilláció.

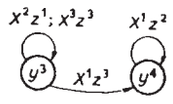
Egy állapottáblának lehetnek olyan rovatai is, amelyekben nem szerepel előírt Y és Z kombinációérték. Az ilyen rovatokhoz tartozó X és y kombinációértékekhez a leképezések (ill. a logikai feladat) nem írnak elő Y és Z kombinációkat. Ezeket a rovatokat ezért a tervezés során a lehető legcélszerűbben tölthetjük ki annak érdekében, hogy a kialakuló hálózat a lehető leggazdaságosabb legyen (lásd később). Az így megválasztható Y, ill. Z kombinációkat — a kombinációs hálózatokhoz hasonlóan — *közömbös (don't care)* állapotoknak, ill. kimeneti kombinációknak nevezzük. A közömbös állapotok és kimeneti kombinációk azért nincsenek a logikai feladatban meghatározva, mert előfordulhat, hogy bizonyos bemeneti kombinációk vagy azok sorozatai sohasem lépnek fel, vagy ha fel is lépnek, a hálózat reagálása erre tetszőleges lehet. Azokat az állapottáblákat, amelyekben közömbös állapot vagy közömbös kimeneti kombináció szerepel, *nem teljesen határozott (specifikált)* állapottábláknak nevezik. A nem teljesen határozott állapottáblák alapján tervezett sorrendi hálózatok természetesen mindig valamilyen határozott állapotot, ill. kimeneti kombinációt állítanak elő az előre nem specifikált esetekben is. (Lásd nem teljesen specifikált logikai függvények kombinációs hálózatok esetén.)

3.2.2. Az állapotgráf

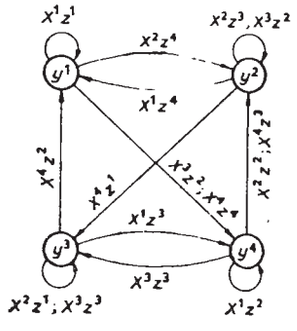
Az állapottábla megismerése után könnyen szerkeszthetünk a sorrendi hálózatok működésének leírására egy szemléltető ábrát, amelyet *állapotgráfnak* neveznek. Ennek lényege az, hogy a hálózat állapotaira jellemző y kombinációknak köröket feleltetünk meg. Az egyes bemeneti kombinációk hatására lejátszódó állapotváltozásokat az egyes köröket összekötő vagy a körökre visszakanyarodó nyilakkal ábrázoljuk. A nyilakra ráírjuk a mindenkori bemeneti és kimeneti kombinációt. Pl. a 3.10. ábrán levő állapotgráfrészletből az alábbi működés olvasható ki.

Ha a hálózat bemenetére y^3 jut, akkor akár X^2, akár X^3 hatására a szekunder változók következő kombinációja Y^3 lesz. A kimeneti kombináció pedig a Z^1, ill. Z^3 értéket veszi fel. A visszakanyarodó nyíl tehát azt jelzi, hogy Y^3 alakult ki, amely visszahatva újra y^3-t hoz majd létre a bemeneten. Ha a bemenetre y^3 jut, akkor X^1 hatására a szekunder változók következő kombinációja Y^4, a kimeneti kombináció pedig Z^3 értékű lesz. Az y^4 jelű kör felé mutató nyíl tehát azt jelzi, hogy az így kialakult Y^4 visszahatva y^4-t hoz majd létre a bemeneten. Ha a bemenetre y^4 kombináció jut, akkor X^1 hatására Y^4 és Z^2 jön létre. A visszakanyarodó nyíl tehát azt jelzi, hogy Y^4 alakul ki, amely visszahatva újra y^4-t hoz majd létre a bemeneten.

Ilyen értelmezésben a 3.11. ábrán felrajzoltuk a 3.7. ábrán szereplő állapottáblának megfelelő állapotgráfot. Könnyen beláthatjuk, hogy az állapotgráf alapján ugyanúgy követhetjük a hálózat működését — akár szinkron, akár aszinkron esetben —, mint az állapottáblán. Ahhoz, hogy az állapotgráfot egyértelműen tudjuk értelmezni, tudnunk kell, hogy a hálózat aszinkron vagy szinkron működésű-e. Természetesen szerkeszthetők olyan állapotgráfok, amelyekből ugyanazt a működést olvashatjuk ki mind aszinkron, mind szinkron esetben. Hasonlóan az állapottáblához az állapotgráfra is megállapítható az aszinkron értelmezhetőség feltétele. Eszerint csak azok az állapotgráfok értelmezhetők aszinkron működés feltételezésével, amelyekben min-



3.10. ábra. Az állapotgráf felépítésének szemléltetése



3.11. ábra. A 3.7. ábrán szereplő állapottáblának megfelelő állapotgráf

den egyes, átvezető nyílra felírt bemeneti kombináció előbb-utóbb olyan körhöz vezet, amelyhez tartozó visszakanyarodó nyílon szerepel az illető bemeneti kombináció. Mint tudjuk, ez jelenti azt, hogy egyetlen bemeneti kombináció esetén sem alakul ki oszcilláció.

Megfigyelhetjük, hogy a vizsgált állapottábla és állapotgráf Mealy-modell szerinti működést ír le. Az állapottáblán ez nagyon szemléletes, hiszen a kimeneti kombináció egy sor mentén nem azonos minden egyes oszlopban, azaz a kimeneti kombináció a bemeneti kombinációnak is függvénye. Mindkét leírási mód változtatás nélkül alkalmazható Moore-modell esetén is.

3.3. Elemi sorrendi hálózatok (flip-flopok)

A kombinációs hálózatok tervezésekor már láttuk, hogy azokat elemi kombinációs hálózatokból, vagyis kapukból építhettük fel, amelyek a mindenkori építőelemkészletben rendelkezésre álltak. Hasonló módon járhatunk el a sorrendi hálózatok esetében is. Először tehát meg kell ismernünk a leggyakrabban használt elemi sorrendi hálózatokat. Tudjuk, hogy az aszinkron sorrendi hálózatokat felépíthetjük visszacsatolt kombinációs hálózat formájában is, ezért ilyen megoldásban nincs szükségünk elemi sorrendi hálózat felhasználására. Ennek ellenére a továbbiak során olyan elemi sorrendi hálózatokat is bemutatunk, amelyek nemcsak szinkron, hanem aszinkron hálózatok felépítésére is alkalmasak. Külön kell szólnunk szinkron hálózatok esetén a szinkronizációs feltételekről. Nyilvánvaló, hogy a szinkronizációs feltételeket biztosítanunk kell azokban az elemi sorrendi hálózatokban, amelyekből szinkron sorrendi hálózatokat szeretnénk felépíteni. Az elemi hálózatok tárgyalása során a szinkronizációs feltételek biztosításának módját még nem részletezzük, a visszacsatoló ágakban az eddig alkalmazott szimbolikus jelöléseket használjuk. Mint említettük, a szinkronizációs feltételekről akkor fogunk részletesen beszélni, ha már alapjaiban megismertük az aszinkron hálózatok tervezési lépéseit.

Az elemi sorrendi hálózatok önmagukban általában igen egyszerű logikai feladatok megoldására készültek. Emiatt egyetlen szekunder változójuk van. Bemeneti változóik száma általában kettő, és egyetlen kimeneti változójuk van. Az összes ismertetésre kerülő elemi sorrendi hálózat Moore-modell szerint működik, és az $f_z'(y)$ leképezés a lehető legegyszerűbb, vagyis:

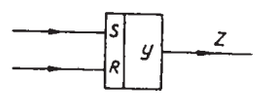
$$f_z'(y):\ Z = y$$

A kimeneti kombináció tehát azonos a szekunder kombinációnak a bemenetre visszahatott értékével. Ezért a kimenet értékét a továbbiakban egyértelműen tudjuk jellemezni, ha megadjuk az elemi sorrendi hálózat állapotát. Egyetlen szekunder vál-

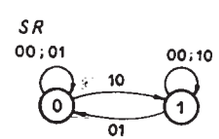
tozóval viszont csak két állapotot tudunk megkülönböztetni, ezért az elemi sorrendi hálózatokat *kétállapotú, billenő elemeknek*, vagy legelterjedtebb nevükön *flip-flopoknak* nevezik. A következőkben sorra vesszük a leggyakrabban használt flip-flop típusokat és leírjuk működésüket.

3.3.1. S—R flip-flop

Bemeneteinek elnevezése a Set (beállítás), ill. a Reset (törlés) szavak rövidítéséből származik. Definiált működése szerint az S bemenetre jutó 1 érték a flip-flop állapotát (szekunder kombinációját, azaz kimenetet) 1 értékűre állítja be. Az R bemenetre jutó 1 pedig 0 értékűre állítja be a flip-flop állapotát. Ha S és R értéke egyidejűleg 1 értékű akkor a flip-flop működése definiálatlan. Ha pedig S és R egyaránt 0, akkor a kimenet, azaz az állapot nem változik, tehát $Y=y$. A 3.12. ábrán az S—R flip-flop állapottáblája és állapotgráfja látható. Az állapottáblán az $SR=11$ rovatokba közömbös bejegyzést írhatunk, hiszen a definiálatlan működés miatt feltételezhetjük, hogy a flip-flop ilyen vezérlést nem kap. A kimeneti kombináció értékét sehol nem tüntettük fel, hiszen az a mindenkori y kombinációval azonos. Megállapíthatjuk pl. az állapottábla alapján, hogy a működés mind szinkron, mind aszinkron módon értelmezhető, hiszen egyetlen (specifikált) oszlopban sem alakul ki oszcilláció (3.13. ábra). Sőt, az is könnyen belátható, hogy jelen esetben ugyanazt a működést kapjuk akár szinkron, akár aszinkron módon, követjük az állapottáblát. Mindkét esetben ugyanis bármely bemeneti kombinációsorozatra ugyanazt a kimeneti kombinációsorozatot (azaz állapotsorozatot) kapjuk. Ebből következik, hogy az S—R flip-flop állapottáblája alapján tervezhető aszinkron hálózat. Mint láttuk, az aszinkron hálózatok mindig megépíthetők visszacsatolt kombinációs hálózattal. Építsük meg ilyen módon az S—R flip-flopot megvalósító aszinkron hálózatot. Ehhez meg kell építenünk az $f_y(X, y)=Y$ leképezést, és az Y—y összekötéssel létre kell hoznunk a



y \ SR	00	01	11	10
0	Y 0	0	–	1
1	1	0	–	1



3.12. ábra. Az S—R flip-flop működésének szemléltetése

y \ SR	00	01	11	10
0	Y 0	0	–	1
1	1	0	–	1

3.13. ábra. Stabil állapotok az S—R flip-flop aszinkron módon értelmezett állapottábláján

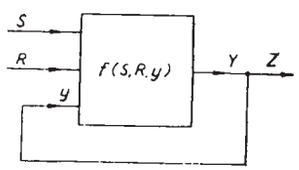
visszacsatolást. Jelen esetben az f_y leképezés egyetlen logikai függvénnyel adható meg, hiszen csak egyetlen szekunder változónk van. Így

$$f_y(X, y):\ Y = f(S, R, y).$$

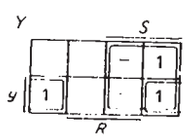
Az $f(S, R, y)$ logikai függvényt kell tehát realizálnunk kombinációs hálózattal, majd a 3.14. ábra szerint el kell végeznünk a visszacsatolást. A kombinációs hálózat realizációját könnyen elvégezhetjük a 3.12. vagy 3.13. ábra alapján, hiszen azok éppen Y értékeket — azaz a megvalósítandó függvény értékeit — tartalmazzák Karnaugh-táblaszerűen peremezve. (Az állapottábla 11 és 10 oszlopát ezért cseréltük meg a természetes bináris sorrendhez képest.) Az állapottáblát tehát a 3.15. ábrán felrajzolt Karnaugh-táblának tekinthetjük. Ebből kiolvashatjuk a legegyszerűbb diszjunktív alakot, amely hazárdmentes is:

$$Y = S + \bar{R}y.$$

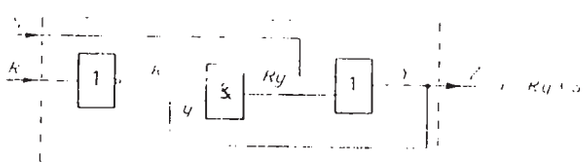
Ennek alapján felrajzolhatjuk az aszinkron működésű S—R flip-flopot megvalósító visszacsatolt kombinációs hálózatot. A 3.16. ábrán az ÉS, VAGY kapus inverteres kétszintű megoldás látható. A 3.17. ábrán bemutatott NAND kapus megvalósításban a számozott kapuk átrendezésével a 3.18. ábrán szereplő elvi logikai rajzhoz jutunk, amely az aszinkron S—R flip-flop leggyakoribb ábrázolása. A 3.18. ábra alapján könnyen beláthatjuk, hogy a flip-flop kimenetének negáltja ($\bar{y}$) a 2-es számú NAND kapu kimenetén rendelkezésünkre áll minden olyan esetben, amelyben az S és az R bemeneten nem áll fenn egyidejűleg 1. A kiinduláskor nem specifikált $SR=11$ bemeneti kombináció hatására a bemutatott megvalósításban az 1-es és a 2-es számú NAND kapuk egyaránt 1 értékű kimenetet szolgáltatnak. Emiatt a flip-flopra kapcsolódó további hálózatrészekben hibás működést okozhatunk, ha azokhoz y és $\bar{y}$ egyaránt eljut.



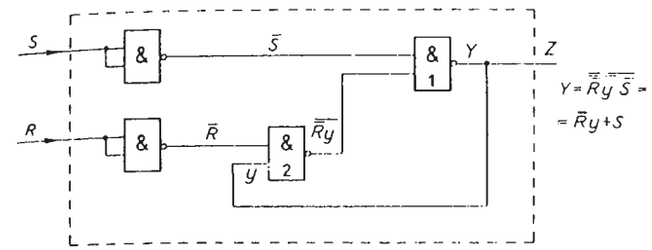
3.14. ábra. Az aszinkron S—R flip-flop visszacsatolt kombinációs hálózattal történő megvalósításának szemléltetése



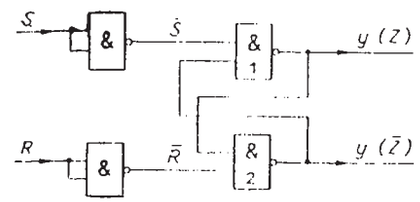
3.15. ábra. Az aszinkron S—R flip-flop megvalósításához visszacsatolandó kombinációs hálózat Karnaugh-táblája



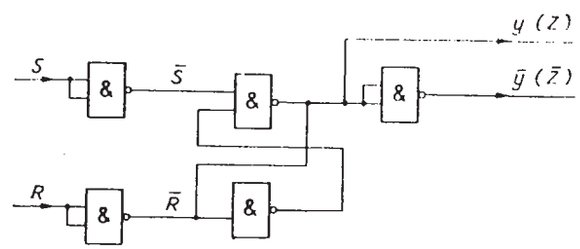
3.16. ábra. Az aszinkron S—R flip-flopot megvalósító visszacsatolt kombinációs hálózat ÉS, VAGY kapus, inverteres elvi logikai rajza



3.17. ábra. Az aszinkron S—R flip-flopot megvalósító visszacsatolt kombinációs hálózat NAND kapus elvi logikai rajza



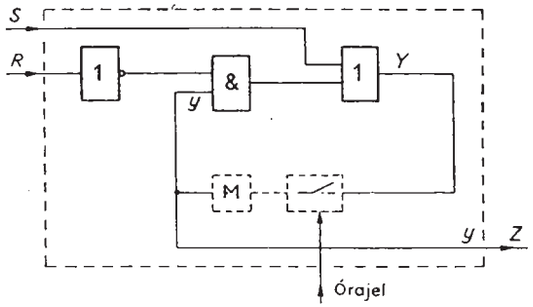
3.18. ábra. Az aszinkron S—R flip-flopot megvalósító visszacsatolt kombinációs hálózat NAND kapus elvi logikai rajzának szokásos ábrázolása



3.19. ábra. A kimeneti jel negáltjának előállítása aszinkron S—R flip-flop esetén, ha az $SR=11$ bemeneti kombináció felléphet

Ekkor ugyanis az 1-es és 2-es jelű kapu kimeneti jele egyidejűleg 1 értékű, tehát a 2-kapu kimenetén nem y, hanem $\bar{y}$ jelenik meg. A további hálózatrészek tehát tartósan fennálló $SR=11$ bemeneti kombináció mellett azt érzékelik, hogy y és $\bar{y}$ egyaránt tartósan 1 értékű, amit a logikai algebra szabályai alapján történő tervezéskor kizártnak tételezünk fel, és így a hálózat működésében hibát okozhat. (Megjegyezzük, hogy a késleltetési viszonyoktól függően természetesen egyébként is előfordulhat, hogy egy logikai hálózat valamely kapubemenetein egy jel és annak negáltja átmenetileg egyaránt 1 értékű. Erre tervezéskor mindig gondolnunk kell, mert hazárdjelenségeket okozhat.)

A vázolt hiba kiküszöbölhető, ha az $\bar{y}$ jelet a 3.19. ábra szerint állítjuk elő. Megfigyelhetjük, hogy ilyenkor $SR=11$ esetén a flip-flop kimenetének értéke 1 lesz, vagyis az S bemenetnek van elsőbbsége az R bemenethez képest.



3.20. ábra. Az S—R flip-flop szinkron megvalósításának szemléltetése

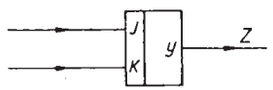
Az S—R flip-flop szinkron megvalósításához ugyanolyan lépésekben juthatunk el, mint az aszinkronhoz. A különbség csupán annyi, hogy a visszacsatoló ágba be kell iktatnunk az órajelvezérlésű kapcsolót és a tárolóelemet (3.20. ábra).

3.3.2. J—K flip-flop

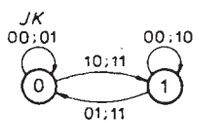
Ennek a flip-flop fajtának szintén két bemenete (J, K) és egy kimenete ($Z=y$) van (3.21. ábra). A megoldott logikai feladat csupán annyiban különbözik az S—R flip-flop esetétől, hogy a $JK=11$ kombináció hatására a flip-flop működése nem definiálatlan. Egyébként az S bemenetnek a J, az R bemenetnek pedig a K bemenet felel meg. Ha $JK=11$, akkor a J—K flip-flop megváltoztatja a mindenkori állapotát. Az állapottábla tehát csak a 11 oszlopban különbözik az R—S flip-flopétól. A 3.21. ábrán az eddigiek alapján felrajzoltuk a J—K flip-flop állapottábláját és állapotgráfját. Látható, hogy a $JK=11$ oszlopban nem jelölhető ki stabil állapot. Ebből következik, hogy a J—K flip-flop állapottáblája csak szinkron sorrendi hálózatot írhat le. Az $f_y(X, y)$ leképezésnek megfelelő $Y=f(J, K, y)$ logikai függvényt ugyanúgy olvashatjuk ki az állapottáblából, mint S—R flip-flop esetén:

$$Y = J\bar{y}+\bar{K}y+J\bar{K}.$$

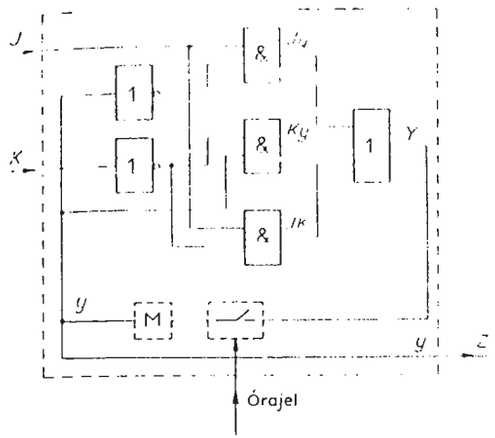
A függvényben a $J\bar{K}$ szorzat csak a hazárdmentesítés miatt szükséges. Az eddig alkalmazott szimbolikus jelölésekkel a 3.22. ábrán látható a J—K flip-flop elvi logikai rajza.



y \ JK	00	01	11	10
0	Y 0	0	1	1
1	1	0	0	1



3.21. ábra. A J—K flip-flop működésének szemléltetése



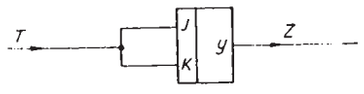
3.22. ábra. A J—K flip-flop megvalósításának szemléltetése

3.3.3. T flip-flop

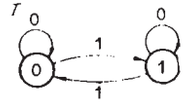
A T flip-flopot a J—K flip-flopból származtatjuk, ha a J és K bemeneteket egymással összekötöttnek képzeljük és T-vel jelöljük (3.23. ábra). Ezáltal olyan működést kapunk, mintha egy J—K flip-flopot kizárólag a 00 és 11 bemeneti kombinációkkal vezérelnénk. Így a J—K flip-flop ismeretében már megállapíthatjuk, hogy a T bemenetre érkező 0 ($JK=00$) bemeneti kombináció hatására a T flip-flop állapota nem változik, a $T=1$ ($JK=11$) bemeneti kombináció hatására pedig a T flip-flop állapota ellenkezőjére változik. Az állapottábla és állapotgráf a 3.23. ábrán látható. Természetesen a fentiek alapján $T=1$ esetén nem képzelhető el stabil állapot. Ezért a T flip-flop állapottáblája csak szinkron sorrendi hálózatot írhat le. Az állapottábla alapján könnyen felírhatjuk az $f_y(T, y)$ leképezésnek megfelelő logikai függvényt:

$$Y = \bar{T}y+T\bar{y} = T \oplus y.$$

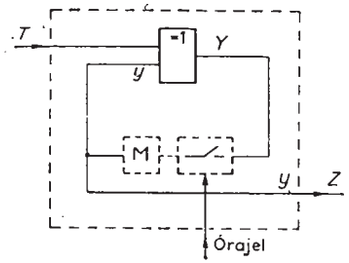
Ennek alapján a 3.24. ábrán felrajzoltuk a T flip-flop elvi logikai rajzát a visszacsatoló ágban feltételezett elemek szimbolikus jelölésével.



y \ T	0	1
0	Y 0	1
1	1	0



3.23. ábra. A T flip-flop működésének szemléltetése



3.24. ábra. A T flip-flop megvalósításának szemléltetése

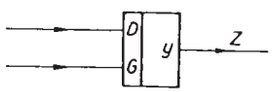
3.3.4. D—G flip-flop

Kétbemenetű flip-flop, bemeneteit D-vel és G-vel jelöljük a Data (adat) és a Gate (kapu) szavak rövidítéseként.

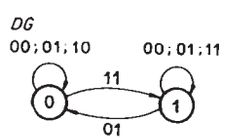
A flip-flop által megoldott logikai feladat úgy fogalmazható, hogy $G=1$ időtartama alatt a flip-flop kimenete (állapota) követi a D bemenetre jutó jelváltozásokat (vagyis $Y=D$). Ha viszont $G=0$, akkor egy újabb $G=1$-ig a flip-flop a D bemenet értékétől függetlenül megtartja a $G=0$ bekövetkezésekor éppen jelenlévő kimeneti értékét ($Y=y$). A logikai feladat alapján az állapottábla és az állapotgráf a 3.25. ábrán látható. Eszerint az állapottábla egyik oszlopában sem alakul ki oszcilláció (3.26. ábra). Az is megállapítható, hogy akár szinkron, akár aszinkron működést feltételezve olvassuk ki a működést az állapottáblából, egyaránt a D—G flip-flopra megfogalmazott logikai feladat megoldását kapjuk. Ezért a D—G flip-flop állapottáblája alapján tervezhető aszinkron hálózat. Az aszinkron D—G flip-flopot — az S—R flip-flophoz hasonlóan — realizáljuk visszacsatolt kombinációs hálózattal. Az állapottábla alapján az $f_y(D, G, y)$ leképezésnek megfelelő logikai függvény:

$$Y = DG + y\bar{G} + yD.$$

Az yD szorzat csak a hazárdmentesítés miatt szükséges.



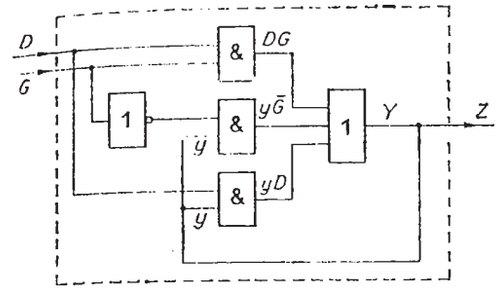
y \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1



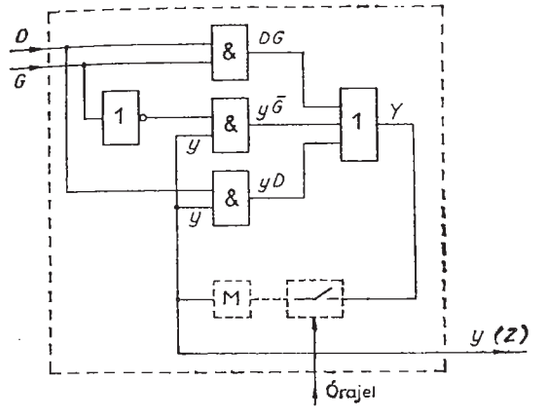
3.25. ábra. A D—G flip-flop működésének szemléltetése

y \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

3.26. ábra. Stabil állapotok a D—G flip-flop aszinkron módon értelmezett állapot-tábláján



3.27. ábra. Az aszinkron D—G flip-flopot megvalósító visszacsatolt kombinációs hálózat elvi logikai rajza



3.28. ábra. A D—G flip-flop szinkron megvalósításának szemléltetése

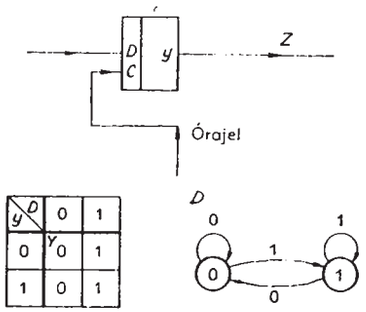
Az aszinkron D—G flip-flopot megvalósító visszacsatolt kombinációs hálózat kétszintű elvi logikai rajza az ismert módon származtatható (3.27. ábra). A szinkron megvalósítást a 3.28. ábra szemlélteti.

Az integrált áramköri építőelemek katalógusaiban az aszinkron D—G flip-flopot gyakran nevezik *latch*-nek és a G bemenetet gyakran C-vel jelölik.

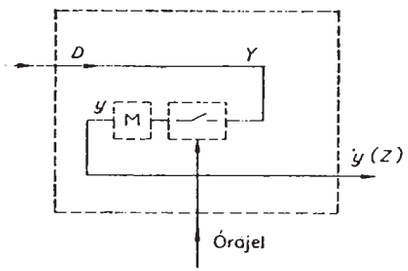
A latch elnevezés azzal magyarázható, hogy jelentése: retesz, zár és a D—G flip-flop $G=0$ esetén „reteszeli" a D bemenet értékének a kimenetre gyakorolt hatását. Gyakori félreértéseket okoz az, hogy általában D-vel és C-vel jelölik a következőkben ismertetendő ún. D flip-flop bemeneteit is. A D—G flip-flop (latch) és a D flip-flop működésében lényeges eltérések vannak, ezért igen fontos, hogy a nem teljesen következetes elnevezések ellenére mindig kellő biztonsággal meg tudjuk határozni az alkalmazott flip-flop típusát.

3.3.5. D flip-flop

Ez a flip-flop olyan elemi szinkron sorrendi hálózat, amelynek egyetlen bemenete van. A kimenet (állapot) minden egyes óraimpulzus hatására azt az értéket veszi fel, amely a bemeneten az óraimpulzus fellépésekor éppen fennáll. A flip-flop ezt az állapotát a bemeneti érték változásaitól függetlenül megtartja egy újabb óraimpulzus



3.29. ábra. A D flip-flop működésének szemléltetése



3.30. ábra. A D flip-flop szimbolikus vázlata

megjelenéséig. Az állapottábla és az állapotgráf a 3.29. ábrán látható. Az $f_y(D, y)$ leképezés jelen esetben az állapottábla alapján az

$$Y = D$$

alakban igen egyszerűen megadható, azaz y-tól független.

A D flip-flop szimbolikus vázlatát a 3.30. ábrán rajzoltuk fel. Látható, hogy a D flip-flop nem más, mint a szinkron sorrendi hálózatok visszacsatoló ágaiban feltételezett elemek tulajdonságait megvalósító hálózat. Ez egyébként a D flip-flop által megoldott logikai feladatból is következik. A D flip-flop állapottáblája formailag aszinkron módon is értelmezhető, de természetesen így nem oldaná meg az előírt logikai feladatot, sőt nem is jelentene sorrendi hálózatot, hiszen Y független y-tól, így nincs visszacsatolás. Ez egyébként abból is látszik, hogy az állapottábla két sora azonos, aminek következtében a két állapot megkülönböztetése felesleges, vagyis aszinkron értelmezéssel kombinációs hálózathoz jutunk. A szinkron megoldásban sincs természetesen visszacsatolás, de Y-ból csak a szimbolikus órajelvezérlésű kapcsoló és a tárolóelem közbeiktatásával állítható elő y, vagyis a flip-flop kimenete (állapota). Ezek a közbeiktatott elemek viszont feltételezett működésük alapján nem valósíthatók meg kombinációs hálózattal.

3.3.6. A flip-flopok működésének összefoglaló ábrázolása

A leggyakrabban használt flip-flop fajták megismerése után megállapíthatjuk, hogy aszinkron sorrendi hálózatokat az $f_y(X, y)$ leképezés alapján visszacsatolt kombinációs hálózaton kívül S—R és D—G flip-flopokból mint aszinkron építőelemekből is felépíthetünk. Ezek a flip-flopok ugyanis aszinkron működésű változatban (vagyis visszacsatolt kombinációs hálózattal) is megépíthetők.

A szinkron sorrendi hálózatok felépítéséhez mindenképpen elemi szinkron sorrendi hálózatokat (szinkron működésű flip-flopokat) kell használnunk a szinkronizációs feltételek biztosítása céljából. A továbbiakban feltételezzük, hogy a flip-flopok az építőelem-készletben rendelkezésünkre állnak. Így a szinkron hálózatok tervezésekor feltételezhetjük, hogy a szinkronizációs feltételek az elemi sorrendi hálózatokban vannak biztosítva.

Az ismertetett flip-flopok főbb tulajdonságait a 3.31. ábrán bemutatott táblázat foglalja össze. A táblázatban rendre feltüntettük az f_y leképezéseket megvalósító logikai függvények legegyszerűbb alakját hazárdmentesítő kiegészítések nélkül. A táblázat azt is tartalmazza, hogy az összes lehetséges állapotváltozást az egyes flip-flop típusok esetén rendre milyen bemeneti kombinációkkal lehet előállítani. Az állapotváltozásokat az yY kombináció jellemzi. Ennek a táblázatrésznek az értelmezését az alábbi példákon mutatjuk meg. Az S—R flip-flop esetén az $yY=01$ rovatba írt 10 bejegyzés azt jelenti, hogy a flip-flop állapotának 0-ról 1-re történő változásához $SR=10$ bemeneti kombináció szükséges. A „—" jelölés helyén akár 0 akár 1 állhat. Tehát $yY=00$ rovatban szereplő 0— bejegyzés arra utal, hogy az SR flip-flop állapota 0 marad akár $SR=00$, akár $SR=01$ kombináció jut a bemenetére. Figyeljük meg, hogy D—G flip-flop sorában az $yY=00$ rovatban szereplő $\begin{matrix}0-\\-0\end{matrix}$ bejegyzés azt jelenti, hogy a flip-flop állapota 0 marad, ha $D=0$ esetén akár $G=0$, akár $G=1$ lép fel, vagy ha $G=0$ esetén akár $D=0$, akár $D=1$ lép fel a bemeneten

Ezt a táblázatrészt előnyösen használhatjuk a későbbiekben a tervezés során

A flip-flop megnevezése	Az $f_y(X, y)$ leképezést megvalósító logikai függvény	A szükséges bemeneti kombináció, ha yY 00	01	10	11	Megjegyzés
$S-R$	$Y=S+y\bar{R}$	S R 0 –	10	01	–0	aszinkron módon is működhet
$J-K$	$Y=J\bar{y}+\bar{K}y$	J K 0 –	1 –	–1	–0	
T	$Y=\bar{T}y+T\bar{y}$	T 0	1	1	0	
$D-G$	$Y=DG+y\bar{G}$	D G 0 – –0	11	01	–0 1 –	aszinkron módon is működhet
D	$Y=D$	0	1	0	1	

3.31. ábra. A flip-flopok működésének összefoglaló táblázata

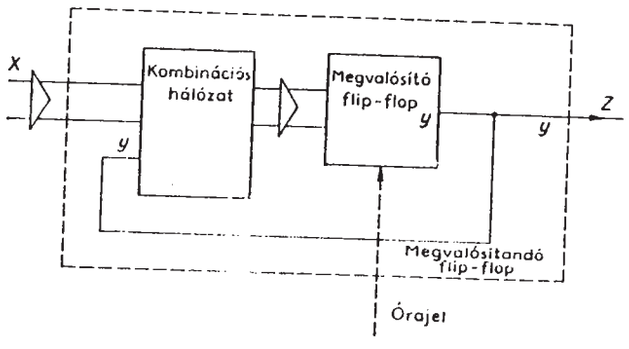
A tervezés célja — ugyanúgy, mint a kombinációs hálózatok esetében — az, hogy az elemi sorrendi hálózatok felhasználásával az adott logikai feladat megoldására a lehető legkedvezőbb tulajdonságú (pl. legkevesebb elemet tartalmazó) sorrendi hálózatot hozzuk létre.

3.3.7. A különböző flip-flop típusok felhasználása egymás megvalósítására

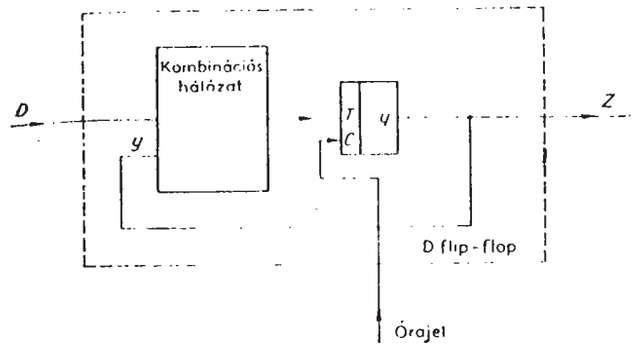
Mielőtt a sorrendi hálózatok tervezési lépéseit megismernénk, bemutatjuk, hogy az előző pontban tárgyalt flip-flopok bármelyikével az összes többi megvalósítható. A megismert flip-flop típusokon kívül természetesen még más működésűeket is definiálhatnánk formailag, de látni fogjuk, hogy nem megy az általánosság rovására, ha a továbbiakban csak a leggyakrabban használt típusokkal foglalkozunk.

A megvalósítás rendszertechnikai vázlatát a 3.32. ábra szemlélteti. A szaggatottan bekeretezett teljes hálózatban a megvalósítandó flip-flop bemeneti kombinációiból egy kombinációs hálózat állítja elő a megvalósító flip-flop bemeneti kombinációit. Ennek a kombinációs hálózatnak az a feladata, hogy úgy vezérelje a megvalósító flip-flopot, hogy a szaggatottan bekeretezett teljes hálózat a felhasználás szempontjából megegyezzék a megvalósítandó flip-flop típussal. E célból érzékeli a megvalósító flip-flop állapotát is, amit az y érték visszavezetésével biztosíthatunk. Enélkül ugyanis bizonyos flip-flop típusok esetén a feladat nem volna megoldható kombinációs hálózattal. Ugyanakkor azt is látnunk kell, hogy y értékének visszavezetése csak rajztechnikailag tűnik visszacsatolásnak. A sorrendi hálózatok elvi működésének magyarázatakor definiált visszacsatolás ugyanis mindig Y és y kombinációk között valósult meg. A 3.32. ábrán tehát — annak ellenére, hogy a megvalósítandó flip-flop természetesen sorrendi hálózat — a sorrendi működést biztosító visszacsatolást nem y visszavezetése képviseli, hanem a megvalósító flip-flopban szükségszerűen meglevő, itt nem ábrázolt Y—y összeköttetés.

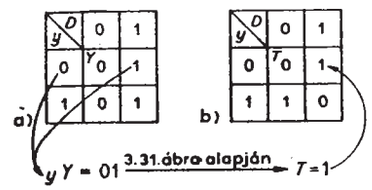
A megvalósítandó flip-flop felépítéséhez csupán a 3.32. ábrán szereplő kombinációs hálózatot kell megtervezni. E kombinációs hálózat által realizált logikai feladatot a megvalósítandó és a megvalósító flip-flop típus működésének ismeretében fogalmazhatjuk meg. Az eljárás lehetőleg egyszerű szemléltetése céljából tételezzük fel, hogy egy T flip-flop felhasználásával a D flip-flopot akarunk megvalósítani. A megoldás rendszertechnikai vázlatát a 3.33. ábra szemlélteti. A kombinációs hálózat kívánt működését táblázatosan is összefoglalhatjuk egy ún. *vezérlési táblán* (3.34*b)* ábra), amelynek fejlécei azonosak a megvalósítandó flip-flop állapottáblájának — példánkban a D flip-flopénak — fejléceivel. A különbség a rovatok értelmezésében és természetesen azok kitöltésében van. A vezérlési tábla rovataiba ugyanis nem az Y értékeket írjuk be, hanem azt a bemeneti kombinációt, amelyet a megvalósító T flip-flopra kell juttatni ahhoz, hogy a megvalósítandó D flip-flop állapottáblájának adott rovatához tartozó Y állapot létrejöjjön.



3.32. ábra. A flip-flop típusok egymás felhasználásával történő megvalósításának rendszertechnikai vázlata



3.33. ábra. D flip-flop T flip-floppal történő megvalósításának rendszertechnikai vázlata

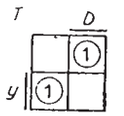


3.34. ábra. A 3.33. ábrán vázolt feladat megoldásának szemléltetése.
(*a)* a megvalósítandó flip-flop állapottáblája, *b)* a vezérlési tábla)

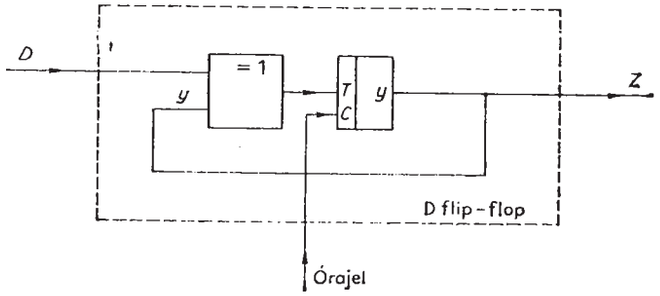
A 3.34. ábrán követhetjük a vezérlési tábla kitöltését. A könnyebb áttekinthetőség céljából a megvalósítandó D flip-flop állapottábláját is feltüntettük (3.34*a)* ábra). A vezérlési tábla rovataiba írandó T értékek meghatározásához szükségünk van egyrészt a megvalósítandó flip-flop állapotátmeneteinek a 3.31. ábrából kiolvasható feltételeire, másrészt a megvalósítandó flip-flop állapottáblájára. Az ábrán bejelöltük az egyik T érték meghatározásának menetét. A bemutatott esetben az $yY=01$ kombináció azt jelenti, hogy a megvalósítandó flip-flop állapotának 0-ról 1-re kell változnia.

A 3.31. ábrán szereplő összefoglaló táblázat T flip-flopra vonatkozó sorából kiolvashatjuk, hogy az $yY=01$ átmenethez a T flip-flop bemenetére 1 értéket kell juttatni. Ez kerül a vezérlési tábla megfelelő rovatába.

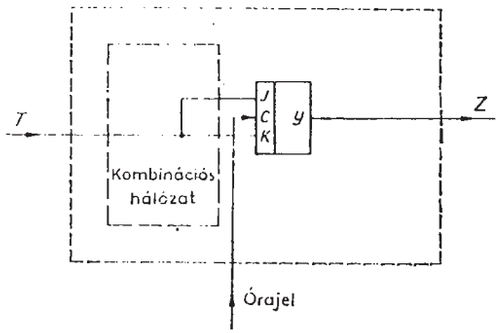
Hasonló módon töltjük ki a 3.34*b)* ábra többi rovatát is. E tábla a $T=f(D, y)$ logikai függvény Karnaugh-táblájának tekinthető (3.35. ábra), és így felírhatjuk a



3.35. ábra. A 3.33. ábrán vázolt feladatban szereplő kombinációs hálózat kimeneti függvényének ábrázolása



3.36. ábra. A 3.33. ábra rendszertechnikai vázlatának megfelelő elvi logikai rajz



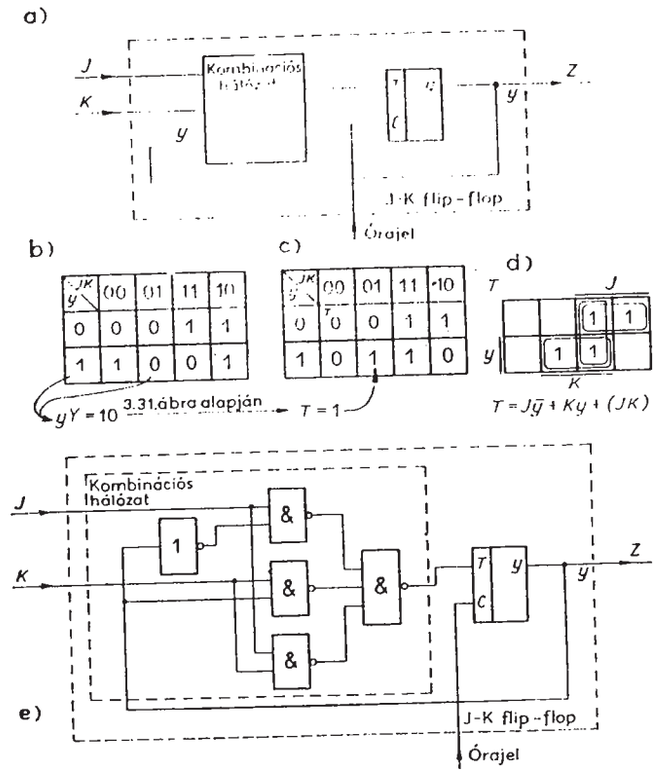
3.37. ábra. T flip-flop megvalósítása J—K flip-flop felhasználásával

3.33. ábrán szereplő kombinációs hálózat kimeneti függvényének legegyszerűbb diszjunktív alakját:

$$T = D\bar{y} + \bar{D}y = D \oplus y,$$

Ezután már felrajzolhatjuk a 3.33. ábrának megfelelő elvi logikai rajzot (3.36. ábra). A feladat megoldásához szükséges kombinációs hálózat tehát KIZÁRÓ VAGY kapcsolatot valósít meg.

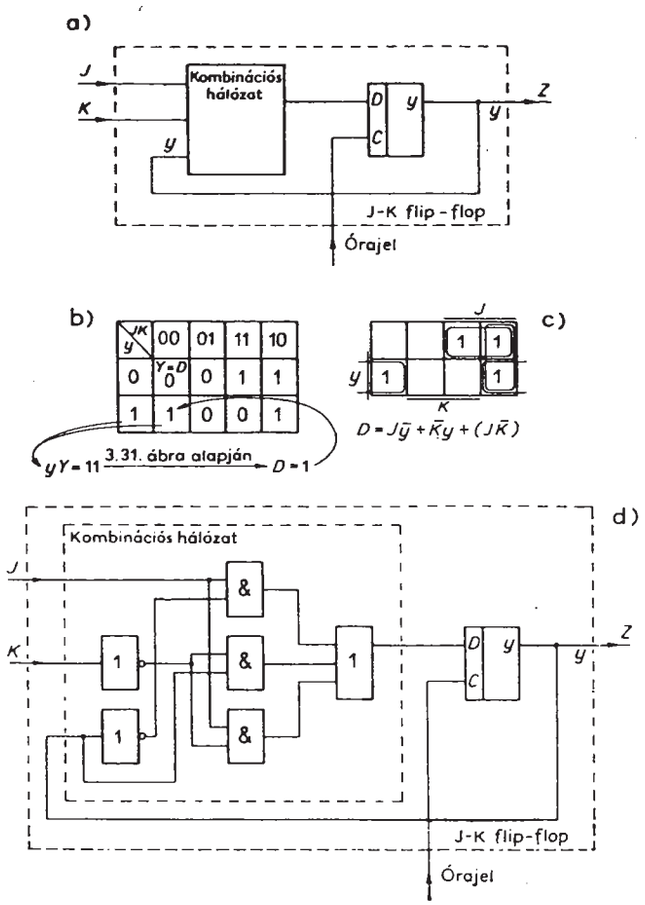
A bemutatott gondolatmenettel is eljuthatunk például ahhoz a megoldáshoz, amelyet a T flip-flop ismertetésekor említettünk a T flip-flopnak a J—K flip-flop felhasználásával történő megvalósítására. Eszerint a J és K bemeneteket összekötöttük, miáltal tulajdonképpen a J—K flip-flopot megfelelően vezérlő — ez esetben igen egyszerű — kombinációs hálózatot hoztuk létre (3.37. ábra). Ebben az esetben a két flip-flop működése olyan kevéssé tér el egymástól, hogy a szükséges kombinációs hálózatot az állapottáblákból közvetlenül meghatározhatjuk. Figyeljük meg,



3.38. ábra. J—K flip-flop megvalósítása T flip-flop felhasználásával (*a)* rendszertechnikai vázlat, *b)* a megvalósítandó flip-flop állapottáblája, *c)* vezérlési tábla, *d)* Karnaugh-tábla, *e)* elvi logikai rajz NAND kapuk felhasználásával)

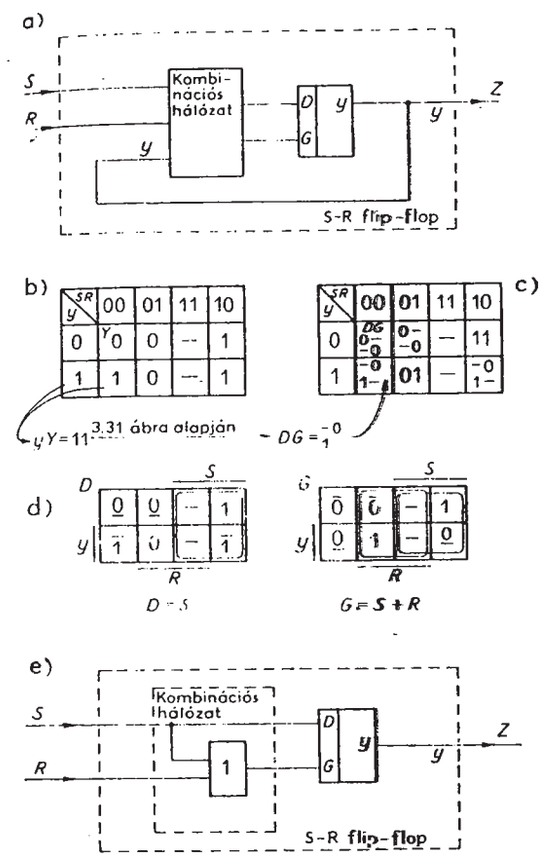
hogy ez esetben a kombinációs hálózatnak nincs szüksége a megvalósító flip-flop állapotának ismeretére, ezért hiányzik y visszavezetése.

A 3.38. ábrán követhető a J—K flip-flop megvalósítása T flip-flop felhasználásával. Ha a megvalósításhoz D flip-flopot használunk, akkor a megvalósítandó flip-flop állapottáblája egyúttal vezérlési táblának is tekinthető, hiszen $Y = D$. Láttuk, hogy a D flip-flop a szinkron sorrendi hálózatok visszacsatoló ágaiban feltételezett elemek tulajdonságaival rendelkezik. Ezért az egyes flip-flop típusok ismertetésekor a szinkron változatok előállítása (pl. 3.22. ábra) D flip-flop felhasználásával történő realizálásnak tekinthető. A 3.39. ábrán azt szemléltettük, hogy a megvalósítás bemutatott lépései szintén a 3.22. ábrán látható elvi logikai rajzhoz vezetnek. A 3.39*d)* ábra nem más, mint a 3.22. ábrán szereplő elvi logikai rajznak kismértékű átrendezése és a visszacsatolóágbeli elemek D flip-floppal történt helyettesítése révén nyert változata.

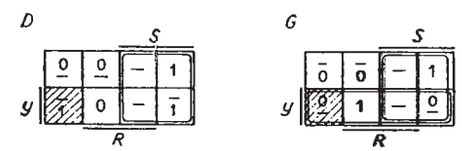


3.39. ábra. J—K flip-flop megvalósítása D flip-flop felhasználásával *(a)* rendszertechnikai vázlat, *b)* a megvalósítandó flip-flop állapottáblája, amely ebben az esetben egyúttal vezérlési tábla, *c)* Karnaugh-tábla, *d)* elvi logikai rajz)

A 3.40. ábrán aszinkron flip-flopokra mutatjuk be az egymás felhasználásával történő megvalósítás lépéseit. A 3.40*e)* ábrán láthatjuk, hogy az adott esetben nincs szükség *y* visszavezetésére. Ez a példa azt is szemlélteti, hogy D—G flip-flop felhasználása esetén a vezérlési táblából kiolvasható Karnaugh-táblákon végzett összevonáskor a *D* és *G* függvények közömbös értékei nem mindig rögzíthetők egymástól függetlenül. Ügyelni kell ugyanis arra, hogy a két bejegyzést tartalmazó rovatokban mindkét Karnaugh-táblán egyaránt a felső vagy egyaránt az alsó bejegyzések szerint végezzük az összevonást. Ellenkező esetben hibás *DG* kombináció keletkezhet. Ha



3.40. ábra. Aszinkron S—R flip-flop megvalósítása aszinkron D—G flip-flop felhasználásával *(a)* rendszertechnikai vázlat, *b)* a megvalósítandó flip-flop állapottáblája, *c)* vezérlési tábla, *d)* Karnaugh-táblák, *e)* elvi logikai rajz)



3.41. ábra. Hibás összevonás szemléltetése D—G flip-flop esetén

például a 3.40*c)* ábrán a vezérlési tábla $SRy=001$ rovatában szerepő $\begin{matrix}-0\\1-\end{matrix}$ bejegyzés alapján a Karnaugh-táblákon $D=0$-át és $G=1$-et rögzítenénk a 3.41. ábrán vázolt módon, akkor a vonalkázott rovatoknak megfelelő állapotváltozás hibás lenne. Az előírt $Y=1$ helyett ugyanis $DG=01$ miatt $Y=0$ alakulna ki. D—G flip-flop-pal történő megvalósítás esetén tehát a *D* és *G* függvények legegyszerűbb alakját általában csak próbálgatással tudjuk meghatározni. Ha egymástól függetlenül végezzük a *D* és *G* függvények minimalizálását, akkor mindig ellenőriznünk kell, hogy a közömbös értékek kiadódott rögzítése nem okoz-e hibás *DG* kombinációt és szükség esetén módosítani kell az összevonást. Ez a pótlólagos megfontolás termé-

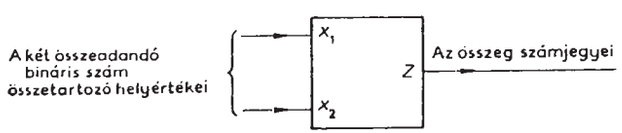
szetesen a D—G flip-flop definiált működéséből következik és a többi megismert flip-flop típus esetén nem szükséges.

A különböző flip-flop típusokkal való megvalósítás gondolatmenetének ismeretében egy adott logikai feladatot megoldó sorrendi hálózat tervezési lépései egyszerűen bemutathatók. A tervezés célja ilyenkor is az, hogy elemi sorrendi hálózatokból — azaz flip-flopokból — építsük fel az előírt működésű sorrendi hálózatot. A megvalósítandó sorrendi hálózat az eddigiekben a megvalósítandó flip-flop volt és a választott megvalósító flip-flop felhasználásával építettük azt fel, vagyis tulajdonképpen igen egyszerű esetre a tervezés lépéseit mutattuk be. Az egyszerűséget nemcsak az okozta, hogy a megvalósítandó hálózat is elemi sorrendi hálózat volt, hanem az is, hogy ebből következően kiinduláskor adott volt az állapottáblája. Általában ugyanis az is a tervezési lépésekhez tartozik, hogy a megvalósítandó sorrendi hálózat által megoldandó logikai feladat valamilyen megfogalmazása alapján létrehozzuk az állapottáblát.

3.4. Szinkron sorrendi hálózatok tervezési lépéseinek bemutatása egy egyszerű példán

A logikai feladat megfogalmazása

Tervezzünk olyan szinkron sorrendi hálózatot, amelynek két bemenete és egy kimenete van, és megvalósítja a soros bináris összegzést. A két bemenetre megfelelő ütemezéssel érkeznek a két összeadandó bináris szám egymás utáni számjegyei a legkisebb helyértéktől kezdődően. A tervezendő hálózat feladata, hogy ugyanilyen ütemezéssel szolgáltassa a kimeneten az összeg egymás utáni bináris számjegyeit. Az egyes helyértékeken levő számjegyek összeadásakor a hálózatnak természetesen figyelembe kell venni az előző helyértéken keletkezett átvitelt (3.42. ábra).



3.42. ábra. A soros összegző működésének szöveges megfogalmazását szemléltető vázlat

Az előzetes állapottábla összeállítása

A logikai feladat szöveges megfogalmazása alapján nyilvánvaló, hogy a hálózatnak különbözőképpen kell reagálnia ugyanarra a bemeneti kombinációra a megelőző bemeneti kombinációktól, ill. a hálózat állapotától függően. A feladat tehát csakis sorrendi hálózattal oldható meg. A szöveges megfogalmazásból nem mindig derül ki

egyértelműen, hogy a hálózatnak minimálisan hány állapottal kell rendelkeznie. Ezért általában a szöveges megfogalmazás alapján inkább több állapotot különböztetünk meg, mint ahány feltétlenül szükséges és ezek alapján töltjük ki az állapottáblát. Az így kapott ún. *előzetes állapottáblában* tehát általában több állapot szerepel a szükségesnél.

Példánkban az egyes bemeneti kombinációk fellépésekor a hálózat állapotát jellemezhetjük

— a kimenet pillanatnyi értékével,

— az előző helyértéken keletkezett átvitel értékével.

Könnyen belátható, hogy soros összegzéskor az egyes bemeneti kombinációk hatására létrejövő kimeneti érték nem függ a pillanatnyi kimeneti értéktől, hanem csak az előző helyértéken keletkezett átviteltől. Ezért csak ettől függően kellene a hálózat állapotait megkülönböztetnünk. Az ilyen döntés bonyolultabb feladatok esetén a szöveges megfogalmazás alapján természetesen nem mindig ilyen egyszerű. Olyan módszerre van tehát szükségünk, amelynek segítségével mindig eljutunk a lehető legkevesebb megkülönböztetendő állapothoz. Ennek szemléltetéséhez példánkban különböztessük meg az állapotokat a kimenet pillanatnyi értéke szerint is, jóllehet — mint az egyszerű megfontolások alapján is beláthatjuk — erre nincs szükség. Jelöljük az egyes állapotoknak megfelelő szekunder kombinációkat az ábécé kisbetűivel:

Az állapot jele	Az előző helyértéken keletkezett átvitel értéke	A kimenet pillanatnyi értéke
a	0	0
b	0	1
c	1	0
d	1	1

Az előzetes állapottáblának tehát 4 sora lesz, az oszlopok száma szintén 4, mivel a hálózatnak két bemenete van. Az állapotokra bevezetett jelölésekkel az előzetes állapottábla a 3.43. ábrán látható. Az állapottábla kitöltését a következőképpen követhetjük.

Tételezzük fel, hogy a vizsgálat kezdetekor az *a* állapotnak megfelelő *y* kombináció és a 00 bemeneti kombináció van a 3.4. ábra szerint működő hálózat bemenetén. Ilyenkor a logikai feladat szerint a kimenet 0 értékű és átvitel nem keletkezik. Ehhez a helyzethez az *a* állapotot rendeltük hozzá, tehát az *Y* kombináció is *a*-nak megfelelő lesz. Ezért írtunk a bal oldali legfelső rovatba *a* 0-át. Ha az *y* kombináció

y \ x_1x_2	00	01	11	10
a	*a* 0	*b* 1	*c* 0	*b* 1
b	*a* 0	*b* 1	*c* 0	*b* 1
c	*b* 1	*c* 0	*d* 1	*c* 0
d	*b* 1	*c* 0	*d* 1	*c* 0

3.43. ábra. A soros összegző előzetes állapottáblája

a-nak felel meg, a bemeneti kombináció pedig 01, akkor $0+1=1$ miatt az Y kombinációnak a b állapotot kell kijelölnie 1 kimeneti értékkel. Ezért szerepel az állapottábla első sorának második rovatában $b1$. Hasonló gondolatmenettel magyarázható a többi rovat bejegyzése is. Például, ha c-nek megfelelő y kombináció és 11 bemeneti kombináció áll fenn a hálózat bemenetén, akkor a bináris összeg értéke — a c állapot által jelzett átvitel figyelembevételével — 1 lesz és átvitel is keletkezik ($1+1+1=$ $=1\rightarrow$átv.). Emiatt került az állapottábla harmadik sorának harmadik rovatába a $d1$ bejegyzés.

Figyeljük meg, hogy az adott példában a kimenet pillanatnyi értéke az egyes rovatok mindkét bejegyzését befolyásolja. Hiszen az állapot jele függ a kimenet értékétől, a második bejegyzés pedig maga a kimenet értéke. Ezért van az, hogy az egyes rovatokban csak az $a0$, $c0$, $b1$ és $d1$ bejegyzéspárok szerepelnek.

Látható, hogy az állapottáblán az a és b sorokban bemeneti kombinációnként azonos bejegyzések szerepelnek. Hasonló a helyzet a c és d sorokban is. Ez természetes, hiszen a-t b-től, valamint c-t d-től csak a kimenet pillanatnyi értéke különbözteti meg. Jelen egyszerű példánkban viszont már a feladat szöveges megfogalmazásából is láttuk, hogy a kimenetek pillanatnyi értéke szerint nem szükséges megkülönböztetni a hálózat állapotait. A feleslegesen megkülönböztetett állapotok következtében tehát az állapottáblán azonos sorok keletkeztek.

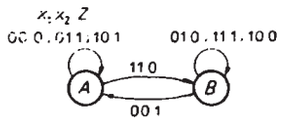
A felvett előzetes állapottábláról könnyen megállapíthatjuk, hogy a tervezendő hálózat Mealy-modell szerint működik. A kimenet értéke ugyanis ugyanazon y kombináció esetén a bemeneti kombinációtól függően változik, azaz egy sor mentén változó értékű. A Moore-modell szerint működő hálózat előzetes állapottábláját a tervezési lépések bemutatása után vizsgáljuk meg.

Az egyszerűsített (összevont) állapottábla

Az előzetes állapottábla felvétele után célunk, hogy megkeressük a feleslegesen megkülönböztetett állapotpárokat és meghatározzuk a lehető legkevesebb megkülönböztetendő állapotot. Ezek ismeretében létrehozhatjuk a lehető legkevesebb sort tartalmazó állapottáblát, amelyet *egyszerűsített (összevont) állapottáblának* nevezünk.

Példánkban feleslegesen különböztettük meg a-t b-től és c-t d-től. Ezért a továbbiakban jelöljük a és b állapotokat egyaránt A-val, c és d állapotokat pedig egyaránt B-

y \ x_1x_2	00	01	11	10
A	$A0$	$A1$	$B0$	$A1$
B	$A1$	$B0$	$B1$	$B0$



3.44. ábra. A soros összegző egyszerűsített (összevont) állapottáblája és állapotgráfja

vel. Az így bevezetett jelölésekkel az előzetes állapottábla formálisan átalakítható a 3.44. ábrán látható kétsoros állapottáblává. Ez már nem tartalmaz feleslegesen megkülönböztetett állapotokat, ezért az összevont állapottáblához jutottunk. Az állapotgráf szintén a 3.44. ábrán látható.

Állapotkódolás

Miután rendelkezésünkre áll az összevont állapottábla, az abban szimbolikusan jelölt állapotok mindegyikéhez hozzá kell rendelnünk egy-egy szekunder kombinációt. Az összevont állapottábla állapotainak (sorainak) számától függ, hogy ehhez hány szekunder változó szükséges. Példánkban, mivel két állapotot (sort) kell megkülönböztetnünk, elegendő egyetlen szekunder változót felvennünk. Válasszuk meg az állapotoknak megfelelő szekunder változó értékeket, az ún. *állapotkódokat* az alábbi módon:

az állapot A, ha $y=0$;
az állapot B, ha $y=1$.

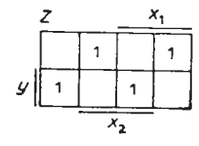
Az adott esetben az állapotkód akkor 0, ha az átvitel 0 értékű, és akkor 1, ha az átvitel 1 értékű. Ez a „természetes" kódolás kézenfekvő, de nyilvánvalóan fordítva is választhattuk volna a kódot. Sőt, általában több soros állapottáblák esetén (több szekunder változó esetén) sok egymástól különböző módon rendelhetünk szekunder kombinációkat az egyes állapotokhoz. Ilyenkor tehát sokféle módon lehet elvégezni az állapotkódolást. Mint a későbbiekben látni fogjuk, a kialakuló hálózat egyszerűsége szempontjából egyáltalán nem közömbös, hogy melyik állapotkódolást választjuk a sok lehetséges közül.

Példánkban a választott kódolással a 3.45. ábrán felrajzoltuk az ún. *kódolt állapottáblát*. Ennek rovatait könnyen kitölthetjük, hiszen az állapotok szimbolikus jelöléseit kell csupán a választott kód szerinti Y értékekkel helyettesítenünk.

A kódolt állapottábla alapján felírhatjuk az $f_z(X, y)$ leképezésnek megfelelő logikai függvényt. Jelen esetben ugyanis az összevont állapottábla fejléceinek kitöltése éppen a Karnaugh-tábla peremezésének felel meg (3.46. ábra). Ennek alapján az $f_z(X, y)$

y \ x_1x_2	00	01	11	10
0	$Y\,Z$ 00	01	10	01
1	01	10	11	10

3.45. ábra. A soros összegző kódolt állapottáblája



3.46. ábra. A soros összegző $f_z(X, y) \Rightarrow Z$ leképezését megvalósító függvény Karnaugh-táblája

leképezést az alábbi logikai függvénnyel adhatjuk meg:

$$Z = \bar{x}_1\bar{x}_2 y + \bar{x}_1 x_2 \bar{y} + x_1 x_2 y + x_1 \bar{x}_2 \bar{y} = S^3_{1,3}(x_1, x_2, y),$$

vagyis Z-re szimmetrikus függvényt kaptunk. Ezek után már csak az $f_y(X, y)$ leképezést kell megvalósítanunk a megismert flip-flopok felhasználásával.

A vezérlési tábla összeállítása

A tervezés következő lépésében minden egyes szekunder változóhoz hozzárendelünk egy flip-flopot. A flip-flopok működésének ismeretében meghatározhatjuk, hogy miként kell működtetni, vezérelni azokat, hogy az általuk megvalósított szekunder változók a kódolt állapottábla szerint változzanak. A flip-flopoknak e vezérlési előírásait a már megismert vezérlési táblán ábrázoljuk. Egyszerű példánkban — lévén csupán egyetlen szekunder változó — csak egy flip-flopot kell alkalmaznunk.

A flip-flop típusának megválasztását legtöbbször az befolyásolja, hogy az alkalmazható építőelem-készletben melyik áll rendelkezésünkre. A megfelelő vezérléshez kiadódó kombinációs hálózat egyszerűsége azonban erősen függhet a választott flip-flop típustól. Ezt a 3.3.7. pontban is megfigyelhettük. Így ha módunkban áll, célszerű többfajta flip-flop választásának hatását megvizsgálni, amit általában csak próbálgatással végezhetünk el. Több szekunder változó esetén természetesen nem szükségszerű, hogy mindegyikhez azonos típusú flip-flopot válasszunk.

Példánkban az egyetlen szekunder változó megvalósításához válasszunk először S—R flip-flopot. A vezérlési tábla kitöltését a 3.47. ábrán követhetjük. A fejlécek a kódolt állapottáblához képest változatlanok. Az egyes rovatokba azokat az SR kombinációkat írjuk, amelyek a kódolt állapottábla azonos rovatában szereplő Y érték létrehozásához szükségesek. A bal oldali legfelső rovatba pl. azért került $SR=0-$ beírás, mert a kódolt állapottáblában ehhez a rovathoz $yY=00$ kombináció tartozik, amelynek előidézéséhez $SR=00$ vagy $SR=01$ kombináció szükséges (R értéke tehát közömbös). Ez kiolvasható a 3.31. ábrán levő összefoglaló táblázatból is. Azonos a gondolatmenet a többi rovat kitöltésekor is. Például az 11 oszlop $y=0$-hoz tartozó rovatában azért szerepel $SR=10$, mert a kódolt állapottáblának ebben a rovatában $yY=01$, vagyis a flip-flop állapotának 1-re kell változnia, amit az $SR=10$ kombinációval lehet előidézni.

A vezérlési tábla ismeretében már megvalósíthatjuk azt a vezérlő kombinációs hálózatot, amely megfelelően vezérli a flip-flopot ahhoz, hogy az előírt $f_y(X, y)$ leképzés megvalósuljon.

y \ x_1x_2	00	01	11	10
0	SR 0 -	0 -	1 0	0 -
1	0 1	- 0	- 0	- 0

3.47. ábra. A soros összegző vezérlési táblája S—R flip-floppal történő megvalósítás esetén

A vezérlési tábla tulajdonképpen tartalmazza az

$$S = f_S(x_1, x_2, y),$$

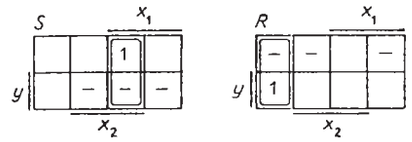
$$R = f_R(x_1, x_2, y)$$

függvények Karnaugh-tábláit, amelyeket a 3.48. ábrán rajzoltunk fel. Eszerint

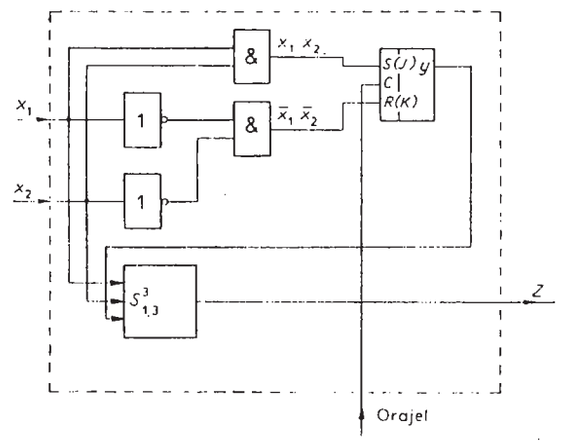
$$S = x_1 x_2; \quad R = \bar{x}_1 \bar{x}_2.$$

A függvények alapján már felrajzolhatjuk a sorrendi hálózat elvi logikai rajzát (3.49. ábra), és ezzel a tervezést befejezettnek tekintjük. A logikai rajzon az órajelet is bejelöltük, ami arra utal, hogy szinkron flip-flopról, ill. szinkron sorrendi hálózatról van szó. Az S és R bemenetek jelei után a J és K jeleket is feltüntettük, mert jelen esetben ugyanilyen hálózat adódik, ha J—K flip-flopot alkalmazunk. Ha J—K flip-flopot használunk, akkor a már ismert módszerrel felírt vezérlési tábla a 3.50. ábra szerinti. Látható, hogy az újabb közömbös J és K értékek éppen olyan helyeken adódtak, ahol a vezérlő kombinációs hálózat legegyszerűbb diszjunktív alakját nem befolyásolják.

Vizsgáljuk meg ezek után, milyen hálózatot kapunk, ha a többi megismert flip-flop fajtát használjuk.



3.48. ábra. A soros összegző S—R flip-floppal történő megvalósításához szükséges S és R függvények Karnaugh-táblái



3.49. ábra. A soros összegző elvi logikai rajza S—R vagy J—K flip-floppal történő megvalósítás esetén

y \ x_1x_2	00	01	11	10
0	JK 0 -	0 -	1 -	0 -
1	- 1	- 0	- 0	- 0

3.50. ábra. A soros összegző vezérlési táblája J—K flip-floppal történő megvalósítás esetén

T flip-flop esetén a vezérlési táblát a 3.51. ábrán láthatjuk. Ennek alapján:

$$T = \bar{x}_1\bar{x}_2 y + x_1 x_2 \bar{y}.$$

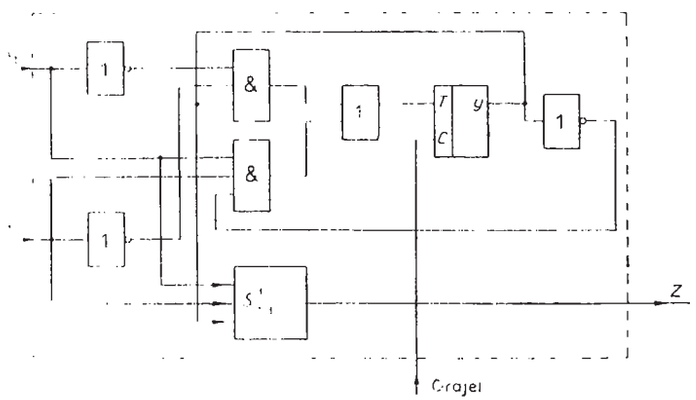
A hálózat elvi logikai rajza a 3.52. ábrán látható. A vezérlő kombinációs hálózat bonyolultabb, mint S—R vagy J—K flip-flop esetén.

Ha D—G flip-flopot választunk, akkor 3.53. ábra szerinti vezérlési tábla adódik. A D—G flip-flop működésének ismeretében könnyen beláthatjuk, hogy néhány rovatban szükségképpen adódott több választási lehetőségünk a *DG* kombinációkra. Azokban a rovatokban, amelyekben két sorban adtuk meg a *DG* értékeket, természetesen a két sor közül bármelyiket választhatjuk a realizációhoz, de — mint tudjuk — a *D* és *G* értékeket ugyanazon rovatban nem választhatjuk felváltva a két sorból. Az összevonás egy megoldása a 3.54. ábrán látható. Eszerint:

$$D = x_2 + x_1 y; \qquad G = x_1 + y.$$

$y \backslash x_1x_2$	00	01	11	10
0	T 0	0	1	0
1	1	0	0	0

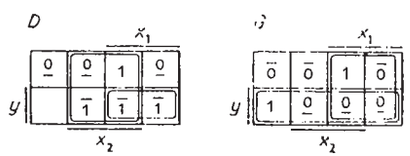
3.51. ábra. A soros összegző vezérlési táblája T flip-floppal történő megvalósítás esetén



3.52. ábra. A soros összegző elvi logikai rajza T flip-floppal történő megvalósítás esetén

$y \backslash x_1x_2$	00	01	11	10
0	D G 0 0	0 – – 0	11	0 – – 0
1	01	– 0 1 –	– 0 1 –	– 0 1 –

3.53. ábra. A soros összegző vezérlési táblája D—G flip-floppal történő megvalósítás esetén



3.54. ábra. A soros összegző D—G flip-floppal történő megvalósításához szükséges *D* és *G* függvények Karnaugh-táblái

A két Karnaugh-táblán végzett összevonások kölcsönhatását jól szemlélteti az x_1y prímimplikáns szükségessége. A *G* táblán képzett y és x_1 primimplikánsok ugyanis az $x_1\bar{x}_2y$ rovatban 1-et tételeznek fel. Ezt az adott esetben csak a rovatban levő alsó (közömbös) bejegyzés 1-re rögzítésével érhetjük el. Így viszont a *D* táblán is az alsó bejegyzést tettük érvényessé ebben a rovatban, ami 1 értékű, tehát lefedést igényel. Ha nem valósítanánk meg az x_1y prímimplikánssal ezt a lefedést, akkor a vizsgált rovatnak megfelelő *DG* kombináció 01 értékű lenne, ami a flip-flop állapotát 0-ra állítaná be, vagyis $Y=0$-t hozna létre, ellentétben a kódolt állapottábla $Y=1$ előírásával. Próbálgatással még más hasonló lefedési megoldásokhoz juthatunk. A választott megoldáshoz tartozó elvi logikai rajz a 3.55. ábrán látható. A 3.3.7. pontban már bemutattuk, hogy a próbálgatásra a D—G flip-flop definiált működése miatt van szükség. A többi megismert flip-flop típus alkalmazásakor nem lép fel a vázolt kölcsönhatás a bemenetek értékei között.

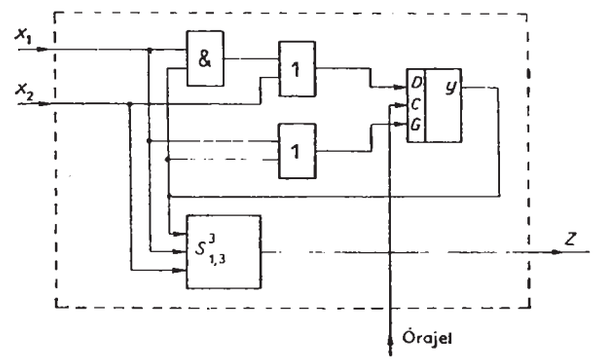
D flip-flop választása esetén a vezérlési tábla a 3.56. ábrán látható. Megállapíthatjuk, hogy *D* értéke minden egyes rovatban azonos a kódolt állapottábla ugyanazon rovatában szereplő *Y* értékkel. Ez természetes, hiszen D flip-flop esetén $Y=D$. Így a vezérlő kombinációs hálózatot az $f_y(X, y)$ leképezésnek megfelelő

$$Y = f(x_1, x_2, y)$$

függvény írja le. Formailag tehát visszacsatolt kombinációs hálózatot kell terveznünk. Ez is természetes, hiszen a D flip-flop működése, mint láttuk, a szinkron sorrendi hálózatok visszacsatoló ágainak feltételezett működésével azonos. A vezérlési tábla alapján tehát felírhatjuk:

$$Y = D = x_1x_2 + x_1y + x_2y = S^3_{2,3}(x_1, x_2, y).$$

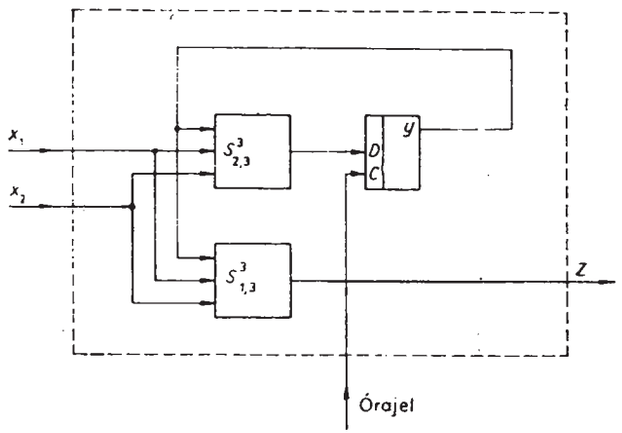
A függvény jelen esetben szimmetrikus. Az elvi logikai rajz a 3.57. ábrán látható.



3.55. ábra. A soros összegző elvi logikai rajza D—G flip-floppal történő megvalósítás esetén

$y \backslash x_1x_2$	00	01	11	10
0	D 0	0	1	0
1	0	1	1	1

3.56. ábra. A soros összegző vezérlési táblája D flip-floppal történő megvalósítás esetén

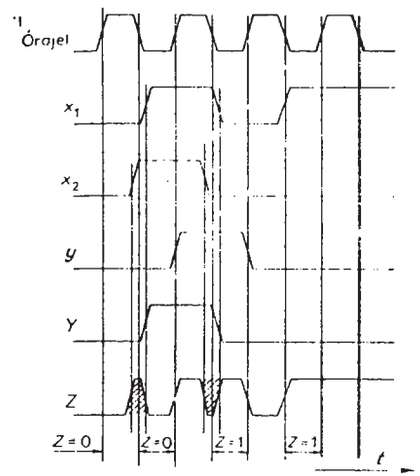


3.57. ábra. A soros összegző elvi logikai rajza D flip-floppal történő megvalósítás esetén

A különböző flip-flop típusokból felépített elvi logikai rajzokhoz természetesen úgy is eljuthattunk volna, hogy a megvalósítást elvégezzük valamelyik flip-flop típus felhasználásával, és ezt flip-flop típust rendre megvalósítjuk a többi flip-floppal a 3.3.7. pontban megismert módon. Ilyen megoldás esetén a flip-flopokat vezérlő kombinációs hálózatot tulajdonképpen két részből állítjuk össze: az egyik rész a kiindulási flip-flop vezérlését végzi, a másik pedig a kiindulási flip-flopot megvalósító flip-flopot. Könnyen beláthatjuk, hogy a két részletben történő megvalósítás a kombinációs hálózat egyszerűsítése szempontjából kedvezőtlenebb.

A működés szemléltetése idődiagrammal

A különböző típusú flip-flopok felhasználásával kialakított szinkron sorrendi hálózatok mindegyike természetesen csak akkor oldja meg az adott logikai feladatot, ha a kimeneti értékek értelmezési időtartományát megfelelően jelöljük ki, vagyis betartjuk az erre vonatkozó szinkronizációs feltételt. Ennek szükségességét jól szemléltethetjük a hálózat működésének időbeli ábrázolásával. A 3.58. ábrán látható idődiagram az $x_1x_2=00—11—00—10$ bemeneti kombinációsorozat hatására lejátszódó változásokat szemlélteti azzal a feltételezéssel, hogy az első vizsgált óraimpulzus megjelenésekor $y=0$ és a bemeneti változások az óraimpulzus lefutó éleinek hatására történnek. Az idődiagram felépítése a 3.4. ábrán szemléltetett általános szinkron működésnek felel meg. Az ábrázolt jelek trapéz alakja azt szemlélteti, hogy a felfutó és lefutó élek a valóságban mindig véges meredekségűek, vagyis a jelváltozásoknak csak egy időtartománya adható meg időpont helyett. Ez a kimenetek értelmezési tartományának kijelölésében további nehézséget okoz nem szomszédos bemeneti változások esetén. Nem feltételezhetjük ugyanis, hogy a nem szomszédos bemeneti változások során a bemeneti jelek egyszerre változnak. Ennek szemléltetésére az idődiagramon az $x_1x_2=00\rightarrow 11$ bemeneti kombinációváltozást például az $x_1x_2=01$ kombináció közbeiktatásával ábrázoltuk. Ebben az esetben az átmeneti



3.58. ábra. A soros összegző működésének szemléltetése idődiagrammal

$x_1x_2=01$ kombináció hatása Z értékében is jelentkezik, és a vonalkázott bizonytalan szélességű impulzus jön létre, amelyet a vizsgált bemeneti kombinációsorozatra adott kimeneti válaszok között nem szabad figyelembe vennünk. Hasonló a helyzet az $x_1x_2=11\rightarrow 00$ változáskor, amikor a 3.58. ábra szerint átmenetileg létrejön az $x_1x_2=10$ kombináció. A bemutatott idődiagramon az egyéb jelterjedési késleltetések hatásait elhanyagoltuk, de a függőleges vonalak azokat az éleket kötik össze, amelyek által jelölt változások között ok—okozati összefüggés áll fenn. Megfigyelhetjük, hogy az adott változási sorrend mellett akkor kapjuk az előírt feladatot megoldó kimeneti jelsorozatot, ha a kimenet értékét csak az óraimpulzus-szünetekben értelmezzük. A bináris összeadás szabályából következő kimeneti jelsorozat ugyanis a vizsgált bemeneti kombinációsorozat hatására: $Z=0, 0, 1, 1$. Ezt például úgy érhetjük el, hogy a hálózat által szolgáltatott kimeneti jelet ÉS kapcsolatba hozzuk az órajel negáltjával. A 3.58. ábrán bejelöltük az így kialakuló értelmezési tartományokat a kimenet várt értékeire vonatkozóan. Látható, hogy ily módon nem küszöbölhetjük ki a vonalkázott átmeneti értékek esetleges zavaró hatását. Ehhez általában pótlólagos flip-flopokra van szükség, amelyek Moore-modell szerint működő hálózatokban a tervezés során eleve kiadódnak.

Moore-modell tervezése

Tervezzünk Moore-modell szerint működő szinkron sorrendi hálózatot a korábbi példánkban szereplő soros összegző megvalósítására. Az előzetes állapottábla felvételéhez a logikai feladat alapján különböztessük meg ugyanazokat az állapotokat, mint Mealy-modell esetén. Az *a*, *b*, *c* és *d* állapotok jelentése tehát változatlan.

Az előzetes állapottábla kitöltésekor mindig szem előtt kell tartanunk, hogy $f'_z(y)$ alakú leképezést kell megvalósítanunk, vagyis egy bemeneti kombináció hatása

y \ x_1x_2	00	01	11	10
a	a 0	b 0	c 0	b 0
b	a 1	b 1	c 1	b 1
c	b 0	c 0	d 0	c 0
d	b 1	c 1	d 1	c 1

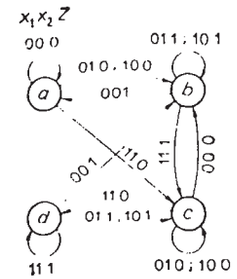
3.59. ábra. Moore-modell szerint működő soros összegző előzetes állapottáblája

csak azután fog érvényesülni a kimeneten, miután az általa létrehozott Y visszahatott a bemenetre y-ként. Ennek figyelembevételével 3.59. ábra alapján az alábbi gondolatmenettel követhetjük az előzetes állapottábla kitöltését.

Tegyük fel, hogy vizsgálatunk kezdetekor a 3.6. ábra szerint működő hálózatban az y kombináció az a állapotnak felel meg, a bemeneti kombináció pedig $x_1x_2=00$. Ekkor a kimenetnek 0 értékűnek kell lennie és átvitel nem keletkezik. Ezért kell a bal oldali legfelső rovatba a 0-t írnunk. Az állapottábla első sorának kitöltésekor mindig azt kell feltételeznünk, hogy y értéke az a állapotnak felel meg ezért — mivel Moore-modellt tervezünk — az első sorban csakis 0 kimeneti értéket írhatunk (lásd a állapot értelmezése). Ha pl. 01 bemeneti kombináció lép fel, akkor olyan állapotba kell kerülnie a hálózatnak, amelyhez 1-es kimeneti értéket és 0 értékű átvitelt rendeltünk hozzá. Az 1-es kimeneti érték azonban csak akkor alakulhat ki Moore-modell esetén, ha Y már visszahatott y-ként a bemenetre. Az első sor 01 bemeneti kombinációval jelzett rovatába ezért írtunk b 0-t. Ugyanilyen gondolatmenettel tölthetjük ki a többi rovatot is. Ha egy óraimpulzus megjelenésekor például a b állapotnak megfelelő y kombináció jut a bemenetre, a bemeneti kombináció pedig $x_1x_2=11$, akkor a hálózatnak olyan állapotba kell kerülnie, amelyben a kimenet értéke 0, az átvitel értéke pedig 1. Ezt az állapotot c-vel jelöltük. A 0 kimeneti érték viszont csak akkor lép fel, ha y-ná válik a c állapotnak megfelelő Y kombináció. Mindaddig, amíg ez be nem következik, még a b állapotnak megfelelő y kombináció értéke határozza meg a kimeneti értéket, amely az állapotok értelmezése szerint 1. Ezért kell a b-vel jelzett sornak az 11 bemeneti kombinációhoz tartozó rovatába c 1-t írnunk. Az adott példával kapcsolatban megfigyelhetjük, hogy az a és c sorok rovatainak második bejegyzése 0, a b és d soroké pedig 1.

Ha az így kapott állapottáblát összehasonlítjuk a Mealy-modell tervezésekor felírt előzetes állapottáblával (3.43. ábra), akkor megállapíthatjuk, hogy a kimeneti értéktől eltekintve azonosak. A Mealy- és a Moore-modell szerinti állapottábla között az a különbség, hogy az előbbiben az új kimeneti értéket kell bejegyezni a következő állapot mellé, az utóbbiban pedig a kiindulási kimeneti értéket.

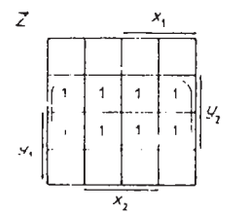
Jelenleg az állapottáblán nem léteznek feleslegesen megkülönböztetett állapotok. Ez természetes is, hiszen Mealy-modell esetén azt találtuk, hogy a kimeneti érték szerint nem volt szükséges megkülönböztetés, Moore-modell esetén viszont a kimenet értékét kizárólag a mindenkori állapot határozza meg. Az állapotokat tehát a kimenet értéke szerint is meg kell különböztetnünk. Emiatt példánkban nem lehetséges további összevonás. Az állapotgráf a 3.60. ábrán látható.



3.60. ábra. Moore-modell szerint működő soros összegző állapotgráfja

y_1y_2 \ x_1x_2	00	01	11	10
00	Y_1Y_2Z 00 0	010	100	010
01	00 1	011	101	011
10	010	100	110	100
11	011	101	111	101

3.61. ábra. Moore-modell szerint működő soros összegző kódolt állapottáblája



3.62. ábra. A Moore-modell szerint működő soros összegző $f_Z'(y)\Rightarrow Z$ leképezését megvalósító $Z=f(y_1, y_2)$ függvény Karnaugh-táblája

Ezután a következő feladat az állapotkódolás megválasztása. Látható, hogy a négy állapot megkülönböztetéséhez két szekunder változó szükséges. Az így képezhető négy bináris kombinációt 4!-féleképpen tudjuk az állapotokhoz hozzárendelni. Mivel az állapotkódolás optimális megválasztására módszert még nem ismerünk, válasszuk az alábbi kódokat:

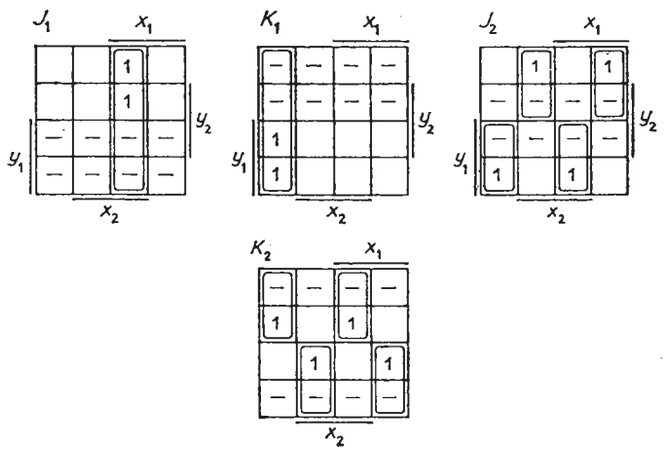
	y_1	y_2
a	0	0
b	0	1
c	1	0
d	1	1

ahol y_1 és y_2 a két szekunder változót jelöli.

Töltsük ki ezek alapján a kódolt állapottáblát (3.61. ábra). A kódolt állapottábla alapján felrajzolhatjuk a Z kimenet függvényének Karnaugh-tábláját (3.62. ábra) és felírhatjuk az egyszerűsített függvényt:

y_1y_2 \ x_1x_2	00	01	11	10
00	J_1K_1 0 – ; J_2K_2 0 –	0 – ; 1 –	1 – ; 0 –	0 – ; 1 –
01	0 – ; – 1	0 – ; – 0	1 – ; – 1	0 – ; – 0
10	– 1 ; 1 –	– 0 ; 0 –	– 0 ; 1 –	– 0 ; 0 –
11	– 1 ; – 0	– 0 ; – 1	– 0 ; – 0	– 0 ; – 1

3.63. ábra. Moore-modell szerint működő soros összegző vezérlési táblája J—K flip-flopokkal történő megvalósítás esetén



3.64. ábra. Moore-modell szerint működő soros összegző J—K flip-flopokkal történő megvalósításához szükséges J és K függvények Karnaugh-táblái

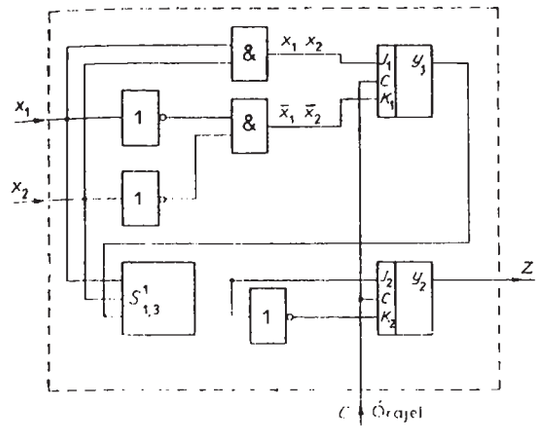
Válasszunk a hálózat megvalósításához J—K flip-flopokat és a 3.31. ábra táblázatában megismert módon készítsük el a vezérlési táblát mindkét szekunder változónak megfelelő flip-flopra. A 3.63. ábrán egy táblázatba foglaltuk mindkét vezérlési táblát. Ennek alapján könnyen felrajzolhatjuk a J_1, K_1, J_2, K_2 függvényekre vonatkozó Karnaugh-táblákat (3.64. ábra). Így az alábbi egyszerűsített függvényekhez jutunk:

$$J_1 = x_1x_2; \qquad K_1 = \bar{x}_1\bar{x}_2,$$

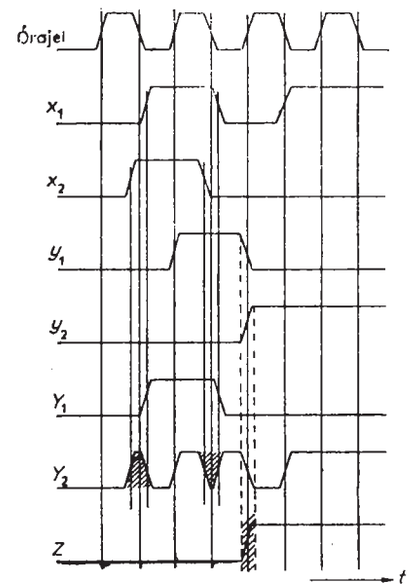
$$J_2 = \bar{x}_1\bar{x}_2y_1+\bar{x}_1x_2\bar{y}_1+x_1x_2y_1+x_1\bar{x}_2\bar{y}_1 = S_{1,3}(x_1, x_2, y_1),$$

$$K_2 = \bar{J}_2.$$

Az elvi logikai rajz alapján (3.65. ábra) szemléletesen látszik, hogy az y_1 jelű flip-flop vezérlése azonos a Mealy-modell szerint adódott vezérléssel (3.49. ábra), a különbség csupán a kimenet előállításában van. Erre változatlanul kiadódott az $S^3_{1,3}$ szimmetrikus függvény, de ez jelen esetben az y_2 flip-flop vezérlését végzi és ez utóbbinak állapota szolgáltatja a kimenetet. Mivel a flip-flopokról a 3.6. ábra alapján feltételezzük, hogy az állapotukat csak az óraimpulzusok felfutó eleinek hatására vál-



3.65. ábra. Moore-modell szerint működő soros összegző elvi logikai rajza J—K flip-floppal történő megvalósítás esetén



3.66. ábra. Moore-modell szerint működő soros összegző idődiagramja

toztathatják, ezért az y_2 flip-flop biztosítja a kimenet érzéketlenségét a két felfutó él közötti átmeneti bemeneti kombinációkra.

Az összehasonlítás érdekében a 3.66. ábrán felrajzoltuk a Moore-modell szerint működő hálózat idődiagramját ugyanarra a bemeneti kombinációsorozatra, amelyre a 3.58. ábra írta le a Mealy-modell működését. Az idődiagramon jól megfigyelhető, hogy a Moore-modell esetén a kimeneti kombináció értelmezési tartományának kijelölését egyszerűbb elvégeznünk. A kimeneti kombináció ugyanis ezúttal órajel-periódusonként csak egyszer változhat. Így a vonalkázott átmeneti tartományon kívül a kimeneti kombinációt órajel-periódusonként bárhol értelmezhetjük.

Egyszerű példánkban az elvi logikai rajzokat összehasonlítva megfigyelhetjük, hogy a Moore-modell szerinti működést az biztosítja, hogy a Mealy-modell kimenetére egy pótlólagos flip-flop (y_2) kapcsolódik, amely nem engedi a kimenetre az átmeneti bemeneti kombinációk hatását. Ezek csak Y_2 értékét tudják befolyásolni, amelynek időbeli változása megegyezik a Mealy-modell kimenetének változásával. A 3.66. ábrán is vonalkázással jelöltük az átmeneti bemeneti kombinációk által okozott impulzusokat Y_2 idődiagramján, amelyek tulajdonképpen az y_2 flip-flop vezérlésében lépnek fel, de olyan időtartományban, hogy y_2 előírt változásait nem zavarják.

Természetesen a két szekunder változóhoz választhattunk volna más-más típusú flip-flopot. Általában nem is tudjuk előre eldönteni, hogy milyen típusú flip-flopok igénylik az adott esetben a legegyszerűbb vezérlő kombinációs hálózatot.

A bemutatott példa kapcsán meggyőződhettünk arról, hogy a Moore-modell szerint működő szinkron sorrendi hálózatok tervezési lépései ugyanazok, mint a Mealy-modell szerint működőké, csupán az előzetes állapottábla felvételében van eltérés.

A szinkron sorrendi hálózatok tervezési lépéseinek összefoglaló áttekintése

1. *Az előzetes állapottábla felírása.* Ehhez szisztematikus módszer nem létezik, a szöveges feladat megfelelő értelmezését feltételezve intuitív jellegű. A feleslegesen megkülönböztetett állapotok a tervezés további lépéseiben összevonhatók, tehát nem jelentenek különösebb nehézséget.

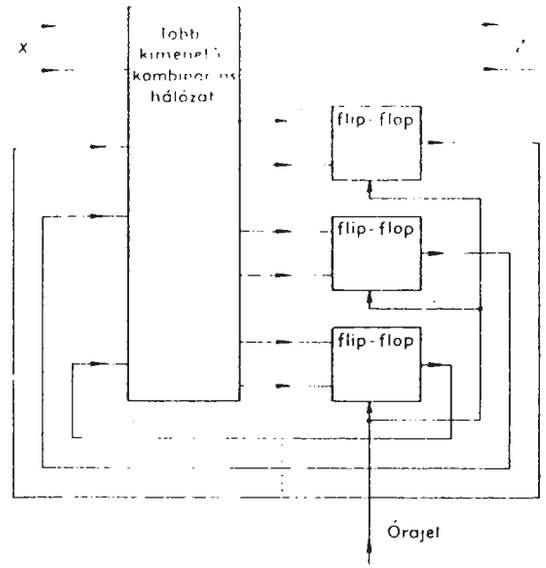
2. *Az összevont (egyszerűsített) állapottábla létrehozása.* Ha az állapottábla teljesen specifikált, akkor erre a tervezési lépésre szisztematikus módszerek ismeretesek. Ha az állapottábla nem teljesen specifikált, akkor ugyanezek a módszerek használhatók, de a segítségükkel kapott eredményből csak intuitív lépések útján kaphatjuk meg a legegyszerűbb összevont állapottáblát.

3. *Az állapotkódolás megválasztása.* A hálózat egyszerűsége szempontjából kedvező kódolás megválasztására léteznek módszerek, de egyikről sem bizonyítható az optimalitás.

4. *Az alkalmazandó flip-flopok megválasztása.* Erre a tervezési lépésre sem létezik szisztematikus eljárás, a flip-flopok típusát többnyire a rendelkezésre álló építőelemkészlet alapján határozzuk meg.

5. *A vezérlési tábla létrehozása.* Ennek módszerét az egyszerű példán bemutattuk. Látszik, hogy a módszer teljesen szisztematikus.

6. *A vezérlő kombinációs hálózat és a kimenetet előállító kombinációs hálózat realizációja.* A függvénynek a vezérlési tábla alapján történő felírásához csak D—G flip-flop alkalmazásakor kell próbálgatással élni. A továbbiakban a kombinációs hálózatokra megismert tervezési módszereket kell alkalmaznunk. A flip-flopokat vezérlő és a kimenetet előállító kombinációs hálózatok tulajdonképpen egyetlen több kimenetű kombinációs hálózatnak tekinthetők, mivel bemeneti változóik azonosak. Ezért a szinkron sorrendi hálózatok a 3.67. ábra szerinti általános vázlat szerint építhetők fel. A kombinációs hálózatok tervezésekor tehát általános esetben mindig



3.67. ábra. A szinkron sorrendi hálózatok felépítésének általános vázlata

a több kimenetű hálózatokra megismert módszereket kell alkalmaznunk, annak ellenére, hogy a bemutatott egyszerű példában a kimenetenkénti minimalizálással is eljutottunk a legegyszerűbb hálózathoz.

A 3.67. ábrán bemutatott általános vázlaton a sorrendi működést biztosító visszacsatolásokat a D flip-floptól eltekintve természetesen nem a flip-flopok kimeneteinek visszavezetései valósítják meg. Ezek csak rajztechnikailag tekinthetők visszacsatolásoknak. A sorrendi működéshez szükséges visszacsatolások az egyes flip-flopokon belül valósulnak meg, hiszen ezekből az elemi sorrendi hálózatokból és a több kimenetű kombinációs hálózatból építhető fel az általános vázlat szerint egy tetszőleges szinkron sorrendi hálózat.

3.5. Aszinkron sorrendi hálózatok tervezési lépéseinek bemutatása egy egyszerű példán

Az aszinkron hálózatok esetén a továbbiakban mindig feltételezzük, hogy a közvetlenül egymás után következő bemeneti kombinációk csak egyetlen bemeneti változóértékben különböznek, vagyis szomszédosak. Enélkül a feltételezés nélkül az aszinkron sorrendi hálózat működése nem követhető egyértelműen az állapottábla alapján, mert funkcionális hazárd jön létre a nem szomszédos bemeneti kombináció-változások miatt. Ha ugyanis pl. az $x_1x_2=00$ bemeneti kombinációt közvetlenül az

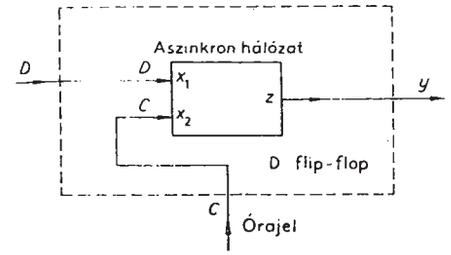
11 kombináció követi, akkor az állapottáblán — az aszinkron működési módból következően — a 00 oszlopban levő stabil állapotból átkerül a hálózat egy olyan stabil állapotba, amely az 11 oszlopban van előírva. A valóságban azonban — mint azt a funkcionális hazárd tárgyalásakor megismertük — a két bemeneti jel változását nem feltétlenül észleli azonos időpontban a hálózat, ill. annak egy része. Emiatt a hálózat a valóságban úgy működik, mintha a 00→11 bemeneti változás helyett a 00→01→11 vagy a 00→10→11 bemeneti változás érte volna, attól függően, hogy a hálózat, ill. annak egy része az x_1 vagy x_2 jel változását érzékelte-e először. Ennek következtében a változás hatására lejátszódó jelenségek formailag úgy követhetők az állapottáblán, mintha a 00 bemeneti kombináció után közvetlenül nem az 11, hanem a 01 vagy 10 bemeneti kombinációk érnék a rendszert. Ez viszont azt jelenti, hogy a 00 oszlopban kialakult stabil állapotot követheti egy 01 vagy 10 oszlopbeli stabil állapot az előírt 11 oszlopbeli stabil állapot helyett. Természetesen előbb-utóbb kialakul az 11 bemeneti kombináció és így az állapottáblán is átjutunk az 11 oszlopba, de nem a 00 oszlopból közvetlenül, hanem a 01 vagy az 10 oszlopból. Emiatt az is előfordulhat, hogy az állapottáblától függően nem az eredetileg előírt stabil állapotba kerül a hálózat.

Mivel a funkcionális hazárdot okozó késleltetési hatások nem tarthatók kézben, ezért a nem szomszédos bemeneti változások hatására a hálózat működése is véletlenszerű lenne. Ha az állapottábla megfelelő kitöltésével kellene biztosítanunk, hogy bármilyen átmeneti bemeneti kombináció esetén se kerüljön az előírttól eltérő stabil állapotba a hálózat, akkor egyrészt bonyolultabbá válna az állapottábla, másrészt továbbra is nehézséget jelentene az, hogy a hálózat bizonyos részei más-más átmeneti bemeneti kombinációt érzékelhetnek. Nem szomszédos bemeneti változáskor természetesen olyan kombinációk is felléphetnek átmenetileg a bemeneten, amelyek a működés során szomszédos bemeneti változáskor is előfordulhatnak. Ilyen esetben az állapottábla létrehozásakor nem is lehet megkülönböztetni az átmeneti bemeneti kombinációkat. A továbbiakban ezért feltételezzük, hogy aszinkron hálózat esetén a bemeneti változások szomszédosak.

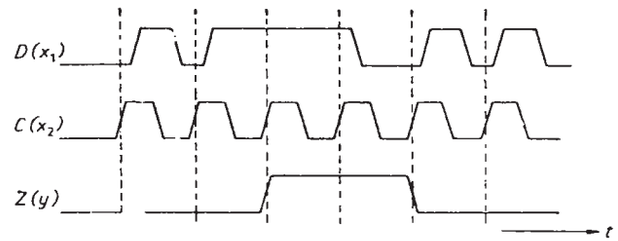
A logikai feladat megfogalmazása

Az aszinkron sorrendi hálózatok tervezési lépéseinek bemutatásához olyan egyszerű példát választunk, amely egyúttal azt is szemlélteti, hogy miként valósítják meg a szinkronizációs feltételeket az integrált áramköri építőelemek szinkron flip-flopjaiban.

A módszer lényege az, hogy a szinkron flip-flopot az integrált áramköri tokon belül aszinkron hálózatként valósítják meg, az órajelet is független bemeneti jelnek tekintve. Természetesen az aszinkron hálózat által megoldandó logikai feladatot a cél érdekében megfelelően kell megfogalmazni a megvalósítandó flip-flop kívánt működése alapján. Ha az így megvalósított aszinkron hálózatot az előírt módon vezéreljük a flip-flop típusára jellemző bemenetein és az órajelet biztosítjuk számára, akkor a felhasználás szempontjából szinkron flip-flopként működik.



3.68. ábra. A szinkron D flip-flop aszinkron hálózattal történő megvalósításának vázlata



3.69. ábra. Az aszinkron hálózattal megvalósított szinkron D flip-flop működésének szemléltetése

Példaként bemutatjuk, hogyan lehet ezzel a módszerrel felépíteni a D flip-flopot, amely típus a szinkron sorrendi hálózatok visszacsatoló ágainak definiált tulajdonságait valósítja meg. Ha olyan aszinkron hálózatot akarunk tervezni, amely megfelelő vezérlés hatására D flip-flopként működik, akkor kétbemenetű hálózatot kell feltételeznünk (3.68. ábra). Az egyik bemenet a flip-flop típusára jellemző vezérlőbemenet (D), a másik pedig az órajel (C).

A D flip-flop kívánt működése alapján a tervezendő aszinkron hálózatra pl. az alábbi módon fogalmazhatjuk meg a logikai feladatot: Z csak akkor változhat, amikor C értéke 0-ról 1-re változik, éspedig olyan módon, hogy ekkor vegye fel D-nek e pillanatban fennálló értékét.

A fenti feltételt szemlélteti a 3.69. ábrán látható idődiagram az ábrázolt $D(x_1)$ változás esetére. Mivel a nem szomszédos bemeneti változásokat kizártnak tekintjük, ezért a működés során fel kell tételeznünk, hogy a $D(x_1)$ és a $C(x_2)$ bemeneti jelek sohasem változnak egyszerre. Erre az így létrejövő D flip-flop használatakor mindig ügyelnünk kell, hiszen D és C egyidejű változásakor a működési feltétel alapján nem egyértelmű, hogy a flip-flopnak milyen D érték alapján kell a kimenetet beállítani. A D flip-flopot megvalósító aszinkron sorrendi hálózat által megoldandó logikai feladat ismeretében a továbbiakban ezen a példán mutatjuk be az aszinkron sorrendi hálózatok tervezési lépéseit.

Az előzetes állapottábla összeállítása

Aszinkron hálózatok esetében előnyös, ha az előzetes állapottáblán minden sorba csak egyetlen stabil állapotot írunk. Ennek oka az, hogy a kizárt nem szomszédos bemeneti változásoknak megfelelően így minden sorban közömbös bejegyzést tehetünk, ami a későbbiekben megnöveli az állapotösszevonási lehetőségeket. A példánkra vonatkozó előzetes állapottábla (3.70. ábra) kitöltését az alábbi gondolatmenettel végezhetjük.

Kiindulásképpen tételezzük fel, hogy a bemeneten $x_1x_2=00$ kombináció áll fenn, és ennek hatására stabil állapot alakul ki, amit jelöljünk *a*-val és rendeljünk hozzá $Z=0$ kimeneti értéket. Ezért írtunk az állapottábla 00 oszlopának *a*-val jelölt rovatába *a* 0-t stabil állapotként. (Az aszinkron hálózatok előzetes állapottáblájának kitöltésekor általában is célszerű ilyen ún. kiindulási állapotot definiálni.) Ebből a stabil állapotból kiindulva az előbbiek értelmében nem léphet fel közvetlenül az 11 bemeneti kombináció, hiszen az a pillanatnyilag fennálló $x_1x_2=00$-hoz képest nem szomszédos bemeneti változást jelentene. Ezért írhatunk az első sor 11-gyel jelölt rovatába közömbös (—) bejegyzést. Látható, hogy ha az első sorban még további stabil állapotok volnának, akkor ez a közömbös bejegyzés nem volna megengedhető, hiszen ekkor az $x_1x_2=11$ kombináció szomszédos bemeneti változás hatására is felléphetne a sorban szereplő többi stabil állapotból kiindulva. A későbbiekben részletezendő összevonási eljárás eredményeként természetesen az összevont állapottáblán egy sorban több stabil állapot adódhat, de az előzetes állapottáblán a soronkénti egy stabil állapot felvétele a közömbös bejegyzések számát növeli, ami éppen az összevonási eljárás hatékonysága szempontjából kedvező. Ha a közömbös bejegyzések számát már az állapottábla kitöltésekor csökkentenénk, akkor lényegében állapotösszevonást végeznénk, de nem biztos, hogy a legegyszerűbb összevont állapottábla szempontjából a legkedvezőbb módon.

Ha a kiindulási *a* 0 stabil állapotban a bemeneti kombináció 00-ról 01-re változik, akkor a megoldandó logikai feladat szerint $D=0$ mellett C értéke 0-ról 1-re változott. Ilyenkor a D flip-flopnak 0 értékű kimeneti jelet kell szolgáltatnia. Megállapodásunk értelmében az első sorban több stabil állapotot nem jelölünk ki, ezért

y \ DC / x_1x_2	00	01	11	10
a	(*a*) 0	*b* 0	-	*c* 0
b	*a* 0	(*b*) 0	*d* 0	
c	*a* 0	-	*e*	(*c*) 0
d	-	*b* 0	(*d*) 0	*c* 0
e		*g* 1	(*e*) 1	*f* 1
f	*h* 1		*e* 1	(*f*) 1
g	*h* 1	(*g*) 1	*e* 1	
h	(*h*) 1	*b*	-	*f* 1

3.70. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat előzetes állapottáblája

írtunk az első sor 01 jelű rovatába *b* 0-t, ami instabil állapotot jelöl ($Y \neq y$). Ha az így kialakuló *b* 0 mint Y kombináció y-ként visszajut a bemenetre, akkor ez az állapottáblán azt jelenti, hogy átjutunk a *b* jelű sorba. Természetesen célszerű a lehető leggyorsabb működést megvalósítani (lásd normál működés), vagyis feltételezni, hogy $x_1x_2=01$ fennállása mellett további állapotváltozás nem játszódik le. A *b* jelű y kombináció tehát a *b* jelű Y kombinációt hozza létre, azaz kialakul a *b* 0 stabil állapot. Ezért írtunk a második sor 01 jelű rovatába *b* 0-t. Azonos gondolatmenettel juthatunk el az *a* 0 stabil állapotból kiindulva az $x_1x_2=10$ kombináció fellépésének hatására a *c* 0 stabil állapotba. Ezzel az előzetes állapottábla első sorának kitöltését befejeztük. A második sor kitöltésekor a *b* 0 stabil állapotból kell kiindulnunk, amihez a bemeneten $DC=x_1x_2=01$ kombináció fennállásának feltételezése szükséges. Ha ebben az állapotban a bemeneti kombináció 11-re változik, vagyis változatlan $C=1$ mellett D értéke 0-ról 1-re változik, akkor a D flip-flop kimenetének nem szabad változnia, tehát továbbra is 0 értékűnek kell maradnia. A második sor 11 jelű rovatába ezért mindenképpen 0 kimeneti értéket kell írnunk, az új állapotot pedig *d*-vel jelöljük. A *d* állapot stabilizálódását a *b* állapotéhoz hasonlóan írhatjuk elő a *d* jelű sorban.

A *b* jelű sorban az $x_1x_2=10$-hoz tartozó rovatba közömbös bejegyzést írhatunk, mert ide megállapodásunk értelmében a *b* stabil állapotból kiindulva csak nem szomszédos bemeneti változás hatására juthatnánk. Ha a *b* stabil állapotból kiindulva a bemeneti kombináció 00-ra változik, akkor ez azt jelenti, hogy a változatlan $D=0$ mellett az óraimpulzus eltűnt (1-ről 0-ra változott), ami a D flip-flop kimenetet nem változtathatja meg. Ezért a *b* jelű sor $x_1x_2=00$ hoz tartozó rovatába 0 kimeneti értéket kell írnunk. Könnyen beláthatjuk, hogy a kialakuló uj stabil állapot *a* lesz, hiszen a megoldandó logikai feladat szempontjából tulajdonképpen a kiindulási helyzet állt elő. Ezért írtunk a második sor 00 jelű rovatába *a* 0-t, ami az első sorban levő *a* 0 stabil állapotba vezeti a hálózatot. Ha nem vettük volna észre, hogy egy már meglevő állapot felhasználható a működés leírására és új állapotba vezettük volna a hálózatot, akkor ez csupán megnövelte volna az előzetes állapottábla terjedelmét. Az összevonási eljárás során ugyanis törvényszerűen kiadódik, hogy az így felvett új állapot és a már definiált *a* állapot megkülönböztetése felesleges. (Természetesen előbb-utóbb észre kell vennünk a kitöltéskor a már meglevő állapotok felhasználhatóságát, különben nem lesz véges az állapottábla.)

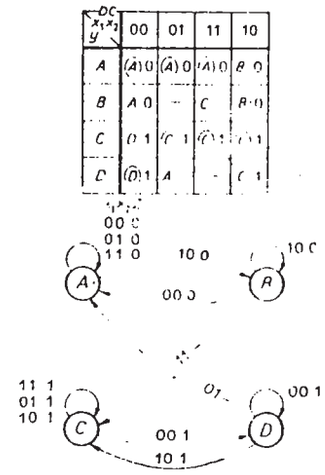
A harmadik sor kitöltésekor a *c* 0 stabil állapotból indulunk ki, amely az 10 jelű rovatban helyezkedik el. Most a harmadik sor 01 jelű rovatába írhatunk közömbös bejegyzést. Tételezzük fel, hogy a *c* 0 stabil állapotból kiindulva fellép az $x_1x_2=11$ bemeneti kombináció, ami azt jelenti, hogy $D=1$ mellett az óraimpulzus megjelenik. Ekkor a D flip-flopnak 1 kimeneti értéket kell előállítani. Ezért az 11 jelű oszlopban olyan stabil állapotba kell jutnia a hálózatnak, amelyhez 1 kimeneti érték tartozik. Ezt az állapotot jelöltük *e*-vel. A harmadik sor 11 jelű rovatában, vagyis az instabil állapot mellé, közömbös kimeneti értéket írtunk. A *c* 0 stabil állapotból az *e* 1 stabil állapotba történő állapotátmenet során ugyanis a kimenet értéke változik, és teljesen

közömbös a megoldandó feladat szempontjából, hogy a közbenső instabil állapothoz még a régi, vagy már az új kimeneti érték tartozik. Ilyen esetekben tehát célszerű közömbös bejegyzést írni az instabil állapothoz tartozó kimeneti értékre, mert ezáltal növelhetjük a kimenet függvényének egyszerűsítési lehetőségeit. Természetesen nem írhatunk közömbös kimeneti értéket olyan állapotátmenetek esetén kialakuló instabil állapot mellé, amelyekhez azonos kimeneti érték tartozik. Ebben az esetben ugyanis a közömbös átmeneti kimeneti érték a megvalósításkor a stabil állapothoz tartozó kimeneti értéktől eltérővé válhat, ami az állapotátmenet során impulzust állíthat elő a kimeneten. Ez általában zavarólag hat a kimenetre kapcsolódó hálózatok működésére.

Az előzetes állapottábla kitöltése a leírt módon folytatható. Befejezésül az utolsó sor kitöltését mutatjuk be. Itt a kiindulási állapot a h stabil állapot $x_1x_2=00$ bemeneti kombináció mellett és a kimenet értéke 1. A 01 bemeneti kombináció fellépése ebben a helyzetben az óraimpulzus (C) megjelenését jelenti $D=0$ mellett. Emiatt a D flip-flopnak 0 kimeneti értéket kell előállítania. Könnyen belátható, hogy ezt egyszerűen biztosítjuk, ha a 01 oszlopban már meglevő b 0 stabil állapotba vezetjük a hálózatot. Ezért írtunk a h jelű sor 01 jelű rovatába instabil állapotként b-t, amelyhez a közömbös kimeneti értéket ezúttal is azért rendelhettük hozzá, mert az állapotátmenet során a kimenet értéke változik. A sor 11 jelű rovatába közömbös bejegyzést írhatunk a nem szomszédos bemeneti változás miatt. Ha a h stabil állapotból kiindulva a fennálló $x_1x_2=00$ bemeneti kombináció után 10 jut a bemenetekre, akkor a D flip-flopnak továbbra is 1 kimeneti értéket kell szolgáltatnia, hiszen az óraimpulzus szünetben ($C=0$) változott D értéke 0-ról 1-re, aminek nem szabad kimeneti változást okoznia. Ehhez az 10 oszlopban nem szükséges új stabil állapotot felvenni, mert az f stabil állapotba vezetve a hálózatot az előírt működést kapjuk. Ezért írtuk a h jelű sor 10 jelű rovatába az f instabil állapotot 1 kimeneti értékkel. A h jelű sor azért az utolsó sora az előzetes állapottáblának, mert egyetlen rovatában sem kellett új, az illető oszlopban még nem szereplő állapotba vezetni a hálózatot és az összes többi felvett stabil állapotnak megfelelő sort is kitöltöttük. Ha az egyes sorok kitöltésekor nem mindig vesszük észre az adott oszlopban már meglevő alkalmas stabil állaba történő átvezetés lehetőségeit, akkor természetesen mindaddig új sorok keletkeznek, amíg ezeket a lehetőségeket ki nem használjuk. A későbbiekben bemutatandó állapot-összevonási eljárásnak éppen az a célja, hogy a feleslegesen felvett sorokat, vagyis a feleslegesen megkülönböztetett állapotokat kimutassa.

Az egyszerűsített (összevont) állapottábla

Az előzetes állapottábla alapján egyszerű példánkban az összevonási eljárás ismerete nélkül is megállapíthatjuk, hogy az a, b és d jelű állapotokat felesleges volt megkülönböztetnünk egymástól. Az ezeknek megfelelő sorok közül bármely kettő azonos bejegyzéseket tartalmaz a specifikált (nem közömbös) helyeken. Ugyanez mondható el az e, f és g állapotokról is. A feleslegesen megkülönböztetett állapotokat lássuk el



3.71. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat összevont állapottáblája és állapotgráfja

azonos jelölésekkel pl. az alábbi módon:

A: a, b, d,

B: c,

C: e, f, g,

D: h.

A bevezetett jelölésekkel az előzetes állapottábla alapján felírt összevont állapottáblát és állapotgráfot a 3.71. ábrán láthatjuk.

Állapotkódolás

Az összevont állapottábla négy állapotának megkülönböztetéséhez legalább két szekunder változó szükséges. Jelöljük ezeket y_1-gyel, y_2-vel és válasszuk az alábbi kódokat:

	y_1	y_2
A	0	0
B	0	1
C	1	0
D	1	1

A kódolt állapottáblát a 3.72. ábrán rajzoltuk fel. Ezzel az állapotkódolással könnyen bemutathatjuk, hogy az aszinkron hálózatok esetén a kódolás megválasztása nemcsak a kialakuló hálózat egyszerűségét befolyásolja, hanem a hálózat helyes vagy hibás működését is meghatározhatja. Ennek bemutatása érdekében tételezzük fel, hogy a hálózat $x_1x_2=10$ mellett az $y_1y_2=01$ stabil állapotban van. Változzon

y_1y_2 \ x_1x_2 (DC)	00	01	11	10
00	Y_1Y_2 Z (00) 0	(00) 0	(00) 0	01 0
01	00 0	—	10 –	(01) 0
10	11 1	(10) 1	(10) 1	(10) 1
11	(11) 1	00 –	—	10 1

3.72. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat kódolt állapottáblája

ezután a bemeneti kombináció 11-re. A 3.72. ábrán követhetjük, hogy a hálózat számára ilyenkor az 11 oszlopban az $y_1y_2=10$ stabil állapot van előírva, amelyet az $y_1y_2Y_1Y_2=0110$ instabil állapot hatására kell elérnie. A bemeneti változás hatására tehát y_1-nek 0-ról 1-re, y_2-nek pedig 1-ről 0-ra kell változnia. Könnyen belátható, hogy az állapottáblából kiolvasott helyes stabil állapot csak akkor alakul ki, ha a két szekunder változó értéke azonos időpontban változik. Ennek szemléltetéséhez tételezzük fel, hogy y_2 gyorsabban változik y_1-nél. Az $x_1x_2=11$ bemeneti kombinációváltozás hatására mindkét szekunder változó értékének változása elő van írva, de feltételezésünk szerint először y_2 változik, mégpedig 1-ről 0-ra. Emiatt átmenetileg létrejön az $y_1y_2=00$ szekunder kombináció. Ez formailag azt jelenti az állapottáblán, hogy átkerülünk az 11 jelű oszlop $y_1y_2=00$ jelű rovatába. Innen viszont rögtön látszik, hogy a kialakult $y_1y_2=00$ kombináció nem is lesz átmeneti, mert 11 bemeneti kombináció esetén a szekunder változók új értékére $Y_1Y_2=00$ van előírva. Az $y_1y_2=00$ állapot tehát stabilizálódik az eredetileg előírt $y_1y_2=10$ helyett. Az y_1-re előírt változás tehát nem tud bekövetkezni, mert $y_1y_2=00$ és $x_1x_2=11$ hatására $Y_1=0$ jön létre. Ennek a hibás állapotátmenetnek természetesen az a következménye, hogy a hálózat nem oldja meg az előírt logikai feladatot.

Ha y_1-et tételezzük fel gyorsabbnak y_2-höz képest, akkor a vizsgált helyzetben átmenetileg az $y_1y_2=11$ szekunder kombináció alakul ki, ami formailag az 11 jelű oszloprovat utolsó sorába vezérli a hálózatot. Ez a rovat nincs specifikálva, tehát a kialakuló stabil állapotot az határozza meg, hogy a megvalósítás során az Y_1Y_2 kombinációt hogyan rögzítjük.

Megállapíthatjuk, hogy a hálózat helyes vagy hibás működése, sőt általában a hibás stabil állapot is a szekunder változók *egymáshoz képesti* működési sebességétől függ. Ezt a sebességarányt a hálózatban fellépő jelterjedési késleltetések befolyásolják, ezért a hálózat működése véletlenszerű. A bemutatott jelenséget a két szekunder változó közötti *kritikus versenyhelyzetnek* nevezik.

Természetesen a versenyhelyzet nem mindig okoz hibás stabil állapotot. Ha például az előző vizsgálatunkban feltételezzük, hogy a megvalósítás során nem rögzítjük a közömbös Y_1Y_2 kombinációt 00-ra vagy 11-re, akkor nem alakulhat ki hibás stabil állapot gyorsabb y_1 esetén. Ilyenkor ugyanis az átmenetileg létrejövő $y_1y_2=11$ kombináció $x_1x_2=11$ mellett akár $Y_1Y_2=01$-et, akár $Y_1Y_2=10$-et hoz létre, az állapottábla kitöltéséből következően mindig az előírt $y_1y_2=10$ stabil állapotba kerül a hálózat. A megvalósítás során csupán a közbenső instabil állapotok függnek az ily módon rögzített Y_1Y_2 kombinációtól.

	y_1y_2 \ x_1x_2 (DC)	00	01	11	10
A	00	Y_1Y_2 Z (00) 0	(00) 0	(00) 0	01 0
B	01	00 0	—	11 –	(01) 0
C	11	10 1	(11) 1	(11) 1	(11) 1
D	10	(10) 1	00 –	—	11 1

3.73. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat kódolt állapottáblája a kritikus versenyhelyzet kiküszöbölése céljából

A szekunder változók közötti versenyhelyzet tehát nem mindig hoz létre hibás stabil állapotot. Előfordulhat, hogy a szekunder változók egymáshoz képesti működési sebessége csak azt befolyásolja, hogy a helyes stabil állapotot milyen instabil állapotok közbeiktatásával éri el a hálózat. Ilyen esetben a *versenyhelyzet nem kritikus*. Természetesen a nem kritikus versenyhelyzet könnyen okozhat véletlenszerű tranziens impulzusokat a kimeneten, hiszen a versenyhelyzettől függően közbeiktatódott instabil állapotokhoz tartozó kimeneti értékek átmenetileg fellépnek.

Azt is láttuk, hogy a logikai feladat szempontjából közömbösnek adódó állapotok a kritikus versenyhelyzet veszélye miatt az állapotkódolástól függően nem rögzíthetők tetszőlegesen a realizáció során.

A következőkben egyszerű példánkban bemutatjuk, hogy a kritikus versenyhelyzet elkerülhető megfelelő állapotkódolással. Válasszuk az alábbi állapotkódokat:

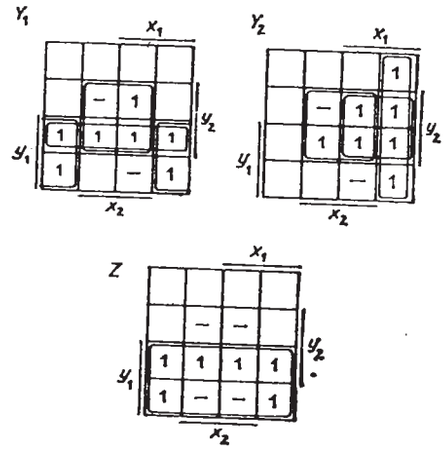
	y_1	y_2
A	0	0
B	0	1
C	1	1
D	1	0

Az így kiadódó kódolt állapottábla a 3.73. ábrán látható. Megfigyelhetjük, hogy ilyen állapotkódolás mellett minden állapotváltozás esetén csak egyetlen szekunder változó változtatja az értékét, tehát versenyhelyzet nem fordulhat elő. A későbbiekben módszereket mutatunk be a kritikus versenyhelyzetet kiküszöbölő állapotkódolás megválasztására.

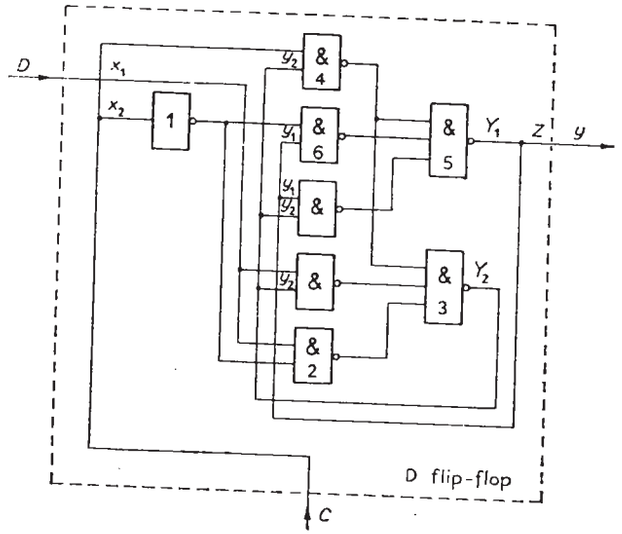
A vezérlési tábla összeállítása

Aszinkron hálózatról lévén szó, elvégezhetjük a megvalósítást visszacsatolt kombinációs hálózattal. Ebben az esetben a vezérlési tábla azonos a kódolt állapottáblával, hiszen az Y_1-t és Y_2-t előállító logikai függvényeket kell megvalósítani, majd visszacsatolni. A kódolt állapottáblából ezért könnyen kiolvashatjuk az Y_1-re, Y_2-re és Z-re vonatkozó Karnaugh-táblákat (3.74. ábra), és ezek alapján elvégezhetjük az egyszerűsítést:

$$Y_1 = x_2y_2+\bar{x}_2y_1+y_1y_2,$$
$$Y_2 = x_2y_2+x_1y_2+x_1\bar{x}_2,$$
$$Z = y_1.$$



3.74. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat visszacsatolt kombinációs hálózattal történő felépítéséhez szükséges Karnaugh-táblák



3.75. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat visszacsatolt kombinációs hálózattal felépített elvi logikai rajza

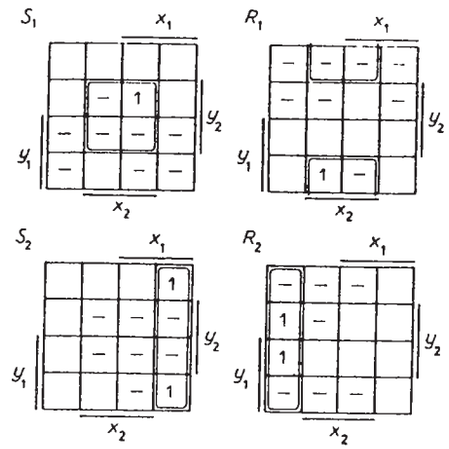
A hazárdmentesítésre feltétlenül szükség van, hiszen láttuk, hogy a szekunder változók átmeneti értékei helytelen stabil állapotba vezérelhetik az aszinkron hálózatot.

Egyszerű példánkban ezúttal nem volt szükséges több kimenetű mininalizálást végezni, mert az x_2y_2 közös prímimplikáns szembetűnő. Ennek figyelembevételével a 3.75. ábrán látható a D flip-flopot megvalósító aszinkron sorrendi hálózat egy lehetséges elvi logikai rajza. Néhány kaput a későbbi hivatkozás céljából megszámoztunk.

Megfigyelhetjük, hogy a kimenet értéke független a bemeneti változóktól, vagyis Moore-modell adódott a hálózatra anélkül, hogy törekedtünk volna erre. Ez természetesen a megoldandó logikai feladat jellegéből következett. Ha az aszinkron elő-

y_1y_2 \ DC x_1x_2	00	01	11	10
00	S_1R_1 0- S_2R_2 0-	0- 0-	0- 0-	0- 10
01	0- 01	-- --	10 -0	0- -0
11	-0 01	-0 -0	-0 -0	-0 -0
10	-0 0-	01 0-	-- --	-0 10

3.76. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat vezérlési táblája S—R flip-flop felhasználása esetén



3.77. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat *S* és *R* függvényeinek Karnaugh-táblái S—R flip-flop felhasználása esetén

zetes állapottábla felvételekor minden sorba csak egyetlen stabil állapotot írunk, akkor tulajdonképpen még nem döntöttünk afelől, hogy a hálózat Mealy- vagy Moore-modell szerint működik-e. Ez csak az összevont állapottábla létrehozása folyamán dől el. Ennek során ugyanis csak akkor tételezünk fel Mealy-modellt, ha összevonunk olyan állapotokat is, amelyekhez a bemeneti kombinációtól függően eltérő kimeneti értékek tartoznak, ill. egy sorban levő közömbös kimeneteket egymástól eltérő módon rögzítünk.

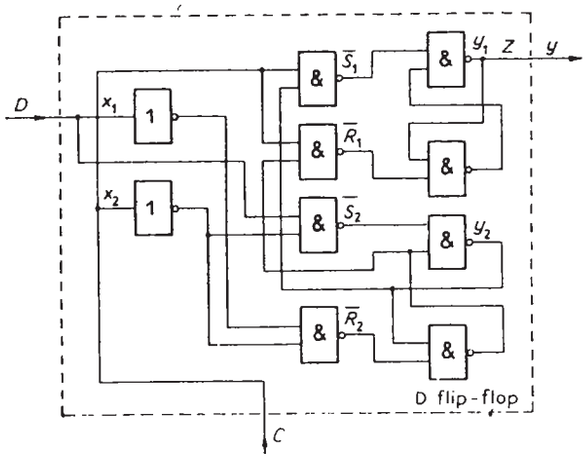
Természetesen az aszinkron hálózatok felépítését is elvégezhetjük a 3.67. ábra általános vázlata szerint aszinkron flip-flopok felhasználásával. Példánkban válasszunk aszinkron S—R flip-flipot.

A vezérlési tábla a 3.76. ábrán, az ennek alapján felírt Karnaugh-táblák pedig a 3.77. ábrán láthatók. A bejelölt összevonások alapján felírhatjuk a legegyszerűbb diszjunktív alakokat:

$$S_1 = x_2y_2 = Cy_2; \qquad R_1 = x_2\bar{y}_2 = C\bar{y}_2;$$

$$S_2 = x_1\bar{x}_2 = D\bar{C}; \qquad R_2 = \bar{x}_1\bar{x}_2 = \bar{D}\bar{C}.$$

A kimeneti függvény változatlanul $Z = Y_1$. A kapott függvényalakok alapján felrajzolhatjuk az S—R flip-flop felhasználásával felépített elvi logikai rajzot (3.78. ábra).



3.78. ábra. A D flip-flopot megvalósító aszinkron sorrendi hálózat S—R flip-flop felhasználásával felépített elvi logikai rajza

A lényeges hazárd

Az eddigiekben megtervezett egyszerű hálózatunkról még mindig nem állíthatjuk, hogy biztonságosan az előírt logikai feladat szerint fog működni. Ennek bemutatása céljából kiindulásképpen tételezzük fel, hogy a hálózat $x_1x_2=11$ bemeneti kombináció mellett az $y_1y_2=00$ stabil állapotban van. Változzon ezután a bemeneti kombináció 10-ra. A kódolt állapottábla alapján (3.73. ábra) láthatjuk, hogy ennek hatására a hálózat az előírás szerint az $x_1x_2=10$ bemeneti kombináció oszlopában levő $y_1y_2=01$ stabil állapotba kerül. A következőkben bemutatjuk, hogy a késleltetési viszonyok kedvezőtlen alakulása esetén ehelyett az állapot helyett ugyanezen oszlop $y_1y_2=11$ állapotában stabilizálódhat a hálózat. Kövessük a leírt bemeneti változás hatását a 3.75. ábra logikai rajzán.

Tegyük fel, hogy az x_2 bemeneti változó 1-ről 0-ra történő változását először az *1* jelű INVERTER-en keresztül a *2* jelű NAND kapu észleli. Ezután a hatás eljut a *3* jelű NAND kapura és $Y_2=1$ jön létre, ami visszahatva $y_2=1$-et hoz létre a *4* jelű NAND kapu bemenetén. Ha a jelterjedési késleltetések miatt a *4* jelű NAND kapu a másik bemeneten csak ezután érzékeli x_2 megváltozását, akkor az *5* jelű NAND kapun keresztül $Y_1=1$ jöhet létre. Ennek visszahatása ($y_1=1$) révén a *6* jelű NAND kapun keresztül stabilan fennmarad az $Y_1=1$ érték, feltéve, hogy a *6* jelű NAND kapu a másik bemenetén már ezt megelőzően érzékelte a bemeneti változást. Mivel $Y_2=1$-et az $x_1x_2=10$ kombináció a *2* NAND kapun keresztül állandóan fenntartja, kialakul az $y_1y_2Y_1Y_2=1111$ stabil állapot, ami nyilvánvalóan a hálózat hibás működését jelenti.

A hibás állapotátmenet létrejöttének oka tehát az, hogy voltak a hálózatnak olyan kapubemenetei, amelyeken a szekunder változók értékváltozása előbb érvényesült, mint a bemeneti változás hatása (pl. *4* jelű NAND kapu). Emiatt pedig olyan

instabil állapot jött létre ($Y_1Y_2=11$), amely a hibás stabil állapotba vezette a hálózatot.

Amikor az aszinkron hálózatok működését az állapottáblán követjük, mindig feltételezzük, hogy a hálózatot először a bemeneti változás éri (másik oszlopba kerülünk), majd ezután érvényesül a szekunder változók megváltozása (az új oszlopban más sorba kerülünk). Példánkban láttuk, hogy ez a feltételezés a jelterjedési késleltetések egymáshoz képesti arányai következtében nem mindig teljesül. Ezért a hálózat az állapottáblán megadott működéstől eltérő módon működhet a késleltetési viszonyoktól függően. A leírt jelenséget úgy is magyarázhatjuk, hogy a helyes vagy hibás működés attól függ, hogy a bemeneti és a szekunder változások egymáshoz képest milyen időrendben játszódnak le. Ha a hálózat ettől az időrendtől függően hibás stabil állapotba kerülhet akármilyen állapotkódolást választunk is, akkor — az elterjedt szóhasználat szerint — a hálózatban *lényeges hazárd* van.

A későbbiekben módszert mutatunk be arra, hogy miként lehet megállapítani egy aszinkron hálózat állapottáblájáról, vajon a bemeneti és a szekunder változások vázolt versenyhelyzete okozhat-e hibás stabil állapotot vagy sem.

A lényeges hazárd okozta bizonytalanságot nem lehet kiküszöbölni az állapotkódolás más megválasztásával, mert az az állapottábla felépítéséből, vagyis magából a megoldandó logikai feladatból ered. Kiküszöbölését csak úgy lehet elvégezni, ha a szekunder változók megváltozásainak visszahatását megfelelő módon lelassítjuk, így biztosítva a bemeneti változások hatásának elsődlegességét a hálózat minden kapubemenetén. Vizsgált példánkban a bemutatott hibás állapotátmenetet kiküszöbölhetjük, ha Y_2 visszahatásának útjába megfelelően megválasztott késleltetést építünk be. A lényeges hazárdot tartalmazó aszinkron sorrendi hálózatok visszacsatoló ágaiba tehát késleltető hatásokat kell beépítenünk, például páros számú INVERTER sorba kapcsolásával.

A lényeges hazárd okozta hibalehetőség nem függ attól sem, hogy a megvalósítást visszacsatolt kombinációs hálózattal vagy flip-flopokkal végezzük. Vizsgált példánkban ugyanúgy kimutatható az előbbi kiindulási feltételekkel a hibás állapotátmenet, ha azt a 3.78. ábrán szereplő S—R flip-flopos elvi logikai rajzon követjük. Ha x_2 értékének 1-ről 0-ra történő változása először $\bar{S}_2=0$ értéket hoz létre, és az ezáltal létrejövő $y_2=1$ még az x_2 változását megelőzően jut el az $\bar{S}_1$ jelet előállító NAND kapura, akkor átmenetileg $\bar{S}_1=0$ állhat fenn, ami $y_1y_2=11$ állapotot okoz. Így a hibás stabil állapot ezúttal is megvalósulhat. Flip-flopos realizálás esetén a késleltető hatások beépítésével egyenértékű, ha a kombinációs kapukhoz képest a flip-flopok jóval lassúbb működésűek. Ha ilyen tulajdonságú építőelem-készletet használunk, akkor a lényeges hazárd nem okoz hibát. Integrált áramköri építőelemek esetén azonban a flip-flopok működési sebessége összemérhető a kombinációs kapukéval, így a lényeges hazárd okozta hibák kiküszöbölése érdekében mindig el kell végezni a vizsgálatot és a szükséges késleltetéseket be kell iktatni a megfelelő visszacsatoló ágakba.

A 3.79. ábrán táblázatosan összefoglaltuk, hogy a jelterjedési késleltetések milyen

A jelenség oka	A jelenség elnevezése	Az okozott hiba	A kikuszobolés módja
A bemeneti változók közötti versenyhelyzet (Nem szomszédos bemeneti kombináció változás)	Funkcionális hazárd	Előre nem meghatározható tranziens jelértékek lehetősége miatt a hálózat működése nem definiálható egyértelműen	Szomszédos bemeneti kombináció változások biztosítása vagy szinkronizáció
A szekunder változók közötti versenyhelyzet	Kritikus versenyhelyzet	A szekunder változók változási sebességarányától függően hibás stabil állapot kialakulásának lehetősége	Az állapotkódolás megfelelő megválasztása
A bemeneti változók és a szekunder változók közötti versenyhelyzet	Lényeges hazárd	A bemeneti és szekunder változók változási sebességarányától függően hibás stabil állapot kialakulásának lehetősége	A megfelelő visszacsatoló ágakba megfelelő értékű késleltető hatás beépítése

3.79. ábra. A jelterjedési késleltetések által az aszinkron sorrendi hálózatokban okozott hibalehetőségek összefoglalása

hibalehetőségeket okozhatnak az aszinkron sorrendi hálózatokban. A kombinációs hálózatokról szóló részben a statikus és dinamikus hazárdot kiküszöböltnek tételeztük fel, ezért a táblázatban ezekre a jelenségekre nem utaltunk.

Könnyen beláthatjuk, hogy a versenyhelyzetekből adódó hibalehetőségek csak az aszinkron hálózatokban okoznak nehézséget. Szinkron hálózatok esetén ugyanis a szinkronizációs feltételek biztosításával ezek a hibalehetőségek kiküszöbölhetők.

Az aszinkron sorrendi hálózatok tervezési lépéseinek összefoglaló áttekintése

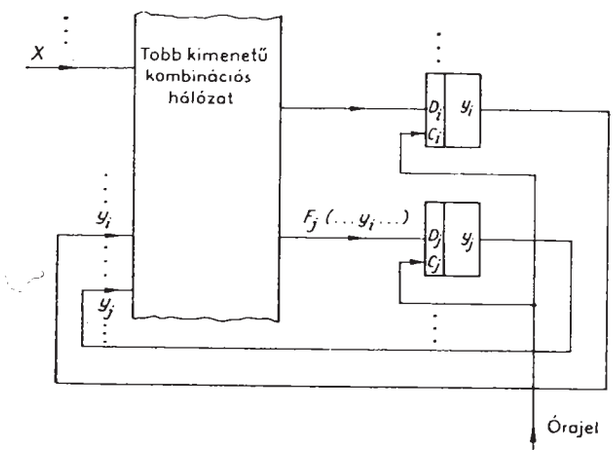
1. *Az előzetes állapottábla felírásakor* az aszinkron működési módot kell szem előtt tartanunk. Kizárólag szomszédos bemeneti kombinációváltozásokat tételezünk fel, és minden sorban csak egy stabil állapotot írunk.
2. *Az összevont állapottábla létrehozásának* módszerei ugyanazok lehetnek, mint szinkron hálózatok esetén.
3. *Az alkalmazandó flip-flop típus megválasztására* érvényesek a szinkron hálózatok esetében tett megállapítások. Az aszinkron hálózatok ezenfelül visszacsatolt kombinációs hálózattal is realizálhatók.
4. Az állapotkódolás megválasztásakor nem a legegyszerűbb hálózat kialakítása, hanem a kritikus versenyhelyzet kiküszöbölése a cél. Erre szisztematikus eljárások léteznek.
5. *A vezérlési tábla felírása és az elvi logikai rajz* létrehozása ugyanúgy végezhető, mint szinkron hálózatok esetén.
6. *Lényeges hazárd* szempontjából ellenőrizni kell az aszinkron hálózatot, és meg kell határoznunk, hogy melyek azok a szekunder változók, amelyek visszahatását késleltetni kell.

A későbbiekben az egyes tervezési lépésekre részletesen ismertetünk eljárásokat.

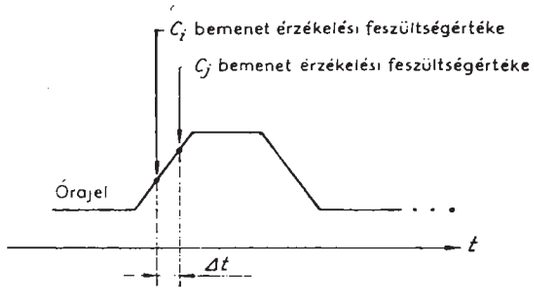
3.6. A szinkronizációs feltételek biztosítása az integrált áramköri építőelemek szinkron flip-flopjaiban

Az előző pontban megismertük a D flip-flop aszinkron hálózattal történő megvalósításának egy módját. A D flip-flopot tartalmazó integrált áramköri tokokban (pl. SN7474N) lényegében a 3.78. ábrán szereplő elvi logikai rajznak megfelelő aszinkron sorrendi hálózatot valósítják meg. A gyártó cégek természetesen garantálják, hogy a tokon belül a lényeges hazárd által okozott hiba nem lép fel.

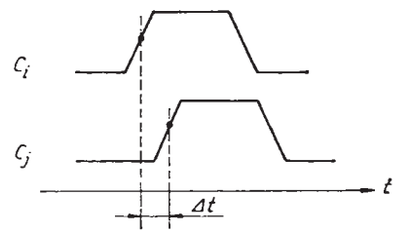
Elvi működéséből következően az így létrejövő D flip-flopot alkalmasnak tartottuk szinkron sorrendi hálózatok felépítésére, hiszen megvalósítja a szinkron sorrendi hálózatok elvi működési vázlatában jelölt visszacsatoló ágbeli elemek funkcióit. A szinkron sorrendi hálózatok D flip-flop felhasználásával történő felépítésének módszerét megismertük a 3.4. pontban. A definiált D flip-flop azonban a gyakorlati megvalósításokban sok esetben bizonytalan működésű szinkron sorrendi hálózatot ad. Ennek bemutatása céljából induljunk ki a 3.67. ábrán látható általános vázlatból. Tételezzük fel, hogy egy állapotátmenet során két flip-flop változtatja az állapotát és a D bemeneteiket vezérlő függvény változói között szerepelnek egymás pillanatnyi állapotai is. Ilyen helyzetet szemléltettünk a 3.80. ábrán. Tételezzük fel, hogy az állapotkódokat úgy választottuk meg, hogy valamelyik állapotátmenet során y_i 1-ről 0-ra, y_j pedig 0-ról 1-re változik. A D_j bemenetre jutó F_j függvény értéke tehát ilyenkor $y_i=1$ esetén szükségszerűen 1, hiszen feltételezésünk szerint változói között szerepel y_i és ennek értéke az állapotátmenetet megelőző állapotban 1. A legközelebbi óraimpulzus felfutó éle okozza az állapotváltozást, vagyis mind y_i, mind y_j megváltozását. A D_i bemenetet vezérlő függvényről csak azt tételezzük fel, hogy az állapotátmenetet előidéző óraimpulzus megjelenésének pillanatában 0 értékű.



3.80. ábra. A D flip-flop által okozott működési bizonytalanság szemléltetése



3.81. ábra. Az érzékelési feszültségérték különbözősége által okozott érzékelési időpont eltérés szemléltetése D flip-flop esetében



3.82. ábra. Két D flip-flop állapotváltoztatási időpontjában fellépő különbség ábrázolása eltérő óraimpulzusok feltételezésével

A két flip-flop az óraimpulzus megjelenésének hatására változtatja állapotát és természetesen lehetnek különbségek közöttük működési sebességben, az órajelet a C_i, ill. a C_j pontokra juttató vezetékek jelterjedési késleltetésében, valamint az óraimpulzus felfutó élének észleléséhez szükséges feszültségértékben. Ez az utóbbi tényező azért okozhat más-más időpontbeli reagálást a két flip-flop esetében az óraimpulzus felfutó élére, mert a felfutó él a valóságban természetesen véges meredekségű, így különböző érzékelési időpontot is jelent. A 3.81. ábrán szemléltetett esetben feltételezzük, hogy a C_i bemenet kisebb feszültségértéknél érzékeli az óraimpulzus felfutását, mint a C_j bemenet. Ennek hatására az érzékelés időpontjában a bejelölt Δt különbség jön létre. Könnyen belátható, hogy a két flip-flopra felsorolt, különbséget okozó tényezők összhatásukban úgy foghatók fel, mintha a két flip-flopra nem ugyanaz az óraimpulzus jutna. A 3.82. ábrán azt feltételezzük, hogy az y_i flip-flop változik előbb, vagyis mintha korábban kapná az óraimpulzust, mint y_j. Ilyen felfogásban a bejelölt Δt időkülönbség képviseli az összes működésbeli különbséget a két flip-flop között, tehát magát az állapotváltozást ezek után egyidejűnek képzelhetjük.

A vizsgált állapotátmenet során (y_iy_j:10→01) kedvezőtlen esetben az is előfordulhat, hogy y_i korábban bekövetkező 0-ra való változásának hatása még a Δt időkülönbség letelte előtt visszajut a D_j bemenetre, vagyis az F_j függvény y_j változójának értéke 0-vá válik. Ha ezek után még azt is feltételezzük, hogy F_j értéke $y_i=0$ esetén 0, akkor D_j-re 0 juthat, mielőtt még az y_j flip-flop érzékelné az óraimpulzus felfutó élét. Ennek természetesen az lesz a következménye, hogy a felfutó él érzékelésekor y_j flip-flop állapota nem tud 1-re változni, vagyis nem jön létre az előírt állapotátmenet.

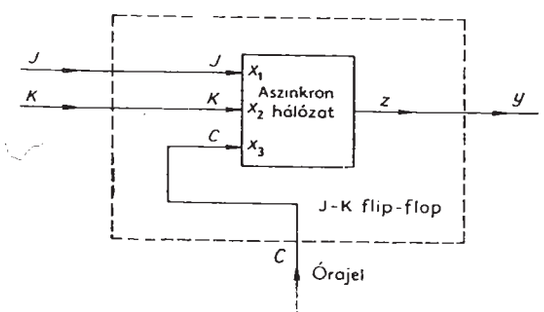
A bemutatott hibás működés lehetősége miatt a definiált működésű D flip-flop nem alkalmazható általánosan szinkron sorrendi hálózatok gyakorlati megvalósítására. Ha az állapotkódokat úgy választjuk meg, hogy minden előforduló állapotátmenet során csak egyetlen D flip-flop változtatja az állapotát, akkor a fentiekben bemutatott hibás működés természetesen kizárt. Az ilyen ún. *szomszédos kódolás* azonban az állapottáblától függően nem mindig alakítható ki annak megváltoztatása nélkül (lásd később). A szomszédos kódolás ezen kívül olyan erős megkötést jelent a kódválasztásra, hogy általában nem tudjuk megfelelő mértékben figyelembe venni a kialakuló hálózat egyszerűségének szempontjait.

A definiált működésű D flip-flopot tehát csak olyan speciális szinkron sorrendi hálózatok megvalósítására használhatjuk, amelyek működésében jellegükből következően nem fordulhatnak elő a 3.80. ábra alapján bemutatott esetre visszavezethető helyzetek. Ilyenek például az átmeneti információtárolást végző hálózatok (regiszterek) és a szinkronizálást megvalósító hálózatok.

Ahhoz, hogy a szinkron sorrendi hálózatok gyakorlati megvalósítására általánosan alkalmazható flip-flophoz jussunk, másként kell definiálnunk az óraimpulzus érzékelési időpontjához képest az állapotváltozás időpontját. Ezzel nem változtatjuk meg a szinkron sorrendi hálózatok megismert elvi működését, hiszen éppen azért kell módosítanunk a definiált D flip-flop működését, mert az csak az elvi vizsgálatok szintjén volt alkalmas a szinkronizációs feltételek biztosítására. Természetesen csak olyan változtatásokat hajtunk végre a flip-flop működési definíciójában, amelyek nem érintik a 3.4., ill. 3.6. ábrán vázolt szinkron működés lényegét. Az ilyen változtatásokra nyilvánvalóan van lehetőség, hiszen az elvi magyarázatot segítő 3.4. és 3.6. ábrákon tulajdonképpen önkényesen váltaszottuk meg az X és y változásoknak az óraimpulzushoz képesti időpontjait.

Az alábbiakban bemutatjuk a működési definíció alkalmas megváltoztatásának egy lehetséges módját. A flip-flop megvalósítására ezúttal is aszinkron hálózatot tervezünk. A változatosság kedvéért a megvalósítandó flip-flop típusa legyen J—K. A megvalósítás vázlata ekkor a 3.83. ábra szerinti. Az aszinkron hálózat által megoldandó logikai feladatot fogalmazzuk meg az alábbi módon:

1. Az óraimpulzus-szünetekben bármely időpontban beállítjuk a kívánt JK érté-



3.83. ábra. A J—K flip-flop aszinkron hálózattal történő megvalósításának vázlata

keket. Ezeknek a bemeneti változásoknak a szinkron flip-flop állapotában (Z kimenet értéke) nem szabad változást okozniuk.

2. Az óraimpulzus tartama ($C=1$) alatt a J és K értékeket nem változtatjuk.

3. Az óraimpulzus eltűnésének hatására (C értékének 1-ről 0-ra változása) a Z kimenet értéke váljon egyenlővé azzal az értékkel, amelyet az óraimpulzus eltűnésekor fennálló J és K értékek a J—K flip-flop által megoldandó logikai feladat alapján előírnak.

Látható, hogy az 1. és 2. feltétellel a J—K flip-flopot vezérlő hálózat működésére adtunk előírást. A 2. feltétel a tervezés során egyszerűsítést tesz lehetővé, ugyanis növeli az állapottábla közömbös bejegyzéseinek számát. A 3. feltétel a 3.80. ábrával kapcsolatban bemutatott hibás működési lehetőséget hivatott kiküszöbölni.

Az előzetes állapottábla kitöltését a 3.84. ábra alapján követhetjük. Elvégezve az állapot-összevonásokat és bevezetve az alábbi jelöléseket:

A: $a, b, c, d, e, f,$

B: g, h

C: $p, n,$

D: $i, j, k, l, m, o,$

a 3.85. ábrán szereplő összevont állapottáblához jutunk.

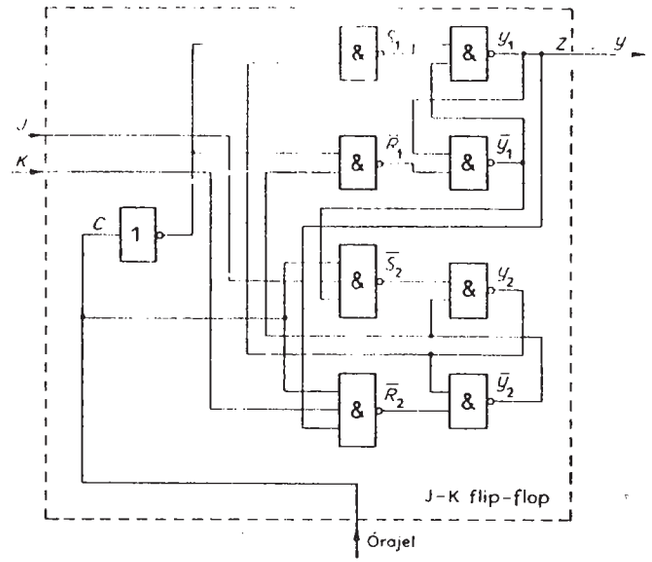
Válasszuk a 3.85. ábrán feltüntetett versenyhelyzetmentes állapotkódolást. A hálózat megvalósításának menete ismert, ezért azt nem részletezzük. Ha aszinkron

y \ CJK (x_1x_2)	000	001	011	010	100	101	111	110
a	(a) 0	b 0	–	c 0	d 0	–	–	–
b	a 0	(b) 0	e 0	–	–	f 0	–	–
c	a 0	–	e 0	(c) 0	–	–	–	g 0
d	a 0	–	–	–	(d) 0	–	–	–
e	–	b 0	(e) 0	c 0	–	–	h 0	–
f	–	b 0	–	–	–	(f) 0	–	–
g	–	–	–	i –	–	–	–	(g) 0
h	–	–	j –	–	–	–	(h) 0	–
i	k 1	–	j 1	(i) 1	–	–	–	m 1
j	–	l 1	(j) 1	i 1	–	–	n 1	–
k	(k) 1	l 1	–	i 1	o 1	–	–	–
l	k 1	(l) 1	j 1			p 1	–	–
m	–	–		i 1		–	–	(m) 1
n	–	–	e –	–		–	(n) 1	–
o	k 1	–	–	–	(o) 1	–	–	–
p	–	b –	–	–		(p) 1	–	–

3.84. ábra. A J—K flip-flopot megvalósító aszinkron sorrendi hálózat előzetes állapottáblája

$y_1\ y_2$	y \ CJK	000	001	011	010	100	101	111	110
00	A	(A) 0	(A) 0	(A) 0	(A) 0	(A) 0	(A) 0	B 0	B 0
01	B	–	–	D –	D –	–	–	(B) 0	(B) 0
10	C	–	A –	A –	–	–	(C) 1	(C) 1	–
11	D	(D) 1	(D) 1	(D) 1	(D) 1	(D) 1	C 1	C 1	(D) 1

3.85. ábra. A J—K flip-flopot megvalósító aszinkron sorrendi hálózat összevont állapottáblája



3.86. ábra. A J—K flip-flopot megvalósító aszinkron sorrendi hálózat S—R flip-flop felhasználásával felépített elvi logikai rajza

S—R flip-flopot alkalmazunk, akkor az alábbi logikai függvényeket kapjuk:

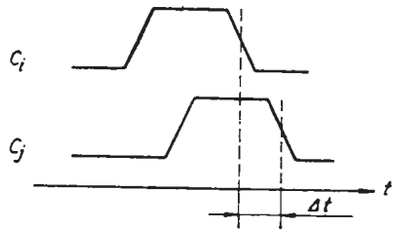
$$S_1 = y_2\bar{C}, \qquad R_1 = \bar{y}_2\bar{C},$$

$$S_2 = JC\bar{y}_1, \qquad R_2 = KCy_1,$$

$$Z = y_1.$$

Láthatjuk, hogy ezúttal is Moore-modellhez jutottunk. A NAND kapukból felépített hálózat elvi logikai rajzát a 3.86. ábrán mutattuk be.

Ha megfigyeljük a két aszinkron S—R flip-flop vezérlési függvényeit, akkor megállapíthatjuk, hogy az y_1 jelű csak az óraimpulzus-szünetekben, az y_2 jelű pedig csak az óraimpulzus tartama alatt változtathatja meg állapotát. Az óraimpulzus tartama alatt ($C=1$) az y_2 jelű flip-flop állapota beáll az óraimpulzus-szünetekben beállított J és K értékeknek, valamint a megvalósítandó J—K flip-flop pillanatnyi állapotának ($y_1=Z=y$) megfelelően. Így az y_2 jelű flip-flop valósítja meg a megoldandó logikai feladatnak megfelelő döntést. Ennek a döntésnek az eredményét az y_1 jelű flip-flop továbbítja a J—K flip-flop kimenetére, hiszen az óraimpulzus eltűnésének hatására ($\bar{C}=1$) „szolgaian" lemásolja az y_2 jelű flip-flop állapotát. A két aszinkron flip-flop



3.87. ábra. Két master—slave flip-flop állapotváltoztatási időpontjában fellépő különbség ábrázolása eltérő óraimpulzusok feltételezésével

$y_1 y_2$	y \ CJK	000	001	011	010	100	101	111	110
00	A	Ⓐ0	Ⓐ0	Ⓐ0	Ⓐ0	Ⓐ0	Ⓐ0	B 0	B 0
01	B	D 0	D 0	D 0	D 0	Ⓑ0	Ⓑ0	Ⓑ0	Ⓑ0
10	C	A 1	A 1	A 1	A 1	Ⓒ1	Ⓒ1	Ⓒ1	Ⓒ1
11	D	Ⓓ1	Ⓓ1	Ⓓ1	Ⓓ1	Ⓓ1	C 1	C 1	Ⓓ1

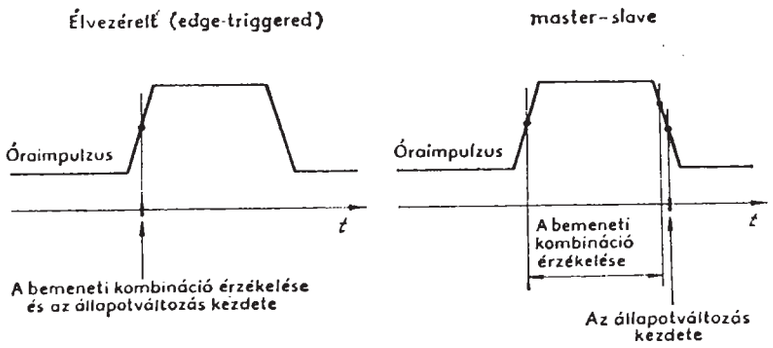
3.88. ábra. A master—slave J—K flip-flopot megvalósító aszinkron sorrendi hálózat összevont állapottáblája kiegészítve a nem specifikált állapotok rögzítésével

vázolt működésbeli kapcsolatára utalva nevezik az így létrejövő J—K flip-flopot *master—slave* (mester—szolga) működésűnek. Az elmondottakból következik, hogy az y_2 jelű a master, az y_1 jelű pedig a slave flip-flop.

Vizsgáljuk meg, hogy az így kapott master—slave flip-flop alkalmazása esetén felléphet-e a 3.80. ábrával kapcsolatban leírt hibalehetőség. Az időben eltolt óraimpulzusok feltételezésével szemléltetett működési sebességbeli különbséget most a lefutó élek között kell vizsgálni, mivel az állapotváltozás az óraimpulzus eltűnésekor jön létre (3.87. ábra). Vizsgálatunk szempontjából nincs jelentősége annak, hogy a flip-flop típusa D vagy J—K, hiszen master—slave elven D flip-flopot is létrehozhatunk. A 3.80. ábrán vázolt helyzet előállításához most azt kell feltételeznünk, hogy a C_i lefutó éle által okozott y_i változás még a Δt időkülönbség eltelte előtt, azaz y_i szempontjából még az óraimpulzus tartama alatt ($C_j = 1$) visszajut az y_j jelű flip-flop bemenetére. Ezáltal a master—slave működésűnek feltételezett y_j jelű flip-flopra nem teljesül az az előírás, hogy az óraimpulzus tartama alatt a bemenetére jutó jelek ne változzanak. A master—slave flip-flop állapottáblájának felvételekor a 3.84. ábrán az óraimpulzus alatti bemeneti változás eseteire a működést nem specifikáltuk, vagyis a megfelelő rovatokba közömbös bejegyzéseket tettünk. A közömbös bejegyzések egy része az állapot-összevonás után is megmaradt az összevont állapottáblán. Nyilvánvaló ezek után, hogy az óraimpulzus tartama alatti bemeneti változások a master—slave flip-flopot a nem specifikált állapotokba vihetik át. A működés tehát attól függ ilyen esetekben, hogy az adott megvalósítás hogyan rögzíti a nem specifikált állapotokat. Könnyen ellenőrizhető, hogy a 3.85. ábrából kiindulva nyert vezérlési függvényeink a nem specifikált állapotokat úgy rögzítik, hogy azok egy része nem okoz hibás működést akkor sem, ha az óraimpulzus alatti bemeneti változások hatására kialakulnak ezek az állapotok. A 3.88. ábrán a 3.85. ábrán szereplő összevont állapottáblát láthatjuk kiegészítve az adott megvalósítás által rögzített állapotokkal. Folytonos nyíllal bejelöltünk egy állapotátmenetet. Szaggatott nyíllal jelöltük azt az

állapotátmenetet, amely akkor megy végbe, ha az óraimpulzus tartama alatt megváltozik a J és K bemenetekre jutó kombináció. Látható, hogy ilyenkor az óraimpulzus eltűnésének hatására az A 0 helyett a D 1 állapotba jut a hálózat, ami hibás működésnek tekinthető. Ha a Δt időkülönbségen belüli visszahatás például így érvényesül az y_j jelű flip-flop bemenetén, akkor a hibalehetőség ugyanúgy fennáll, mint a korábban definiált D flip-flop esetében. A master—slave működési elv tehát ilyen szempontból csupán annyi előnyt jelent, hogy nem minden óraimpulzus alatti bemeneti változás okoz hibás működést, vagyis a 3.80. ábrával kapcsolatos feltételezések esetén sem mindig hibás a működés. Ezért a master—slave működésű flip-flopok sem használhatók általánosan szinkron sorrendi hálózatok megvalósítására. Az óraimpulzus alatti bemeneti változások egy részének hatástalansága miatt azonban szélesebb azoknak a speciális szinkron sorrendi hálózatoknak a köre, amelyekben a működés jellegéből következően a master—slave működésű flip-flopok gyakorlatilag is biztonságosan alkalmazhatók. Ilyen hálózatok a regisztereken és szinkronizáló egységeken kívül például a számlálók. Meg kell jegyeznünk, hogy amikor az óraimpulzus tartama alatti bemeneti változások hatástalanságáról beszélünk, akkor hallgatólagosan mindig feltételezzük, hogy a 3.87. ábrán jelölt Δt időkülönbség értéke nem haladja meg az óraimpulzus időtartamát. Ha ugyanis ezt nem tételeznénk fel, akkor előfordulhatna, hogy C_i és C_j nem fedik át egymást és így y_i megváltozásának hatása az y_j jelű flip-flop bemenetén még C_j előtt érvényesülhetne, ami nem tenné lehetővé a master—slave működés esetén említett szélesebb körű gyakorlati alkalmazhatóságot.

Master—slave flip-flop esetén tehát az állapottábla közömbös bejegyzéseinek rögzítésétől függ, hogy az óraimpulzus alatti bemeneti változások közül melyik okoz hibás működést és melyik nem. A közömbös bejegyzések rögzítését nemcsak a vezérlési függvények egyszerűsítése befolyásolja, hanem az állapottábla összevonásakor keletkező egyenértékű megoldások közüli választás is. Az állapot-összevonási eljárások ismertetésekor megismerjük annak magyarázatát, hogy példánkban másfajta állapot-összevonásokkal is eljuthattunk volna olyan összevont állapottáblához, amely egyszerűség szempontjából is egyenértékű a 3.85. ábrán szereplővel, de attól eltérő módon rögzít néhány közömbös bejegyzést. Ezért az így megvalósuló flip-flop annak ellenére, hogy ugyanúgy megoldja az előírt logikai feladatot, mégis másként viselkedik az óraimpulzus tartama alatti bemeneti változások hatására. Az eltérést az is fokozhatja, hogy az összevonások során másként rögzített nem specifikált bejegyzések a vezérlési függvények egyszerűsítésekor is más helyzetet teremtenek, ami újabb eltéréseket okozhat a rögzítésben. Az integrált áramköri katalógusokban általában csak a flip-flop specifikált működéséről tesznek említést, így nem mindig állapítható meg közvetlenül, hogy az adott típus miként működik az óraimpulzus tartama alatti bemeneti változásokra, vagyis a 3.80. ábrán vázolt helyzetben fennáll-e a hibás működés lehetősége vagy sem. Ezt csak úgy tudjuk egyértelműen megállapítani, hogy elvégezzük a tokon belüli aszinkron sorrendi hálózat analízisét, vagyis a logikai függvények alapján felírjuk az állapottáblát. Ez a vizsgálat igen fontos olyan-

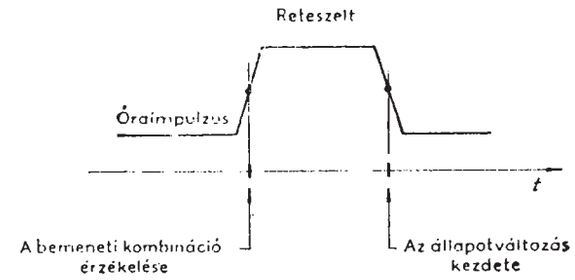


3.89. ábra. A bemeneti kombináció érzékelésének és az állapotváltozás kezdetének időpontjai élvezérelt és master—slave flip-flop esetén

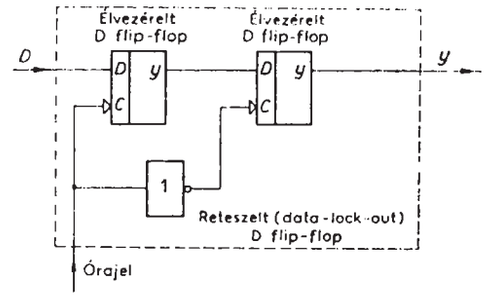
kor, amikor egy adott berendezés gyártásakor például beszerzési okokból át kell térni egy másik cég ún. ekvivalens típusára, amelyről csupán azt tudjuk, hogy csak a specifikált működés esetében egyenértékű. Sokszor az is előfordul, hogy ugyanaz a gyártó cég logikai működés szempontjából több egyenértékű flip-flop típust hoz forgalomba eltérő működési sebesség, teljesítmény stb. paraméterekkel. Ilyenkor is csak azt tételezhetjük fel, hogy ezek a típusok a specifikált logikai működés szempontjából egyenértékűek. Az eltérő paraméterek biztosítása érdekében ugyanis sokszor más az állapot-összevonás és a vezérlési függvények algebrai alakja, így más a nem specifikált állapotok rögzítése.

A master—slave flip-flop működési elvének következtében a bemeneti kombináció érzékelésének és az állapotváltozás kezdetének időpontjai jobban el vannak választva egymástól, mint a vizsgált D flip-flop esetében (3.89. ábra). Az eddig vizsgált D flip-flopot gyakran nevezik *élvezérelt (edge-triggered)* működésűnek. A 3.89. ábrán megfigyelhetjük, hogy az élvezérelt flip-flop az érzékelés időpontjában az állapotváltoztatást is megkezdi. A master—slave flip-flop viszont az óraimpulzus teljes tartama alatt érzékeli a bemeneti kombinációt és csak az óraimpulzus 0 értékének észlelésekor kezdi meg az állapotváltoztatást. Az ábrán csak azért különböztettük meg a lefutó élen a bemeneti kombináció érzékelésének befejezését az állapotváltozás kezdetétől, mert ezek az időpontok a gyakorlatban (lásd master—slave flip-flop elvi logikai rajza, 3.86. ábra) nem feltétlenül azonos feszültségértékhez tartoznak, sorrendjük azonban nyilvánvalóan nem lehet fordított.

Láttuk, hogy master—slave flip-flop alkalmazása esetén az okozza a szinkron sorrendi hálózat hibás működésének lehetőségét, hogy az óraimpulzus tartama alatt a flip-flop vezérlő bemenetei a késleltetési viszonyoktól függően megváltozhatnak, amit a flip-flop érzékel és emiatt hibás állapotátmenet játszódhat le. A szinkron sorrendi hálózatok gyakorlati megvalósítására általánosan használható flip-flop típus előállítása céljából tovább kell módosítani a flip-flopot megvalósító aszinkron sorrendi hálózat által megoldandó logikai feladatot. Könnyen beláthatjuk, hogy a



3.90. ábra. A bemeneti kombináció érzékelésének és az állapotváltozás kezdetének időpontjai reteszelt (data-lock-out) működésű flip-flop esetén



3.91. ábra. Reteszelt működésű D flip-flop előállítása két élvezérelt D flip-flop felhasználásával

master—slave flip-flop által okozott hibalehetőségeket kiküszöbölhetjük, ha a megvalósító aszinkron hálózatot érzéketlenné tesszük az óraimpulzus tartama alatti bemeneti változásokra. Ehhez az alábbi logikai feladatot fogalmazhatjuk meg például D flip-flop esetére a 3.68. ábra jelöléseivel:

Z értéke csak *C* eltűnésének hatására változzék meg, és ekkor mindig vegye fel *D*-nek azt az értékét, amely *C* megjelenésének pillanatában fennállt.

Ilyen működés esetén a bemeneti kombináció érzékelésének és az állapotváltozás kezdetének időpontjai határozottan külön vannak választva, közöttük éppen egy óraimpulzus szélességének megfelelő idő telik el. Ezt szemléltettük a 3.90. ábrán. Az óraimpulzus tartama alatti bemeneti változásokra való érzéketlenségére utalva az ilyen flip-flopokat általában *reteszelt (data-lock-out)* működésűnek nevezik. Használatos az *élvezérelt master—slave* elnevezés is.

A reteszelt flip-flopot megvalósító aszinkron sorrendi hálózat elvi logikai rajzához a megfogalmazott logikai feladatból kiindulva ugyanúgy juthatunk el, mint az élvezérelt és a master—slave flip-flop esetében. Eddigi ismereteink alapján azonban kihagyhatjuk ezeket a tervezési lépéseket. Két élvezérelt D flip-flop felhasználásával ugyanis a 3.91. ábra szerint előállíthatjuk a reteszelt működésű D flip-flopot. Az első flip-flop az óraimpulzus felfutó élének hatására mintát vesz a *D* bemenet értékéből, a második — mivel negált órajelet kap — a lefutó él hatására lemásolja az első flip-flop állapotát. Az így kialakuló eredő reteszelt flip-flop tehát valóban nem veszi figyelembe az óraimpulzus tartama alatti esetleges bemeneti változásokat.

3.91. ábrán szereplő vázlatnak megfelelő elvi logikai rajzhoz természetesen úgy

is eljuthatunk, hogy a reteszelt flip-flopra megfogalmazott logikai feladat alapján előzetes állapottáblát készítünk és végrehajtjuk a tervezés ismert lépéseit.

Reteszelt működésű flip-flopok alkalmazása esetén nem léphet fel a 3.80. ábrával kapcsolatban említett működési bizonytalanság a szinkron sorrendi hálózat működésében. Ekkor ugyanis a 3.87. ábrán feltételezett eltérő óraimpulzusok hiába okoznak változásokat az y_j flip-flop bemeneten C_j tartama alatt, ezek hatástalanok maradnak a működésre. Ez természetesen csak addig igaz, ameddig biztosítható, hogy a 3.87. ábrán jelölt Δt időkülönbség nem lesz nagyobb az óraimpulzus időtartamánál. Ha ugyanis ez bekövetkezhet, akkor a két óraimpulzusnak nem lesz átfedése és ezáltal a C_i lefutó éle által okozott változások C_j felfutó éle előtt visszahatnak az y_j jelű flip-flop bemenetére. Az ilyen bemeneti változásokra y_j természetesen nem érzéketlen és ugyanolyan működési bizonytalanság léphet fel, mint élvezérelt flip-flopok alkalmazása esetén. A Δt időnek ilyen viszonylagos megnövekedése azonban ellensúlyozható az óraimpulzus szélességének és az órajel frekvenciájának a megfelelő megválasztásával.

A reteszelt működésű flip-flopok tehát általánosan alkalmazhatók szinkron sorrendi hálózatok gyakorlati megvalósítására. A 3.4., ill. 3.6. ábrán vázolt szinkron működés lényegét természetesen nem változtatjuk meg reteszelt működésű flip-flop alkalmazása esetén, csupán az X és y változások óraimpulzushoz képesti időpontjait kell a reteszelt flip-flop működésének megfelelően módosítani.

A 3.78., 3.86. és 3.91. ábrákon bemutatott elvi logikai rajzok lényegében megegyeznek az integrált áramköri építőelemek katalógusaiban közölt elvi logikai rajzokkal az egyes szinkron flip-flopokat megvalósító tokokra vonatkozóan. Megjegyezzük, hogy az elmondottak szerint felépített szinkron flip-flopokat tartalmazó tokok általában olyan bemeneti pontokkal is rendelkeznek, amelyekre adott jellel a megvalósító aszinkron hálózat flip-flopjait tetszőleges állapotba lehet hozni az összes többi bemeneti jel értékétől (így az órajelétől is) függetlenül. Az ilyen bemeneteket általában „Preset" és „Clear" bemeneteknek nevezik, és könnyen elképzelhetjük hatásuk megvalósítását, hiszen az aszinkron flip-flopok S és R bemeneteire kell hatniuk a többi hatással VAGY kapcsolatban. A flip-flopok ilyen beállítási lehetősége rendkívül előnyös a sorrendi hálózatok kiindulási állapotainak biztosítása szempontjából.

A bemutatott flip-flopokat megvalósító aszinkron sorrendi hálózatok mindegyikének állapottáblájáról megállapítható, hogy tartalmaznak lényeges hazárdot. Az integrált áramköri tokban megvalósított aszinkron hálózatokról azonban feltételezhetjük, hogy a lényeges hazárd jelensége nem okoz hibát a működésben. A megbízható működést a gyártó cég minden esetben garantálja, vagyis a lényeges hazárd okozta hibát kiküszöbölő késleltető hatásokat a gyártástechnológia során építik be a visszacsatoló ágakba.

A szinkron flip-flopokat tartalmazó integrált áramköri tokokat — bemutatott felépítésükből következően — természetesen aszinkron sorrendi hálózatként is felhasználhatjuk. Ilyenkor az órajelet is logikai bemeneti jelnek tekintve a szinkron flip-flopot megvalósító aszinkron hálózatok állapottáblájának megfelelő működésű

egységként alkalmazhatjuk a tokot. Az ilyen alkalmazásokhoz természetesen nem elegendő a tok szinkron flip-flopként specifikált működésének ismerete, mert ebből általában nem következik egyértelműen a nem specifikált állapotátmenetek rögzítési módja. Ezért a tokon belül megvalósított aszinkron hálózat állapottábláját az ilyen alkalmazásokhoz mindig pontosan meg kell ismerni a következőkben bemutatandó analízislépések segítségével.

3.7. Sorrendi hálózatok analízise

A sorrendi hálózatok megismert tervezési lépései igen egyszerűen és előnyösen alkalmazhatók analízis jellegű feladatok során. Ilyenkor az a célunk, hogy egy ismeretlen működésű, pl. elvi logikai vázlatával adott sorrendi hálózat által megoldott logikai feladatot megismerjünk. A tervezési lépések ismeretében nem kell mást tennünk, mint azokat fordított sorrendben alkalmazni. Az eljárást egy egyszerű példán mutatjuk be.

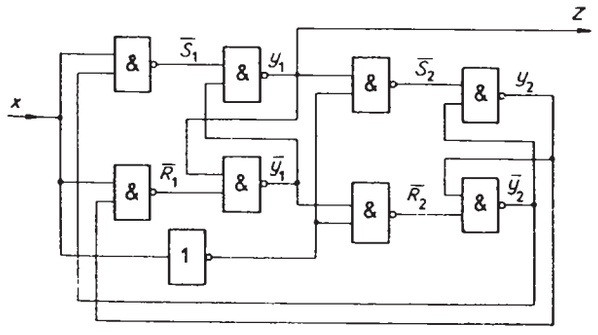
Határozzuk meg a 3.92. ábrán elvi logikai rajzával adott aszinkron sorrendi hálózat által megoldott logikai feladatot. A rajz alapján könnyen felismerhetjük a NAND kapukból felépített két aszinkron S—R flip-flopot. Ha vezérlőbemeneteiket rendre S_1, R_1, S_2, R_2-vel jelöljük, akkor az ábra alapján az alábbi módon írhatjuk fel a vezérlőbemenetekre jutó logikai függvényeket:

$$S_1 = x\bar{y}_2, \qquad R_1 = xy_2,$$

$$S_2 = \bar{x}y_1, \qquad R_2 = \bar{x}\bar{y}_1.$$

Ebben a kiindulási lépésben a megfelelő S és R bemeneteket egymással felcserélve is értelmezhettük volna anélkül, hogy ez az analízis végeredményét befolyásolná, hiszen ekkor a megfelelő y és $\bar{y}$ értékeket is következetesen fel kellett volna cserélnünk. A bevezetett jelölésekkel a kimeneti függvény:

$$Z = y_1.$$



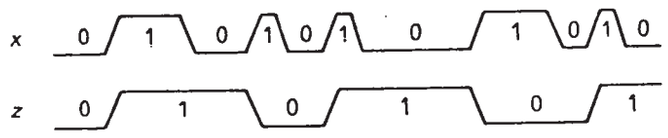
3.92. ábra. Példa analizálandó aszinkron sorrendi hálózatra

y_1y_2 \ x	0	1
00	S_1R_1 00 ; S_2R_2 01	10 ; 00
01	00 ; 01	01 ; 00
11	00 ; 10	01 ; 00
10	00 ; 10	10 ; 00

3.93. ábra. Példa az analízis során kitöltött vezérlési táblára

y_1y_2 \ x	0	1
00	Y_1Y_2 00 ; Z 0	10 0
01	00 0	01 0
11	11 1	01 1
10	11 1	10 1

3.94. ábra. Példa az analízis során létrehozott állapottáblára



3.95. ábra. A példaként analizált aszinkron sorrendi hálózat működésének szemléltetése idődiagrammal

A felírt függvények alapján kitölthetjük a vezérlési táblát (3.93. ábra), amelynek alapján a 3.94. ábrán látható állapottáblához jutunk. Az állapottáblából a hálózat működését egyértelműen kiolvashatjuk. A működést egy feltételezett bemeneti változássorozat hatására az $y_1y_2=00$ stabil állapotból indulva a 3.95. ábra idődiagramja szemlélteti. Akár az állapottábla, akár a bemutatott idődiagram alapján megállapíthatjuk, hogy a Z kimenet értéke mindannyiszor ellenkezőjére változik, valahányszor x értéke 0-ról 1-re változik. Az ilyen működés következményeként egy vizsgált időtartományban Z értékében feleannyi 0—1 átmenet észlelhető, mint amennyi az x érték változása során előfordul. Emiatt az ilyen működésű hálózatokat 2-es frekvenciaosztóknak nevezik és előnyösen alkalmazhatók számláló tulajdonságú hálózatok felépítésére.

Az analízis lépéseit aszinkron sorrendi hálózatra mutattuk be, de nyilvánvaló, hogy az eljárás lényegében azonos szinkron esetben is. Az eltérés csupán annyi, hogy az állapottáblából a szinkron működésnek megfelelően kell kiolvasni a hálózat által megoldott logikai feladatot.

3.8. Állapot-összevonási eljárások

A sorrendi hálózatok tervezésének első lépése az előzetes állapottábla felvétele. Az előző fejezetben bemutatott egyszerű példán is láttuk, hogy az előzetes állapottábla több állapotot is tartalmazhat, mint amennyi az előírt logikai feladat megoldásához feltétlenül szükséges. Ahhoz, hogy a legegyszerűbben és leggazdaságosabban valósítsuk meg az előírt feladatot megoldó hálózatot, először is a lehető legkevesebb állapotot tartalmazó állapottáblát kell megkeresnünk.

Az állapot fogalmának bevezetésénél említettük, hogy a hálózatot ért megelőző hatások az állapotban tükröződnek, azaz a bemeneti kombináció és az állapot egy adott pillanatban elegendő egy adott hálózat (mely adott leképezéseket valósít meg) jellemzéséhez, kimeneti kombinációjának meghatározásához. Ha egy sorrendi hálózatnak van két olyan állapota, amelyekből kiindulva bármilyen bemeneti kombinációsorozatra megegyező kimeneti kombinációsorozatot kapunk, akkor ez a két állapot a környezet számára a hálózatnak ugyanazt az állapotát jelenti, hiszen azok a bemenetek és kimenetek megfigyelésével nem különböztethetők meg. Az ilyen állapotok egyetlen állapottá vonhatók össze, és az így keletkezett sorrendi hálózat működése a környezet szempontjából pontosan megegyezik az eredeti hálózatéval.

Példaként próbáljunk különbséget találni a 3.43. ábra szerinti és a 3.44. ábra szerinti állapottáblájú hálózatok működése között. Feladatunkat ezután úgy fogalmazhatjuk meg, hogy meg kell keresnünk az előzetes állapottábla valamennyi nem megkülönböztethető állapotát és ezeket össze kell vonni. Állapottáblánkat akkor tekinthetjük minimálisnak, ha valamennyi állapota megkülönböztethető.

További vizsgálatainkat először teljesen specifikált sorrendi hálózatokra korlátozzuk, majd eredményeinket kiterjesztjük nem teljesen specifikált hálózatokra.

A továbbiakban a következő rövidítéseket használjuk:

NMK = nem megkülönböztethető
TSH = teljesen specifikált sorrendi hálózat
NTSH = nem teljesen specifikált sorrendi hálózat.

3.8.1. Állapot-összevonás teljesen specifikált hálózatok esetén

Az eddigiek alapján fogalmazzuk meg pontosan, mikor megkülönböztethető és mikor nem az egy TSH két állapota.

Definíció. *Egy TSH két állapota (a és b) akkor megkülönböztethető, ha létezik olyan bemeneti kombinációsorozat, melyre a hálózat az a és b állapotból kiindulva különböző kimeneti kombinációsorozatot szolgáltat. A két állapot NMK, ha nem létezik ilyen bemeneti kombinációsorozat.*

Vizsgáljuk meg például a 3.96. ábrán látható állapottáblával adott (1) hálózat

y \ x	X^1	X^2
a	az^1	dz^2
b	bz^1	dz^2
c	dz^2	bz^1
d	dz^2	bz^1

(1)

3.96. ábra. Példa a nem megkülönböztethető állapotok megkereséséhez

b és *c* állapotát. Azonnal látható, hogy akár X^1-gyel, akár X^2-vel kezdődő bemeneti kombinációsorozatot feltételezve, már az első bemeneti kombináció hatására különböző kimenetet ad a *b* és *c* állapot.

X^1-re:

b-ből indulva Z^1,

c-ből indulva Z^2

X^2-re:

b-ből indulva Z^2,

c-ből indulva Z^1 adódik.

b és *c* állapotok tehát megkülönböztethetők.

Vizsgáljuk meg most *c* és *d* állapotot. Látható, hogy akár X^1-gyel, akár X^2-vel kezdődik a bemeneti kombinációsorozat, a két állapothoz tartozó kimeneti sorozat ugyanúgy kezdődik; X^1 esetén Z^2-vel, X^2 esetén Z^1-gyel. Az is látható, hogy az első fellépő bemeneti kombináció hatására a hálózat ugyanabba az állapotba kerül, akár *c*-ből, akár *d*-ből indult, mégpedig X^1-re *d*-be, X^2-re *b*-be. Ebből nyilvánvaló, hogy akárhogyan folytatódik is a sorozat, a továbbiakban nem lesz eltérés a kimeneti sorozatban, *c* és *d* állapotok tehát NMK-k.

A TSH-k NMK állapotait ekvivalens állapotoknak nevezik. Az ekvivalencia jelölésére a $\equiv$ szimbólumot fogjuk használni. Definíciónk alapján egy állapottábla valamennyi ekvivalens állapotának megkeresése nagyon nehézkes, hiszen az a bemeneti sorozatok hosszára sem ad korlátot, és a legtöbb esetben az ekvivalencia eldöntése nem olyan egyszerű feladat, mint az (1) hálózat vizsgált állapotainál. Az ott alkalmazott gondolatmenetet azonban általánosíthatjuk és kimondhatunk egy tételt, melynek alapján a keresés könnyebb lesz.

Egy TSH két állapota (a és b) akkor, és csak akkor ekvivalens, ha

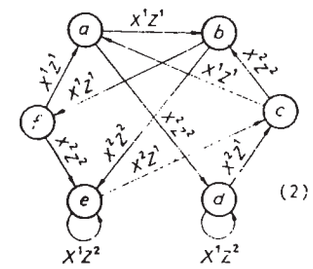
- *akár a, akár b állapotban a hálózatot érő bármely bemeneti kombináció ugyanazt a kimeneti kombinációt hozza létre, és*
- *akár a, akár b állapotból a hálózat bármely bemeneti kombináció hatására ekvivalens állapotba kerül.*

E nyilvánvalóan triviális tétel alapján két állapot ekvivalenciáját a következőképpen vizsgálhatjuk.

Ellenőrizzük, hogy a két vizsgált állapotban a hálózat kimenetei minden bemeneti kombinációra azonosak-e. Ha nem, ez eleve kizárja a két állapot ekvivalenciáját. Ha igen, meg kell vizsgálni minden bemeneti kombinációra a következő állapotok

y \ x	X^1	X^2
a	bz^1	dz^2
b	fz^1	ez^2
c	az^1	bz^2
d	dz^2	cz^1
e	ez^2	cz^1
f	az^1	ez^2

a)



b)

(2)

3.97. ábra. Példa az ekvivalens állapotpárok megkereséséhez

ekvivalenciáját ugyanezzel a módszerrel. Így egy feltételláncot kapunk. Ha olyan feltételhez jutunk, amely a kimenetek alapján nem teljesülhet, ez a lánc valamennyi megelőző feltételének teljesülését kizárja.

Vizsgáljuk meg például a 3.97. ábrán látható (2) hálózat *a* és *c* állapotát. A kimenetek mindkét állapotban

X^1-re Z^1

X^2-re Z^2; az első feltétel teljesül.

Következő állapotaik:

a-ból X^1-re *b*, X^2-re *d*,

c-ből X^1-re *a*, X^2-re *b*.

Feltétel tehát $a \equiv b$ és $d \equiv b$ fennállása.

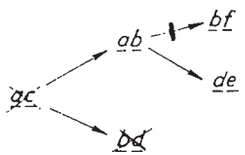
$a \equiv b$ a kimenetek alapján lehetséges, és $b \equiv f$ valamint $d \equiv e$ fennállását követeli meg.

$d \equiv b$ azonban a kimenetek alapján nem lehetséges,

így

$a \equiv c$ sem állhat fenn.

A vizsgálat menetét a következőképpen ábrázolhatjuk:



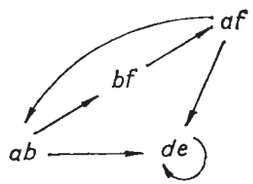
Vizsgáljuk meg most a hálózat *a* és *b* állapotát. Mindkét állapothoz X^1 bemenet mellett Z^1, X^2 bemenet mellett Z^2 kimenet tartozik, az ekvivalencia első feltétele teljesül. X^1 bemenet hatására *a*-ból *b*-be; *b*-ből *f*-be kerül a hálózat, X^2 hatására *a*-ból *d*-be, *b*-ből *e*-be. $a \equiv b$ feltétele tehát az, hogy $b \equiv f$ és $d \equiv e$ fennálljon. Vizsgálatainkat tovább kell folytatni, vajon a $b \equiv f$ és $d \equiv e$ ekvivalenciák fennállnak-e? $b \equiv f$ a kimenetek alapján lehetséges, és csak $a \equiv f$-et feltételezi, mert X^2 hatására mindkét állapotból *e*-be kerülünk és $e \equiv e$ természetesen igaz.

$d \equiv e$ a kimenetek alapján lehetséges, és nem követel új feltételeket a következő állapotokra sem.

Meg kell vizsgálnunk a $b \equiv f$ feltételeként adódott
$a \equiv f$ kapcsolatot.
$a \equiv f$ a kimenetek alapján lehetséges, és $a \equiv b$-t, valamint $e \equiv d$-t követeli meg, melyek egyike sem új feltétel.

Az a és b állapotok ekvivalenciájának vizsgálatát ezzel befejeztük, hiszen minden feltételt megvizsgáltunk és nem jutottunk ellentmondásra. Megállapíthatjuk, hogy $a \equiv b$ fennáll, sőt további ekvivalens állapotokat találtunk, nevezetesen $b \equiv f$, $a \equiv f$, $d \equiv e$ ekvivalenciákat.

A vizsgálat menetét a következőképpen ábrázolhatjuk:



A feltétellánc zárt, ellentmondásmentes, ezért valamennyi benne szereplő ekvivalencia fennáll.

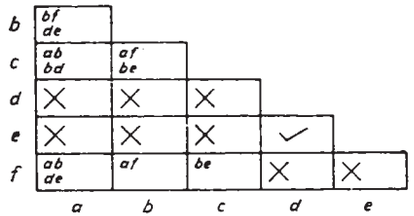
Nem biztos azonban, hogy már megtaláltuk az állapottábla valamennyi ekvivalens állapotpárját. Ebben csak akkor lehetünk biztosak, ha a fenti vizsgálatot valamennyi állapotpárból kiindulva elvégezzük. Látható, hogy ennek során bizonyos állapotpárokat többszörösen vizsgálnánk, hiszen pl. a (2) hálózat *de* állapotpárját már *ab* esetén megvizsgáltuk.

A következőkben ismertetünk egy olyan eljárást, amely alkalmas valamennyi ekvivalens állapotpár megkeresésére anélkül, hogy ismételt vizsgálatokat kellene végezni. Bevezetünk egy célszerű táblázatos ábrázolásmódot, amely megkönnyíti a már megvizsgált állapotpárok nyilvántartását és biztosítja, hogy nem maradnak megvizsgálatlan állapotpárok. Az eljárás szisztematikus, jól algoritmizálható, számítógépes végrehajtásra is alkalmas.

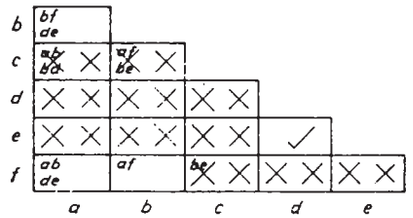
Az eljárás alapgondolata: keressük meg a hálózat valamennyi antivalens (nem ekvivalens) állapotpárját (ezt könnyebb szisztematikusan elvégezni), a fennmaradó állapotpárok ekkor a hálózat ekvivalens állapotpárjai. A módszert a szakirodalomban Paull—Unger-eljárás néven említik. Az antivalens állapotpárok megkeresése a következőképpen történik.

Valamennyi állapotpárt meg kell vizsgálni (ha a hálózat állapotainak száma n, akkor $\binom{n}{2}$ állapotpár van), és külön-külön meg kell jelölni, ha

— a két állapot a kimenetek alapján nem ekvivalens,
— a két állapot ekvivalens,
vagy e döntéseket még nem lehet meghozni, azaz



3.98. ábra. A 3.97. ábrán szereplő állapottáblára vonatkozó lépcsős tábla kitöltésének szemléltetése



3.99. ábra. A lépcsős tábla alapján végezhető ekvivalenciavizsgálati eljárás szemléltetése a 3.97. ábrán szereplő állapottábla összevonásához

— a két állapot ekvivalenciája következő állapotaikra milyen ekvivalenciákat feltételez.

Ehhez célszerű egy ún. *lépcsős táblát* használni, amelyben minden állapotpárnak a tábla egy cellája felel meg. Ha az állapotok nem ekvivalensek, cellájukba $\times$-jel, ha feltétel nélkül ekvivalensek $\surd$ jel, ha ezek egyike sem teljesül, akkor ekvivalenciájuk feltétele írandó. A (2) hálózat kitöltött lépcsős táblája a 3.98. ábrán látható.

Folytatva az eljárást, $\times$ jelet kell írni a lépcsős tábla minden olyan feltételt tartalmazó celljába, amelyben a feltétel teljesülése kizárt amiatt, hogy a feltételt jelentő állapotpároknak megfelelő cellában $\times$ jel van. Ezáltal természetesen $\times$ jelek kerülhetnek olyan cellákba, amelyek kiinduláskor feltételeket tartalmaztak, és így kizáró okává válhatnak újabb feltételek teljesülésének. Ezt a vizsgálatot mindaddig folytatni kell, amíg újabb bejegyzés már nem lehetséges. Célszerű úgy eljárni, hogy kiindulunk az első olyan cellából, amely $\times$-et tartalmaz, és megvizsgáljuk, hogy a cellának megfelelő két állapot ekvivalenciája mely cellákban szerepel feltételként. Az összes ilyen cellába $\times$-et írunk, hiszen az ezeknek megfelelő állapotpárok ekvivalenciája kizárt. Ezután egy második $\times$-et írunk a kizáró okot képviselő kiindulási cellába, jelezve ezáltal, hogy annak minden hatását figyelembe vettük. A vizsgálatot addig kell folytatni, amíg az $\times$-et tartalmazó cellák mindegyikébe bekerül a második $\times$ jel. A 3.99. ábra alapján követhetjük az eljárást a (2) állapottáblára vonatkozóan. Eszerint az $a \equiv b$, $a \equiv f$, $b \equiv f$ és $d \equiv e$ ekvivalenciakapcsolatok állnak fenn.

Az állapottábla egyszerűsítése során az a cél, hogy a lehető legtöbb állapotot tudjuk egyetlen állapottá összevonni. Ez az ekvivalens állapotpárok ismeretében nem mindig ilyen egyszerű feladat, mint a (2) állapottábla esetén. A további vizsgálatok előtt bevezetünk két új fogalmat, az ekvivalenciaosztály és a maximális ekvivalenciaosztály fogalmát.

Egy sorrendi hálózat páronként ekvivalens állapotainak halmazát ekvivalenciaosztálynak nevezik. A (2) hálózat ekvivalenciaosztályai pl.: (ab), (af), (bf), (de), (abf).

Egy sorrendi hálózat valamely ekvivalenciaosztálya maximális, ha egyetlen újabb állapottal sem bővíthető. Másképpen: ha a hálózatnak nincs olyan állapota, amely az adott osztályban nem szerepel és az adott osztály bármelyik állapotával ekvivalens. A (2) hálózat maximális ekvivalenciaosztályai: (*abf*), (*de*), (*c*).

E fogalmak felhasználásával az állapottábla összevonásának feladatát a következőképpen fogalmazhatjuk meg. Meg kell keresni a maximális ekvivalenciaosztályokat, és ezek mindegyikét egyetlen állapottal helyettesítve elő kell állítani az összevont állapottáblát.

A következőkben a bizonyítás mellőzésével bemutatunk egy szisztematikus módszert, amellyel az ekvivalens állapotpárokból — azaz valamennyi kételemű ekvivalenciaosztályból — előállíthatók a maximális ekvivalenciaosztályok. A módszer számítógépre is könnyen programozható.

A módszer alapgondolata: tételezzük fel, hogy a maximális ekvivalenciaosztály valamennyi állapotot tartalmazza. Ezután sorra zárjuk ki az antivalens állapotpárok egy osztályba tartozását. Ha ezt minden antivalens állapotpárra megtettük, a maradó ekvivalenciaosztályok szolgáltatják a hálózat maximális ekvivalenciaosztályait. Példaként induljunk ki a 3.99. ábrán látható lépcsős táblából. Tételezzük fel, hogy az összes állapot ekvivalens, vagyis egyetlen ekvivalenciaosztályt alkotnak:

(*abcdef*).

A lépcsős tábla első oszlopa szerint viszont az *a* állapot nem ekvivalens *c*-vel, *d*-vel és *e*-vel. Emiatt a feltételezett ekvivalenciaosztályból ki kell hagynunk az *a* állapotot:

(*bcdef*).

A lépcsős tábla első oszlopa azt is jelzi, hogy az *a* állapot ekvivalenciája nincs kizárva *b*-vel és *f*-fel, azaz nincs okunk kizárni az (*abf*) ekvivalenciaosztály létezését. Így az első oszlop alapján a kiinduláskor feltételezett egyetlen ekvivalenciaosztály helyett az alábbi kettő létezésére nem találtunk kizáró okot:

(*bcdef*)(*abf*).

A továbbiakban ugyanezt a vizsgálatot kell elvégeznünk rendre a lépcsős tábla oszlopai alapján a mindenkor még feltételezhető ekvivalenciaosztályok mindegyikére.

A második oszlop szerint *b* állapot nem ekvivalens *c*-vel, *d*-vel és *e*-vel. Ezért *b*-t ki kell hagyni minden olyan ekvivalenciaosztályból, amelyben ezekkel az állapotokkal együtt szerepel:

(*cdef*)(*abf*).

Nincs kizárva viszont a második oszlop alapján *b* és *f* ekvivalenciája, ezért (*bf*) ekvivalenciaosztály létezését fel kell tételeznünk:

(*cdef*)(*abf*)(*bf*).

A további vizsgálatokban azonban (*bf*) ekvivalenciaosztályt figyelmen kívül hagyhatjuk, mert (*abf*)-hez képest nem tartalmaz új információt. Ez úgy is magyarázható, hogy (*bf*) nem maximális ekvivalenciaosztály, mert (*abf*) feltételezése esetén az *a* állapottal bővíthető. Mivel célunk a maximális ekvivalenciaosztályok megkeresése, ezért figyelmen kívül hagyható a további vizsgálatokban minden olyan ekvivalenciaosztály, amelyről eleve tudjuk, hogy nem maximális:

(*cdef*)(*abf*).

A lépcsős tábla harmadik oszlopa azt zárja ki, hogy a *c* állapot ekvivalens legyen *d*-vel, *e*-vel és *f*-fel. Ezért az előbbi gondolatmenetet követve a feltételezhető ekvivalenciaosztályok:

(*def*)(*abf*)(*c*).

A (*c*) osztály azért áll csupán egyetlen elemből, mert a lépcsős tábla eddig figyelembe vett oszlopai már kizárják, hogy a *c* állapot bármelyik másik állapottal ekvivalens legyen.

A negyedik oszlop *d* és *f* állapotok ekvivalenciáját kizárja, de *d* és *e* állapotokét nem:

(*ef*)(*de*)(*abf*)(*c*).

Az utolsó oszlop szerint *e* és *f* állapotok nem lehetnek azonos ekvivalenciaosztályban:

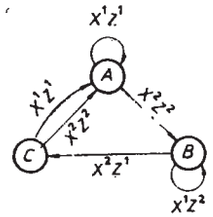
(*f*)(*e*)(*de*)(*abf*)(*c*).

Az (*e*) osztály nyilvánvalóan elhagyható, hiszen (*de*) létezése miatt nem maximális. Ezzel az eljárás végére értünk és a maximális ekvivalenciaosztályok:

(*abf*)(*de*)(*c*).

Az összevont állapottáblához ezek után úgy juthatunk el, hogy a maximális ekvivalenciaosztályok mindegyikét egyetlen állapotnak tekintjük. Az így bevezetett állapotokból kiindulva minden egyes bemeneti kombináció esetén rendre meg kell határozni, hogy milyen következő állapotoknak kell fellépniük ahhoz, hogy az előzetes állapottábla szerinti működést kapjuk. E célból az előzetes állapottáblán ki kell indulnunk a maximális ekvivalenciaosztályok akármelyik állapotából, és meg kell állapítanunk, hogy az egyes bemeneti kombinációk hatására létrejövő következő állapotok melyik maximális ekvivalenciaosztályban szerepelnek. Ezt az osztályt kell egyetlen állapotnak tekintve következő állapotként feltüntetni az összevont állapottáblán az illető bemeneti kombináció rovatában. Egy ekvivalenciaosztály összes állapota azonos kimeneti kombinációval szerepel az azonos bemeneti kombinációhoz tartozó rovatokban, ezért az összevont állapottáblán a kimeneti kombinációkat is könnyen meghatározhatjuk az előzetes állapottáblából kiindulva a maximális ekvivalenciaosztályok alapján. Példánkban vezessük be a maximális ekvivalencia-

y \ x	x^1	x^2
A	AZ^1	BZ^2
B	BZ^2	CZ^1
C	AZ^1	AZ^2



3.100. ábra. A 3.97. ábrán szereplő előzetes állapottáblának megfelelő összevont állapottábla

osztályoknak megfelelő állapotokra az alábbi jelöléseket:

(abf): A,

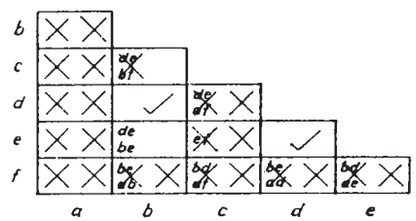
(de): B,

(c): C.

Az összevont állapottábla kitöltése a 3.100. ábrán követhető. Például az A jelű sor X^2 rovatába azért kellett BZ^2-t írnunk, mert a 3.97. ábrán látható, hogy az A-t alkotó a, b és f sorok dZ^2-t illetve, eZ^2-t tartalmaznak az X^2 rovatban, ami bevezetett jelölésünk értelmében B-t írja elő következő állapotként, a kimeneti kombináció pedig nyilvánvalóan Z^2 lesz.

y \ x	x^1	x^2
a	cZ^1	eZ^2
b	eZ^2	bZ^1
c	dZ^2	fZ^1
d	eZ^2	dZ^1
e	dZ^2	eZ^1
f	bZ^2	aZ^1

a)



b)

(abcdef)

(bcdef)(a)

(cdef)(bde)(a)

(def)(c)(bde)(a)

(ef)(de)(c)(bde)(a)

(e)(f)(c)(bde)(a)

(a)(bde)(c)(f)

A B C D

c)

y \ x	x^1	x^2
A	CZ^1	BZ^2
B	BZ^2	BZ^1
C	BZ^2	DZ^1
D	BZ^2	AZ^1

d)

3.101. ábra. Példa az állapotösszevonási-eljárás alkalmazására teljesen határozott sorrendi hálózat esetén (*a*) az előzetes állapottábla, *b*) a lépcsős tábla, *c*) a maximális ekvivalenciaosztályok meghatározása, *d*) az összevont állapottábla)

Egyszerű példánkban az állapot-összevonási lehetőségek az előzetes állapottábla alapján közvetlenül is szembetűnőek voltak, és a bemutatott szisztematikus eljárás alkalmazása nélkül is felépíthettük volna a 3.100. ábrán látható összevont állapottáblát.

Foglaljuk össze ezek után a minimális számú sort tartalmazó összevont állapottábla előállításának lépéseit teljesen határozott sorrendi hálózatok esetén:

1. Valamennyi ekvivalens és antivalens állapotpár megkeresése lépcsős tábla felhasználásával.
2. A lépcsős tábla alapján a maximális ekvivalenciaosztályok meghatározása.
3. A maximális ekvivalenciaosztályoknak egy-egy állapotot megfeleltetve az összevont állapottábla kitöltése.

A lépések mindegyike szisztematikus, számítógéppel elvégezhető. A 3.101. ábrán újabb egyszerű példát láthatunk az eljárás alkalmazására.

3.8.2. Az állapotekvivalencia tulajdonságai

Vizsgáljuk meg, hogy az ekvivalencia, mint egy TSH állapotai közötti reláció, és a maximális ekvivalenciaosztályok milyen tulajdonságúak. Ezen tulajdonságok ismerete segítséget nyújt az egyszerűbb esetek gyors áttekintésében, más módszerek megértésében, használatában.

Az ekvivalenciareláció tulajdonságai a következők:

1. Minden állapot ekvivalens önmagával: $a \equiv a$. Ezt a tulajdonságot *reflexivitásnak nevezik*.
2. Az ekvivalencia kölcsönös két állapot között; ha $a \equiv b$, akkor $b \equiv a$. Ez a tulajdonság a *szimmetria*.
3. Ha $a \equiv b$ és $b \equiv c$, akkor $a \equiv c$, azaz az ekvivalencia *tranzitív*.

Ezek a tulajdonságok a definíció alapján könnyen beláthatók. A tranzitivitás miatt igaz a következő: ha találunk egy olyan a állapotot, amely egy ekvivalenciaosztály valamelyik állapotával ekvivalens, akkor a-val és az a-val ekvivalens valamennyi állapottal bővíthetjük ezt az osztályt.

A maximális ekvivalenciaosztályoknak is megállapíthatjuk néhány triviális tulajdonságát.

1. A maximális ekvivalenciaosztályok az előzetes állapottábla állapotaiból alkotott halmaz részhalmazai.
2. Az előzetes állapottábla minden egyes állapota megtalálható valamelyik maximális ekvivalenciaosztályban.
3. A tranzitív tulajdonságból következően egy TSH maximális ekvivalenciaosztályainak nem lehetnek közös állapotaik, azaz minden állapot csak egy osztályban szerepel. A maximális ekvivalenciaosztályok tehát *diszjunkt* részhalmazai az összes állapot halmazának.

A fenti tulajdonságok alapján megállapíthatjuk, hogy a maximális ekvivalencia-osztályok az összes állapot halmazának egy ún. *partícióját* alkotják. A partíció halmazelméleti fogalom; egy halmaz elemeinek olyan csoportosítását jelenti, hogy minden egyes elem szerepeljen egy, és csakis egy csoportban, vagy más szóval *partíció-blokkban*. A partíció fogalmát a továbbiakban számos esetben fel fogjuk használni az egyes eljárások magyarázatában.

3.8.3. Állapot-összevonás nem teljesen specifikált hálózatok esetén

Eddigi ismereteink alapján tetszőleges teljesen határozott sorrendi hálózat esetén szisztematikus lépésekben meg tudjuk határozni a minimális számú sorral rendelkező összevont állapottáblát. A továbbiakban látni fogjuk, hogy nem teljesen határozott hálózatok esetén az eljárás módosul és általában nem nélkülözhetjük a próbálgatásos, intuitív lépéseket.

A NTSH-k kimeneti kombinációja vagy következő állapota bizonyos állapotokban és bemeneti kombinációk esetén nincs előírva. Ennek oka többféle lehet, pl. a működési feltételekből tudjuk, hogy bizonyos állapotokban bizonyos bemeneti kombinációk nem léphetnek fel, vagy egyszerűen közömbös a hálózat viselkedése valamilyen állapotban valamilyen bemeneti kombinációra.

Az állapot-összevonás célja ilyenkor az, hogy olyan összevont állapottáblához jussunk, amelynek a lehető legkevesebb sora van és az előzetes állapottábla specifikált bejegyzéseivel megegyező működést olvashatunk ki belőle. Az előzetes állapottáblán nem specifikált (közömbös) bejegyzéseket az összevont állapottáblán természetesen tetszőlegesen tehetjük határozottá (rögzíthetjük) az állapot-összevonási lehetőségek növelése érdekében. Felmerülhet a kérdés, hogy vajon NTSH-k esetén mindig eljutunk-e a legkevesebb sorú összevont állapottáblához a következő gondolatmenettel:

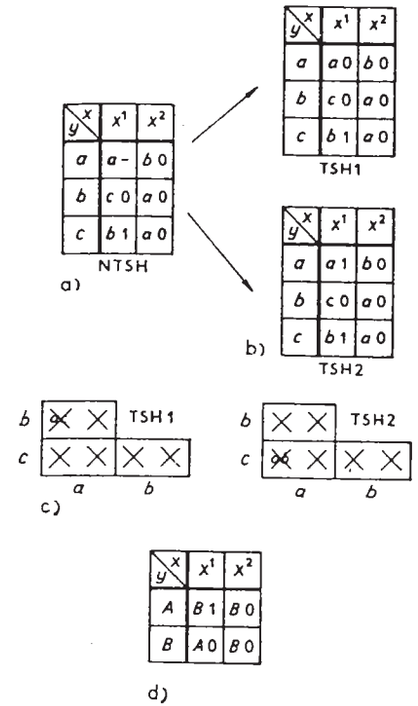
— Rögzítsük az előzetes állapottábla összes közömbös bejegyzését az összes lehetséges módon. Minden egyes ilyen rögzítéssel egy-egy TSH előzetes állapottáblájához jutunk.

— Az így kapott TSH-k mindegyikére hajtsuk végre a megismert állapot-összevonási eljárást.

— A kiadódó összevont állapottáblák közül válasszuk ki azt, amelynek a legkevesebb sora van. Kérdés, hogy ezzel biztosan meghatároztuk-e a kiindulási NTSH előzetes állapottáblájának megfelelő, legegyszerűbb összevont állapottáblát.

A nem teljesen határozott kombinációs hálózatok egyszerűsítésekor tulajdonképpen a fenti gondolatmenetet alkalmaztuk a közömbös bejegyzések rögzítésére, bár a gyakorlati végrehajtás lépései nem mindig követték azt. Kombinációs hálózatok esetében tehát eljuthattunk a minimális megoldáshoz ezzel a gondolatmenettel.

Egy egyszerű példán bemutatjuk, hogy sorrendi hálózatok esetében a fenti gondolatmenet nem vezet el feltételenül a lehető legkevesebb sorú összevont állapottáblához. A 3.102*a)* ábrán adott egy NTSH előzetes állapottáblája, amelyben egyetlen



3.102. ábra. Példa a közömbös bejegyzések rögzítésének az állapot-összevonási lehetőségekre gyakorolt hatására nem teljesen határozott sorrendi hálózatok esetén. *(a)* nem teljesen határozott előzetes állapottábla, *b)* a közömbös bejegyzés lehetséges rögzítéseivel előállítható teljesen határozott előzetes állapottáblák, *c)* a rögzítés után kiadódó lépcsős táblák, *d)* az előzetes állapottábla specifikált bejegyzéseivel megegyező működést adó kétsoros állapottábla)

közömbös bejegyzés van. Így kétféle módon tehetjük határozottá az állapottáblát, vagyis két TSH előzetes állapottábláját állíthatjuk elő. Ezek láthatók a 3.102*b)* ábrán. A 3.102*c)* ábrán követhető a lépcsős táblák kitöltése a két TSH-ra vonatkozóan. Látható, hogy nem adódott állapot-összevonási lehetőség egyik teljesen határozott előzetes állapottáblán sem. Ebből azonban nem vonhatjuk le azt a következtetést, hogy a kiindulási, nem teljesen határozott állapottáblán sincs állapot-összevonási lehetőség. A 3.102*d)* ábrán ugyanis bemutattunk egy kétsoros állapottáblát, amelyből minden bemeneti kombinációsorozat esetén a 3.102*a)* ábrán szereplő előzetes állapottábla specifikált bejegyzéseivel megegyező működést, azaz kimeneti kombinációsorozatot olvashatunk ki. Ennek ellenőrzését az olvasóra bízzuk. Egyszerű példánk bizonyítja, hogy a fenti gondolatmenet, vagyis a közömbös bejegyzések előzetes rögzítése nem mindig teszi lehetővé a legkevesebb sort tartalmazó összevont állapottábla meghatározását.

Egyszerűsítési kísérletünk során azért nem juthattunk erre az eredményre, mert a közömbös bejegyzés határozottá tételével előírtuk, hogy a hálózat ugyanazt a kimeneti kombinációt szolgáltassa valahányszor egy bemeneti kombinációsorozat közben az *a* állapotba kerül és X^1 bemeneti kombinációt észlel. Ez nyilvánvalóan nem volna szükségszerű, hiszen a sorrendi hálózatok működése során a közömbös bejegyzés a bemeneti kombinációk sorrendjétől függően más-más rögzített értéket kaphat.

A továbbiakban bemutatjuk, hogy a közömbös bejegyzések ilyen értelmezésben növelhetik az állapot-összevonási lehetőségeket.

Fogalmazzuk meg az állapotok megkülönböztethetőségének feltételét NTSH esetében a közömbös bejegyzések fenti értelmezését szem előtt tartva.

A megkülönböztethetőség definíciója előtt vezessük be a *specifikációs bemeneti kombinációsorozat* fogalmát: *Valamely bemeneti kombinációsorozat specifikációs egy sorrendi hálózat egy állapotára, ha lejátszódása során az állapotból kiindulva a hálózat valamennyi állapotátmenete és kimeneti kombinációja specifikált.*

A megkülönböztethetőség definícióját ezek után a következőképpen fogalmazhatjuk meg: *Egy NTSH két állapota akkor, és csak akkor megkülönböztethető, ha létezik legalább egy olyan mindkét állapotra specifikációs bemeneti kombinációsorozat, amelyre a két állapotból kiindulva szolgáltatott két kimeneti kombinációsorozat legalább egy bemeneti kombináció esetén különbözik egymástól. A két állapot nem megkülönböztethető, ha ilyen specifikációs bemeneti kombinációsorozat nem létezik.*

Figyeljük meg, hogy a definícióban a specifikációs bemeneti kombinációsorozat fogalmának a felhasználása azt jelenti, hogy a megkülönböztethetőséghez olyan helyen kell különböznie a két kimeneti kombinációsorozatnak, ahol mindkettő specifikált.

Az NTSH nem megkülönböztethető (NMK) állapotait *kompatibilis* (összeegyeztethető) *állapotoknak* nevezik. A kompatibilitás jelölésére a $\sim$ szimbólumot használjuk.

A TSH-k esetében követett gondolatmenethez hasonlóan az NTSH-kra is megfogalmazhatunk egy tételt, amelynek segítségével a kompatibilis állapotpárok egyszerűen kereshetők meg.

Egy NTSH két állapota akkor, és csak akkor kompatibilis, ha

— a két állapothoz tartozó kimeneti kombinációk megegyeznek minden olyan bemeneti kombináció esetén, amely mellett mindkét állapothoz specifikált kimeneti kombinációk tartoznak;

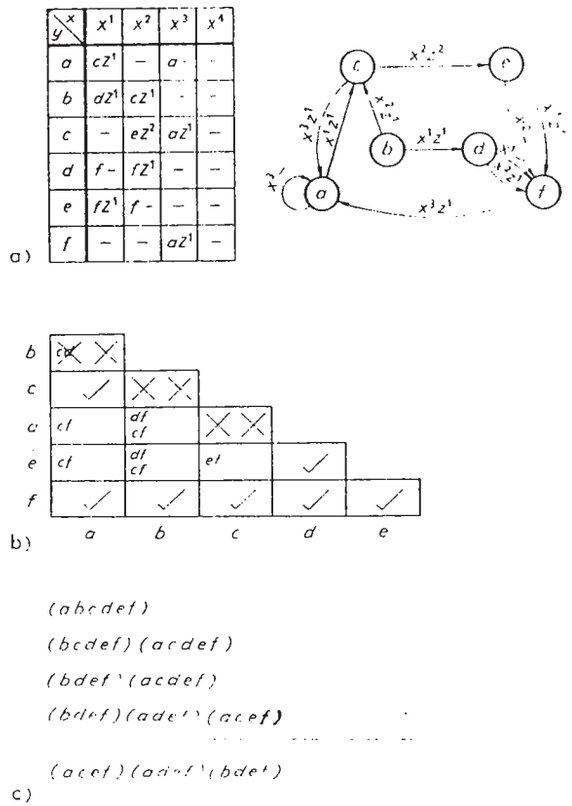
— a két állapotból kiindulva bármely bemeneti kombináció kompatibilis következő állapotokba vezet, ha mindkét következő állapot specifikált.

A kompatibilitás definíciójából a fenti tétel egyértelműen következik. A tétel alapján a kompatibilis állapotpárok megkeresésére is alkalmazhatjuk a TSH esetén megismert lépcsős táblát. Az ekvivalenciaosztályokhoz hasonlóan a kompatibilitási és a maximális kompatibilitási osztályok is definiálhatók.

Egy sorrendi hálózat páronként kompatibilis állapotainak halmazát kompatibilitási osztálynak nevezik.

Egy kompatibilitási osztály maximális, ha nem bővíthető egyetlen újabb állapottal sem.

A maximális kompatibilitási osztályokat ugyanolyan módszerrel kereshetjük meg a kitöltött lépcsős tábla alapján, mint TSH-k esetén a maximális ekvivalenciaosztályokat. Példaként a 3.103. ábrán követhetjük az eljárást. Szembetűnő, hogy a kapott maximális kompatibilitási osztályok — ellentétben a maximális ekvivalenciaosztályokkal — nem diszjunkt részhalmazai a kiindulási állapotnak. Abból, hogy egy



3.103. ábra. Példa a maximális kompatibilitási osztályok meghatározására. (*a*) az előzetes állapottábla, *b*) a lépcsős tábla, *c*) a maximális kompatibilitási osztályok meghatározásának lépései)

állapot több osztályban is szerepel, kompatibilitás esetén nem következik az illető osztályok összevonhatósága egyetlen kompatibilitási osztállyá. Ennek oka az, hogy a kompatibilitás, mint az NTSH állapotai közötti reláció — ellentétben a TSH állapotai közötti ekvivalenciával — nem tranzitív tulajdonságú. Példánkban megfigyelhetjük, hogy $a \sim c$ és $a \sim d$, de ebből nem következik, hogy $c \sim d$, hiszen az előzetes állapottáblán (3.103*a*) ábra) a c és d jelű sorok az X^2 rovatban egymásnak ellentmondó kimeneti kombinációkat tartalmaznak. Az a sor X^2 rovatában levő közömbös bejegyzést ezért különbözőképpen kellene rögzíteni az ac és az ad összevonásokhoz.

Mivel tehát a kompatibilitás nem tranzitív, egy kompatibilitási osztály csak akkor bővíthető újabb állapottal, ha az az osztály valamennyi állapotával kompatibilis.

Az eddigiekben láttuk, hogy a NTSH-knál az állapotok megkülönböztethetetlensége a kompatibilitásrelációval fejezhető ki. A maximális kompatibilitási osztályok megadják az összevonható állapotokat. Vizsgáljuk meg célravezető-e, ha a maximális kompatibilitási osztályoknak feleltetjük meg az összevont állapottábla állapotait. TSH esetén ugyanis ezt a gondolatmenetet követve megkaptuk a minimális számú

sort tartalmazó összevont állapottáblát. Az ekvivalencia fogalmából következően TSH esetében csak egyetlen ilyen állapottábla létezett. Látni fogjuk, hogy egy NTSH-hoz nem feltétlenül csak egyetlen minimális számú állapottal rendelkező állapottábla határozható meg.

Feleltessük meg a 3.103. ábrán meghatározott maximális kompatibilitási osztályoknak az összevont állapottábla állapotait az alábbi módon:

$(acef)$: A,

$(adef)$: B,

$(bdef)$: C.

Az összevont állapottábla kitöltéséhez vegyünk fel egy segédtáblázatot, amely az előzetes állapottábla alapján határozható meg, ha az állapotokat az összevont állapottábla állapotai szerint csoportosítjuk. A 3.104. ábrán követhető a táblázat felépítése a választott A, B és C állapotoknak megfelelően.

Az X^4 bemeneti kombinációnak megfelelő oszlopokat az állapot-összevonás során figyelmen kívül hagyhatjuk, hiszen csupa közömbös bejegyzést tartalmaznak az adott feladatban.

Az összevont állapottáblát ezek után a következő gondolatmenettel tölthetjük ki:

— Az összevont állapotnak képviselnie kell mindazokat az állapotokat, amelyek összevonásával keletkezett. Ezért a kimeneti kombinációkat bemeneti kombinációnként mindig arra az értékre kell rögzíteni, amely az összevonandó állapotok közül akár csak egyhez is specifikált az illető bemeneti kombináció mellett.

— Egy összevont állapotnak megfelelő kompatibilitási osztály bármelyik két állapotából kiindulva a következő állapotoknak bemeneti kombinációnként kompatibilisaknak kell lenniük egymással, azaz valamelyik kompatibilitási osztály részhalmazát kell alkotniuk. Az összevont állapottáblán a következő állapotot tehát mindig úgy kell megválasztani, hogy az olyan kompatibilitási osztálynak feleljen meg, amely tartalmazza ezt a részhalmazt.

		Következő állapotok			Kimenetek		
	y \ x	x^1	x^2	x^3	x^1	x^2	x^3
A	a	c	–	a	z^1	–	–
	c	–	e	a	–	z^2	z^1
	e	f	f	–	z^1	–	–
	f	–	–	a	–	–	z^1
B	a	c	–	a	z^1	–	–
	d	f	f	–	–	z^1	–
	e	f	f	–	z^1	–	–
	f	–	–	a	–	–	z^1
C	b	d	c	–	z^1	z^1	–
	d	f	f	–	–	z^1	–
	e	f	f	–	z^1	–	–
	f	–	–	a	–	–	z^1

3.104. ábra. Segédtáblázat az összevont állapottábla kitöltéséhez a 3.103. ábrán bemutatott feladatra vonatkozóan, ha a maximális kompatibilitási osztályoknak feleltetjük meg az összevont állapottábla állapotait

y \ x	x^1	x^2	x^3
A	AZ^1	$-Z^2$	A vagy BZ^1
B	AZ^1	$-Z^1$	A vagy BZ^1
C	B vagy CZ^1	AZ^1	A vagy BZ^1

3.105. ábra. Az összevont állapottábla a 3.103. ábrán bemutatott feladatra vonatkozóan, ha a maximális kompatibilitási osztályoknak feleltetjük meg az összevont állapottábla állapotait

A fenti gondolatmenet szükségessége egyértelműen következik az állapot-összevonás céljából és a kompatibilitás fogalmából.

A 3.105. ábrán látható a választott A, B és C állapotok alapján a fentiek szerint kitöltött összevont állapottábla. Például az A állapotból X^1 hatására a c és f állapotokat egyaránt tartalmazó összevonás utáni állapotba kell jutnia a hálózatnak. Ez legegyszerűbben a 3.104. ábrán látható segédtáblázatból olvasható ki, de könnyen megállapítható az előzetes állapottábla alapján is. A (cf) részhalmazt a választott A, B és C állapotok közül csak A tartalmazza. Ezért írtunk az összevont állapottábla A jelű sorának X^1 jelű rovatába A-t következő állapotként. Ebben a rovatban a kimeneti kombináció Z^1, hiszen a c és f állapotok mellett egyaránt ez szerepel. Ha az A állapotból X^2 hatására indulunk ki, akkor az összevonás utáni következő állapotnak tartalmaznia kell az e és f állapotokat (3.104. ábra). Mivel az (ef) részhalmazt az A, B és C állapotok mindegyike tartalmazza, ezért az összevont állapottábla A jelű sorának X^2 rovatába bármelyiket beírhatjuk következő állapotként. Ha feltételezhetjük, hogy a megvalósított hálózatnak a működése során nem alakul ki az A, B és C állapotoktól eltérő állapota, akkor ebbe a rovatba közömbös bejegyzést tehetünk a következő állapot helyére. Az állapotkódolás megválasztásakor keletkező fel nem használt kódok (szekunder kombinációk) szintén kialakulhatnak állapotként a hálózat működése során. Például az adott esetben az A, B és C állapotok kódolásához legalább két szekunder változóra van szükség és az így előállítható négy kódból csak hármat használunk. Ha a kódokat az alábbi módon rendeljük az állapotokhoz:

	y_1	y_2
A	0	0
B	0	1
C	1	0

akkor az $y_1y_2=11$ szekunder változókombináció kihasználatlan. Ha a megvalósított hálózatban valamilyen módon fellép az $y_1y_2=11$ kombináció, akkor a hálózat további működését már nem követhetjük az összevont állapottábla alapján. hanem csakis a megvalósított hálózat analízise útján meghatározható kódolt állapottábla alapján. Ez utóbbinak az $y_1y_2=11$ sorában szereplő következő állapotok a

tervezés során — ezen belül a függvények egyszerűsítésekor — adódnak ki és semmiféle kapcsolatban nincsenek a megoldandó logikai feladattal (hacsak erre külön nem ügyelünk). Ezért mindenképpen célszerű elkerülni, hogy a hálózat működése során az $y_1y_2=11$ szekunder változókombináció kialakulhasson. Ha az A jelű sor X^2 rovatába közömbös bejegyzést teszünk a következő állapot helyére, akkor a függvények egyszerűsítésekor $Y_1Y_2=11$ is rögzítődhet ezen a helyen (hacsak erre külön nem ügyelünk). Így tehát a közömbös bejegyzés a megvalósított hálózat hibás működését is okozhatja. A fel nem használt kódok értelmezésével később részletesebben foglalkozunk. Ezúttal csak az állapot-összevonás formális megoldását tartjuk szem előtt, ezért a szóban levő rovatba a 3.105. ábrán közömbös bejegyzést tettünk a következő állapot helyére. Fentiek értelmében azonban tudnunk kell, hogy a megvalósítás szempontjából helyesebb volna az „A vagy B, vagy C" beírás. A kimeneti kombináció értéke ebben a rovatban azért Z^2, mert az e következő állapot mellett ez van specifikálva. Így az f következő állapot mellett szereplő közömbös bejegyzés Z^2-re rögzítődik.

Az A jelű sor X^3 rovatában fel sem merülhet a közömbös bejegyzés következő állapotként, mert bár a 3.104. ábra alapján akár A-t, akár B-t írhatunk, de C-t nem, hiszen ez utóbbi nem tartalmazza az a állapotot.

Az összevont állapottábla többi rovatát is a fenti gondolatmenettel tölthetjük ki.

A 3.105. ábrán szereplő összevont állapottáblát megvizsgálva megállapíthatjuk, hogy az nem tartalmaz kompatibilis állapotokat. Ennek ellenére kimutatható, hogy a sorok számát tovább lehet csökkenteni, vagyis nem jutottunk el a legegyszerűbb összevont táblához a maximális kompatibilitási osztályokhoz rendelve az összevont állapotokat. Ennek szemléltetése céljából ne vonjuk össze az egyes maximális kompatibilitási osztályoknak megfelelő valamennyi állapotot, hanem az osztályokból bizonyos állapotok, esetleg teljes osztályok elhagyásával érjük el, hogy az előzetes állapottábla állapotainak mindegyike csak egyszer szerepeljen, vagyis az osztályokat tegyük diszjunktakká. Válasszuk ki például az $(acef)$ osztályt és a $(bdef)$-ből (ef) elhagyásával kapott (bd) osztályt. Így az előzetes állapottábla minden egyes állapota szerepel a választott két kompatibilitási osztály valamelyikében. Feleltessük meg e két osztálynak az összevont állapottábla állapotait D és E jelölésekkel az alábbi módon:

$(acef)$: D,

(db): E.

Kíséreljük meg kitölteni az összevont állapottáblát ilyen összevont állapotok feltételezésével. A kitöltéshez használható segédtáblázat a 3.106. ábrán látható. Határozzuk meg például az E állapotot követő állapotot az X^1 bemeneti kombináció hatására. A segédtáblázatból kiolvashatjuk, hogy a következő állapotnak egyaránt kellene tartalmaznia a d és az f állapotot. A (df) részhalmazt azonban sem a D, sem az E, tehát a választott összevont állapotok egyike sem tartalmazza. Így az E állapotot követő

		Következő állapot			Kimenet		
	$y \backslash x$	x^1	x^2	x^3	x^1	x^2	x^3
D	a	c	–	a	Z^1	–	–
	c	–	e	a	–	Z^2	Z^1
	e	f	f	–	Z^1	–	–
	f	–	–	a	–	–	Z^1
E	b	d	c	–	Z^1	Z^1	–
	d	f	f	–	–	Z^1	–

3.106. ábra. Segédtáblázat az összevont állapottábla kitöltéséhez a 3.103. ábrán bemutatott feladatra vonatkozóan, ha az $(acef)$ és (bd) kompatibilitási osztályoknak feleltetjük meg az összevont állapottábla állapotait

állapotot nem tudjuk meghatározni, emiatt a választott D és E állapotok önmagukban nem alkalmasak az összevont állapottábla ellentmondásmentes kitöltésére. Ennek oka az, hogy — amint az a lépcsős táblán megfigyelhető — a b és d állapotok kompatibilitásának a d és f állapotok kompatibilitása a feltétele. Ezért b-t és d-t csak akkor vonhatjuk össze, ha d-t és f-et is összevonjuk. A választott D és E állapotok létrehozásakor viszont b és d összevonását úgy végeztük el, hogy f-et d helyett a-val, c-vel és e-vel vontuk össze. Ezáltal az előzetes állapottábla d és f sorainak közömbös bejegyzéseit úgy rögzítettük, hogy a d és f állapotok összevonhatóságát kizártuk. Emiatt viszont (bd) sem tekinthető kompatibilitási osztálynak, vagyis az E állapot megválasztása helytelen volt. Általánosan fogalmazva azt mondhatjuk, hogy a választott $(acef)$ és (bd) kompatibilitási osztályok nem alkotnak ún. *zárt* halmazt. A kompatibilitási osztályok zártságának definícióját a következőképpen adhatjuk meg:

A kompatibilitási osztályok egy 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.

Láttuk, hogy NTSH esetében az összevont állapottábla állapotait csakis olyan kompatibilitási osztályoknak feleltethetjük meg, amelyek zárt halmazt alkotnak. A legkevesebb sort tartalmazó összevont állapottáblához úgy juthatunk el, hogy a legkevesebb kompatibilitási osztályt tartalmazó zárt halmaz elemeihez rendeljük hozzá az összevont állapottábla állapotait. Általában több ilyen halmaz létezhet és megkeresésüket az alábbi — próbálgatást is igénylő — lépésekben végezhetjük.

1. A maximális kompatibilitási osztályok közül válasszuk ki a lehető legkevesebbet úgy, hogy az előzetes állapottábla minden egyes állapota legalább egy osztályban szerepeljen. Másképpen: fedjük le az állapotokat minimális számú maximális kompatibilitási osztállyal.

2. Vizsgáljuk meg, hogy zárt halmazt kaptunk-e. Ha igen, hagyjuk ki a 3. lépést.

3. Vizsgáljuk meg, hogy a több osztályban szereplő állapotoknak egyes osztályokból történő kihagyásával zárttá tehető-e a halmaz. (Természetesen minden egyes állapotnak legalább egy osztályban meg kell maradnia.) Ha igen, akkor alakítsuk át ennek megfelelően a kompatibilitási osztályokat. Ha nem, akkor egészítsük ki a halmazt a zártságot biztosító osztályokkal.

4. Hagyjuk el az osztályokból azokat az állapotokat, amelyek a zártság fenntartásával elhagyhatók (természetesen az így létrejövő osztályok halmazának továbbra is le kell fednie az összes állapotot).

Az 1. lépésben megfogalmazott lefedési feladat megoldható a kombinációs hálózatok egyszerűsítésekor megismert prímimplikánstábla-lefedési módszerrel. Az eltérés csupán annyi, hogy most a mintermeknek az állapotok, a prímimplikánsoknak pedig a maximális kompatibilitási osztályok felelnek meg. A 2. és 3. lépésben leírt vizsgálathoz előnyösen használhatjuk a 3.104. és 3.1[illegible]rán bemutatott felépítésű segédtáblázatokat.

Példaként kövessük a fentiekben összefoglalt lépéseket a 3.103. ábrán bemutatott feladat megoldása céljából. A maximális kompatibilitási osztályokat jelöljük az alábbi módon:

(*acef*): *1*,

(*adef*): *2*,

(*bdef*): *3*.

A prímimplikánstáblához hasonló lefedési tábla kitöltését a 3.107. ábra alapján követhetjük. A tábla alapján megállapíthatjuk, hogy *1* és *3* lényeges maximális kompatibilitási osztályok, hiszen a *b*, illetve a *c* megkülönböztetett állapotot tartalmazzák. Az is megfigyelhető, hogy *1* és *3* egyúttal a legegyszerűbb lefedést is megvalósítja. A zártság vizsgálatát a 3.104. ábrán szereplő segédtáblázat segítségével végezhetjük. Az *A* jelű maximális kompatibilitási osztályt *1*-gyel, a *C* jelűt pedig *3*-mal jelöltük a lefedési eljárás során. A zártság feltétele az X^1 bemeneti kombináció fellépésekor az, hogy *1*-ből kiindulva *c* és *f*, valamint *3*-ból kiindulva *d* és *f* állapotok legyenek együtt valamelyik osztályban. Ezeket a feltételeket *1* és *3* kielégíti. Az X^2 bemeneti kombináció fellépésekor a zártsághoz az szükséges, hogy *1*-ből kiindulva *e* és *f*, valamint *3*-ból kiindulva *c* és *f* állapotok szerepeljenek együtt valamelyik osztályban. Az *1* és *3* osztályok ezt is teljesítik. Megállapíthatjuk tehát, hogy az *1* és *3* maximális kompatibilitási osztályokból álló halmaz zárt.

Vizsgáljuk meg ezek után, hogy elhagyhatók-e állapotok az egyes osztályokból. Ezzel ugyanis feltehetőleg növelhető a megmaradó közömbös bejegyzések száma. Csak *e* és *f* állapotok jöhetnek szóba, mert ezek szerepelnek *1*-ben is és *3*-ban is. A *3* osztályban *b* és *d* összevonásának feltétele *d* és *f*, valamint *c* és *f* összevonása, tehát *f* nem hagyható el sem *1*-ből, sem *3*-ból. Az *e* állapot viszont akár *1*-ből, akár *3*-ból elhagyható a zártság megsértése nélkül (ha *1*-ből hagyjuk el, akkor az *e* és *f* állapotra vonatkozó zártsági feltétel meg is szűnik). Az így előállítható két zárt halmaz elemeire vezessük be az alábbi jelöléseket:

(*acf*): *A*, (*bdef*): *B*;

(*acef*): *C*, (*bdf*): *D*.

Max. komp. osztályok \ Állapotok	*a*	*b*	*c*	*d*	*e*	*f*
1 (*acef*)	×		×		×	×
2 (*adef*)	×			×	×	×
3 (*bdef*)		×		×	×	×

3.107. ábra. A 3.103. ábrán bemutatott feladatra vonatkozó lefedési tábla

A 3.108. ábrán az *A* és *B*, a 3.109. ábrán pedig a *C* és *D* halmazoknak feleltettük meg az összevont állapottábla állapotait. Az adott példában mindkét megoldás biztosítja a legkevesebb sort tartalmazó összevont állapottáblát.

Nem mindig célszerű minden elhagyható állapotot elhagyni a maximális kompatibilitási osztályokból. Az állapotelhagyás csak akkor jelent csökkenést a hálózat állapotszámában, ha ezzel mellőzhetjük újabb kompatibilitási osztály felvételét a zártság biztosítására. Ha például nem hagyjuk el az *e* állapotot sem *1*-ből, sem *3*-ból, akkor az *1*:*E* és *3*:*F* jelölésekkel a 3.110. ábrán látható összevont állapottáblához jutunk, amelyben a közömbös bejegyzés kedvező lehet a megvalósítás szempontjából. Az állapotelhagyásnak tehát a következő hatásai lehetnek:

— Csökkenhet a zártságra vonatkozó feltételek száma, így kevesebb kompatibilitási osztály figyelembevétele elegendő lehet.

— Az egy osztályban összevont állapotok számának csökkenése miatt több közömbös bejegyzés maradhat az egyszerűsített állapottáblán.

— Az elhagyás miatt az elhagyott állapotot kevesebb kompatibilitási osztály tudja képviselni és emiatt az összevont állapottáblán nem mindegy, hogy melyik osztálynak megfelelő következő állapotba lépünk; így csökkenhet a közömbös bejegyzések száma.

Esetenként kell mérlegelni, hogy a fenti lehetséges hatások közül egy adott feladatban melyek érvényesülnek. Egyszerű példánkban a lényeges maximális kompatibilitási osztályok biztosították a legegyszerűbb lefedést és egyúttal a zártságot. Könnyen belátható azonban, hogy bonyolultabb esetben a legegyszerűbb zárt lefe-

y \ x	x^1	x^2	x^3
A	AZ^1	BZ^2	AZ^1
B	BZ^1	AZ^1	AZ^1

3.108. ábra. Az összevont állapottábla a 3.103. ábrán bemutatott feladatra vonatkozóan, ha az (*acf*): *A* és (*bdef*): *B* ekvivalenciaosztályoknak feleltetjük meg az összevont állapottábla állapotait

y \ x	x^1	x^2	x^3
C	CZ^1	CZ^2	CZ^1
D	DZ^1	CZ^1	CZ^1

3.109. ábra. Az összevont állapottábla a 3.103. ábrán bemutatott feladatra vonatkozóan, ha az (*acef*): *C* és a (*bdf*): *D* ekvivalenciaosztályoknak feleltetjük meg az összevont állapottábla állapotait

y \ x	x^1	x^2	x^3
E	EZ^1	$-Z^2$	EZ^1
F	FZ^1	EZ^1	EZ^1

3.110. ábra. Az összevont állapottábla a 3.103. ábrán bemutatott feladatra vonatkozóan, ha az (*acef*): *E* és a (*bdef*): *F* maximális kompatibilitási osztályoknak feleltetjük meg az összevont állapottábla állapotait

dést általában csak próbálgatással határozhatjuk meg az állapotelhagyások lehetséges következményeit is figyelembe véve. Egy adott megoldás értékeléséhez és a felesleges további próbálgatások elkerülése céljából hasznos lehet az alábbi bizonyítható összefüggés ismerete:

$$m \leqq p \leqq \min(n, k),$$

ahol p a legkevesebb állapotot tartalmazó összevont állapottábla állapotainak száma; n az előzetes állapottábla állapotainak száma; k a maximális kompatibilitási osztályok száma; m a legegyszerűbb lefedést biztosító maximális kompatibilitási osztályok száma.

Példánkban $n=6$, $k=3$, $m=2$ áll fenn, és a 3.108., 3.109., 3.110. ábrákon bemutatott megoldások mindegyikére $p=2$ érvényes. Így a

$$2 \leqq 2 \leqq \min(6, 3)$$

összefüggés alapján az állapotok száma szempontjából mindhárom megoldás optimálisnak tekinthető. A közömbös bejegyzés miatt azonban a 3.110. ábrán bemutatott megoldás a többihez képest kedvezőbbnek minősíthető

Összefoglalás

NTSH-k esetén a legkedvezőbb összevont állapottábla előállításának lépései:

1. Valamennyi kompatibilis és inkompatibilis (nem kompatibilis) állapotpár megkeresése lépcsős tábla felhasználásával.
2. A lépcsős tábla alapján a maximális kompatibilitási osztályok megkeresése.
3. A kompatibilitási osztályok legkedvezőbb zárt halmazainak megkeresése.
4. A legkedvezőbb zárt halmaz osztályainak egy-egy állapotot megfeleltetve az összevont állapottábla kitöltése.

Láttuk, hogy a fenti lépések közül a 3.-ban próbálgatásra és találékonyságra van szükség a legkedvezőbb összevont állapottábla meghatározásához.

Példaként oldjuk meg a 3.102. ábrán bemutatott feladatot. A 3.102*a)* ábrán szereplő nem teljesen határozott előzetes állapottáblára vonatkozó lépcsős tábla a 3.111. ábrán követhető. A megismert módszer már lehetővé teszi, hogy a közömbös bejegyzés értékét ne rögzítsük az állapot-összevonást megelőzően. Ha az $(ac):A$ és az $(ab):B$ megfeleltetéseket alkalmazzuk, akkor a 3.102*d)* ábrán bemutatott öszevont állapottáblához jutunk.

A bemutatott állapot-összevonási eljárások egyaránt alkalmazhatók szinkron és aszinkron előzetes állapottáblák esetén, hiszen az egyes lépésekben az állapottáblának

b	ac		(abc)
c	ab	$\times$ $\times$	(ac) (ab)
	a	b	

3.111. ábra. A kompatibilitási osztályok legkedvezőbb zárt halmazának meghatározása a 3.102*a)* ábrán bemutatott előzetes állapottáblából kiindulva

csak olyan formális tulajdonságaira hivatkoztunk, amelyek nem különböznek a szinkron és az aszinkron működési módban. Csupán azt érdemes megjegyezni, hogy az aszinkron sorrendi hálózatok előzetes állapottáblájára tett megkötések következtében az állapot-összevonás során mindig a nem teljesen határozott esetre vonatkozó eljárást kell alkalmaznunk. Ezek a megkötések ugyanis mindig okoznak közömbös bejegyzéseket az előzetes állapottábla rovataiban.

Az egyik ilyen megkötés szerint minden sorban csak egy stabil állapotot veszünk fel. Így minden állapot csak egyetlen bemeneti kombináció mellett lehet stabil, vagyis minden stabil állapothoz csak egyetlen bemeneti kombináció oszlopában vezethet instabil állapot. Ez a felépítés lehetővé teszi, hogy aszinkron esetben először csak azoknak az állapotoknak az összevonhatóságát vizsgáljuk egy lépcsős táblán, amelyek azonos bemeneti kombináció mellett stabilak. Az így kiadódó új közbenső állapottáblához újabb lépcsős táblát állítunk össze és a megismert módon fejezzük be az eljárást. Könnyen beláthatjuk, hogy a második lépcsős tábla rovataiba csak a feltétel nélküli összevonhatóság jelzése ($\surd$), vagy az összevonhatóság kizárt voltának jelzése ($\times$) kerülhet. Ennek oka az, hogy az első lépcsős tábla alapján az összes lehetséges feltételes összevonást elvégezzük a közbenső állapottábla létrehozásakor.

Az aszinkron előzetes állapottáblákra vázolt fenti kétlépéses módszer hátránya az, hogy két lépcsős táblát kell kitölteni. Előnye viszont az, hogy a lépcsős táblák kitöltése egyszerűbb, áttekinthetőbb és a második lépcsős tábla nem tartalmaz feltételt csupán $\surd$ és $\times$ bejegyzéseket.

A kétlépéses eljárást csak a teljesség kedvéért vázoltuk, mert nem szükséges, hogy az állapot-összevonás során megkülönböztessük a szinkron és az aszinkron működést. Az előzetes állapottáblák formálisan azonos módon is kezelhetők az állapot-összevonás szempontjából.

3.8.4. Az állapotkompatibilitás tulajdonságai

Foglalaljuk össze, hogy a kompatibilitás, mint egy NTSH állapotai közötti reláció, milyen tulajdonságokkal rendelkezik:

1. Minden állapot kompatibilis önmagával: $a \sim a$. Ez a tulajdonság a *reflexivitás*.
2. A kompatibilitás kölcsönös két állapot között: ha $a \sim b$, akkor $b \sim a$. A kompatibilitás tehát rendelkezik a *szimmetria*tulajdonsággal.
3. A kompatibilitás *nem tranzitív:* $a \sim b$ és $b \sim c$ fennállásából nem következik $a \sim c$.

Láthatjuk, hogy a kompatibilitás csak tranzitivitás szempontjából különbözik az ekvivalenciától. Ezt a különbséget a közömbös bejegyzések különböző rögzítési lehetőségei okozzák NTSH esetében.

A maximális kompatibilitási osztályok legfontosabb tulajdonságait a következőképpen foglalhatjuk össze:

1. A maximális kompatibilitási osztályok az előzetes állapottábla állapotaiból alkotott halmaz részhalmazai.

2. Az előzetes állapottábla minden egyes állapota megtalálható valamelyik maximális kompatibilitási osztályban.

3. A maximális kompatibilitási osztályoknak lehetnek közös állapotaik, azaz egy állapot több osztályban is szerepelhet. A maximális kompatibilitási osztályok tehát *nem diszjunkt* részhalmazai az összes állapot halmazának.

A maximális kompatibilitási osztályok tulajdonságai tehát csak a diszjunktság szempontjából különböznek a maximális ekvivalenciaosztályokétól.

A fenti tulajdonságok alapján megállapíthatjuk, hogy a maximális kompatibilitási osztályok az összes állapot halmazának *ún. fedőrendszerét* alkotják. A fedőrendszer — a partícióhoz hasonlóan — halmazelméleti fogalom; egy halmaz elemeinek olyan csoportosítását jelenti, hogy minden egyes elem szerepeljen legalább egy (de nem feltétlenül csak egy) csoportban, vagy más szóval *fedőrendszerblokkban*. A tranzitivitás hiánya tehát azt jelenti, hogy a maximális kompatibilitási osztályok nem partíciónak, hanem fedőrendszernek tekinthetők.

3.9. Állapotkódolási eljárások

Az előzetes állapottábla felvétele és egyszerűsítése után a sorrendi hálózatok tervezésének következő lépése az állapotkódolás. Az állapotkódolás során minden állapothoz egy szekunder változókombinációt rendelünk, és ha az állapottáblán minden állapot helyére a hozzárendelt kódot írjuk, a kódolt állapottáblát kapjuk. A kódokat többféleképpen is hozzárendelhetjük az állapotokhoz. Valamennyi olyan hozzárendelés, azaz kódolás, amelyben különböző állapotokhoz különböző kódok tartoznak, azonos logikai feladatot megoldó hálózatot eredményez. A választott kód azonban lényegesen befolyásolja a hálózat bonyolultságát. Ezt a következő példa szemlélteti.

$y \backslash x_1x_2$	00	01	11	10
a	b 0	c 0	b 0	c 0
b	f 0	d 0	f 0	d 0
c	d 0	d 0	d 0	d 0
d	e 0	e 0	e 0	e 0
e	a 0	a 0	a 0	a 0
f	g 0	e 0	g 0	e 0
g	a 1	a 0	a 1	a 0

3.112. ábra. Példaképpeni szinkron összevont állapottábla az állapotkódolás hatásának szemléltetéséhez

	y_1	y_2	y_3
a	0	0	0
b	0	0	1
c	0	1	0
d	0	1	1
e	1	0	0
f	1	0	1
g	1	1	0
Nem használt —	1	1	1

3.113. ábra. Kódolási változat a 3.112. ábrán szereplő állapottáblához

	$y_1y_2y_3 \backslash x_1x_2$	00	01	11	10
a	000	001 0	010 0	001 0	010 0
b	001	101 0	011 0	101 0	011 0
c	010	011 0	011 0	011 0	011 0
d	011	100 0	100 0	100 0	100 0
e	100	000 0	000 0	000 0	000 0
f	101	110 0	100 0	110 0	100 0
g	110	0001	0000	0001	0000

3.114. ábra. Kódolt állapottábla a 3.113. ábra kódolási változata alapján

Adott a 3.112. ábrán látható összevont szinkron állapottábla. Kódoljuk az állapotokat először az egymást követő bináris számoknak megfelelő szekunder változókombinációkkal (3.113. ábra). A kódolt állapottábla a 3.114. ábrán látható. Valósítsuk meg a hálózatot S—R flip-flopokkal. A megvalósítás ismert menetét nem részletezzük, csak a végeredményként keletkező logikai függvényeket adjuk meg:

$$R_1 = \bar{y}_3,$$

$$S_1 = y_2y_3+\bar{x}_1\bar{x}_2y_3+x_1x_2y_3,$$

$$R_2 = y_1y_2+y_2y_3,$$

$$S_2 = \bar{x}_1x_2\bar{y}_1\bar{y}_2+x_1\bar{x}_2\bar{y}_1\bar{y}_2+\bar{x}_1\bar{x}_2y_1y_3+x_1x_2y_1y_3,$$

$$R_3 = y_1+y_2y_3,$$

$$S_3 = \bar{y}_1y_2\bar{y}_3+\bar{x}_1\bar{x}_2\bar{y}_1\bar{y}_3+x_1x_2\bar{y}_1\bar{y}_3,$$

$$Z = \bar{x}_1\bar{x}_2y_1y_2+x_1x_2y_1y_2.$$

A három S—R flip-flopot vezérlő kombinációs hálózatrész és a kimenet előállítása ilyen kódolás mellett kétszintű ÉS—VAGY hálózattal megvalósítva összesen 61 kapubemenetet igényel. Ha észrevesszük, hogy az y_2y_3 prímimplikáns három függvényben szerepel és elegendő egyszer megvalósítani, akkor 57 kapubemenet adódik.

	y_1	y_2	y_3
a	1	0	1
b	0	0	0
c	0	0	1
d	0	1	0
e	1	1	0
f	0	1	1
g	1	1	1
Nem használt	1	0	0

3.115. ábra. Kódolási változat a 3.112. ábrán szereplő állapottáblához

	$y_1y_2y_3$ \ x_1x_2	00	01	11	10
a	101	000 0	001 0	000 0	001 0
b	000	011 0	010 0	011 0	010 0
c	001	010 0	010 0	010 0	010 0
d	010	110 0	110 0	110 0	110 0
e	110	101 0	101 0	101 0	101 0
f	011	111 0	110 0	111 0	110 0
g	111	101 1	101 0	101 1	101 0

3.116. ábra. Kódolt állapottábla a 3.115. ábra kódolási változata alapján

Válasszunk most más kódolást, pl. 3.115. ábra szerintit. A kódolt állapottábla a 3.116. ábrán látható. Ez alapján az alábbi vezérlési függvények adódnak:

$$R_1 = \bar{y}_2,$$

$$S_1 = y_2,$$

$$R_2 = y_1,$$

$$S_2 = \bar{y}_1,$$

$$R_3 = \bar{x}_1\bar{x}_2\bar{y}_2y_3+x_1x_2\bar{y}_2y_3+\bar{x}_1x_2\bar{y}_1+x_1\bar{x}_2\bar{y}_1+\bar{y}_1\bar{y}_2y_3,$$

$$S_3 = y_1y_3+x_1x_2y_2y_3+x_1x_2y_2y_3,$$

$$Z = \bar{x}_1\bar{x}_2y_1y_2y_3+x_1x_2y_2y_3.$$

Láthatjuk, hogy ilyen kódolással 43 kapubemenet szükséges.

Nyilvánvaló tehát, hogy célszerű olyan állapotkódolást választani, amely a lehető legegyszerűbb hálózatot adja. Szinkron sorrendi hálózatok esetében az állapotkódolás megválasztásának ez az egyetlen szempontja. Aszinkron hálózatok esetében azonban láttuk, hogy a kódválasztáskor peremfeltétel a kritikus versenyhelyzet elkerülése. Először a szinkron sorrendi hálózatok állapotkódolásának megválasztására mutatunk be néhány módszert.

3.9.1. Szinkron sorrendi hálózatok állapotkódolása

A hálózat egyszerűsége szempontjából legkedvezőbb kódolást csak úgy határozhatnánk meg, hogy az adott hálózat összes lehetséges kódolásával elvégeznénk a megvalósítást és ezek közül kiválasztanánk a legegyszerűbbet. Vizsgáljuk meg, hogy ehhez hány megoldást kellene összehasonlítanunk. Legyen az állapottábla sorainak száma r, a felhasználandó szekunder változók száma p. A lehetséges kódolások számát ebből a következő gondolatmenettel kapjuk. A képezhető szekunder változókombinációk száma 2^p. Annyi kódolás lehetséges, ahányféleképpen 2^p kombinációból kiválaszthatunk r-et, figyelembe véve, hogy a kiválasztott kombinációkat tetszőleges sorrendben rendelhetjük az állapotokhoz. A kiválasztási lehetőségek száma $\binom{2^p}{r}$; és minden kiválasztás $r!$ sorrendben szerepelhet. Így a lehetséges kódolások száma (N):

$$N = \frac{2^p!}{(2^p-r)!\cdot r!}\cdot r! = \frac{2^p!}{(2^p-r)!}.$$

Belátható, hogy a hálózat bonyolultsága szempontjából nem jelentenek különböző kódolást azok, amelyek a szekunder változók átcsoportosításával vagy invertálásával egymásból előállíthatók.

Például: a, b, c és d állapotokat kell kódolni két szekunder változóval. Egy lehetséges kódolás:

	y_1	y_2
a	0	0
b	0	1
c	1	0
d	1	1

Ha y_1 és y_2 értékeit rendre felcseréljük, akkor az alábbi kódoláshoz jutunk:

	y_1	y_2
a	0	0
b	1	0
c	0	1
d	1	1

Érezhetően csak az elnevezések változtak, ez pedig nem változtathatja meg a vezérlési függvények struktúráját, bármilyen állapottáblát tételezünk is fel. Ugyancsak nem változhat a struktúra pl. az y_2 értékének invertálásával adódó

	y_1	y_2
a	0	1
b	0	0
c	1	1
d	1	0

kódolás esetén sem.

r	p	N_k
2	1	1
4	2	3
5	3	140
8	3	840
9	4	10 810 800
16	4	2 270 268 000

3.117. ábra. A hálózat bonyolultságára gyakorolt hatás szempontjából megkülönböztetendő kódolások számának növekedése

Vizsgáljuk meg, hogy egy adott kódolásból hány újabb kódolás állítható elő a szekunder változók értékének felcserélésével, ill. invertálásával. Invertálással annyi, ahány kombináció képezhető a szekunder változókból, hiszen bármelyik szerepelhet akár az eredeti, akár a negált értékével, tehát 2^p számú. Ezek mindegyikénél tetszőleges sorrendben szerepelhetnek a változók, azaz $p!$-féleképpen. Az adott kódolásból invertálással és felcseréléssel eőállítható tehát $2^p \cdot p!$ számú kódolás. Így a lehetséges kódolások számából az egyszerűsítő hatás szempontjából megkülönböztetendő kódolások számát (N_k) a következőképpen kapjuk:

$$N_k = \frac{N}{2^p \cdot p!} = \frac{2^p!}{(2^p - r)! \cdot p! \cdot 2^p} = \frac{(2^p - 1)!}{(2^p - r)!\, p!}.$$

N_k értékét néhány jellegzetes esetre a 3.117. ábrán látható táblázat tartalmazza. Az értékekből látható, hogy kettőnél több szekunder változó esetén reménytelenül hosszadalmas az összes lehetőség kipróbálása.

A továbbiakban ehelyett néhány olyan módszert mutatunk be, amelyek segítségével a kódválasztáskor valamelyest törekedni tudunk a kialakuló hálózat egyszerűségére anélkül, hogy az egyes kódolások hatását próbálgatással ellenőriznénk. Természetesen korántsem állíthatjuk, hogy ezekkel a módszerekkel mindig eljutunk az optimális kódoláshoz, azaz a lehető legegyszerűbb hálózathoz, hiszen ezt csak az összes lehetőség próbálgatásos vizsgálatával dönthetnénk el.

Az állapotkódoláshoz legalább annyi szekunder változót kell használnunk, amennyi az állapotok megkülönböztetéséhez feltétlenül szükséges. A szekunder változók szükséges legkisebb száma az alábbi egyenlőtlenség alapján határozható meg:

$$2^p \geqq r;$$

ahol r az állapotok száma, p a szekunder változók száma. Azaz a képezhető szekunder változókombinációk száma legalább akkora legyen, mint az állapotok száma. Ebből:

$$p = \lceil \log_2 r \rceil,$$

ahol $\lceil\ \rceil$ szimbólummal jelöltük a következő egész számra való felkerekítést.

Ha több szekunder változót használunk a kódoláshoz, mint amennyi az állapotok megkülönböztetéséhez feltétlenül szükséges, akkor természetesen több logikai függvényt kell megvalósítanunk, ami bonyolultabbá teheti a kialakuló hálózatot. Ugyanakkor a többlet szekunder változók mindig azt eredményezik, hogy az ilyenkor mindig keletkező kihasználatlan kódok miatt növekszik az egyes függvények közömbös értékeinek száma, ami viszont egyszerűsítheti a kiadódó függvényeket. Később látni fogjuk, hogy a többlet szekunder változók alkalmazásával néha összhatásban egyszerűbb hálózatot kapunk, mint az állapotok megkülönböztetéséhez feltétlenül szükséges számú szekunder változó esetén.

Kíséreljük meg a kódválasztás szempontjából használható módon megfogalmazni a kialakuló hálózat egyszerűségének követelményét. A kódolás után kiadódó hálózatot jellemezzük az

$$f_y(x, y) \Rightarrow Y$$

leképezést megvalósító logikai függvényekkel:

$$Y_1 = f_1(x_1, \ldots, x_n, y_1, \ldots, y_k),$$
$$\vdots$$
$$Y_i = f_i(x_1, \ldots, x_n, y_1, \ldots, y_k),$$
$$\vdots$$
$$Y_k = f_k(x_1, \ldots, x_n, y_1, \ldots, y_k).$$

A módszer alapgondolata az, hogy ha a fenti logikai függvények egy adott állapotkódolás után egyszerű algebrai alakúak, akkor bármilyen típusú flip-flopot is választunk a megvalósításhoz, az azokat vezérlő kombinációs hálózat feltételezhetően szintén nem adódik túl bonyolultnak.

Az $f_z(x, y) \Rightarrow Z$, ill. $f'_z(y \Rightarrow) Z$ leképezéseket megvalósító kimeneti függvények egyszerűségét természetesen szintén befolyásolják az állapotkódok. Így előfordulhat, hogy az f_y és az f_z, ill. f'_z leképezések egyszerű megvalósítása egymásnak ellentmondó feltételeket szab az állapotkódokra vonatkozóan, ha a 3.67. ábra szerinti felépítés több kimenetű kombinációs hálózatának egészét kíséreljük meg minimalizálni. A továbbiakban ezért feltételezzük, hogy a flip-flopokat vezérlő kombinációs hálózatrész döntő részét képezi a teljes több kimenetű kombinációs hálózatnak. Ez a feltételezés mindenképpen helytálló például olyankor, amikor a kimenetek száma jóval kisebb, mint a szükséges szekunder változóké. Ha azonban egy adott sorrendi hálózatban feltételezésünk fordítottja teljesül, akkor természetesen a kimeneti függvények egyszerűségét kell elsődleges szempontnak tekintenünk az állapotkódok megválasztásakor. A bemutatandó módszerek alapgondolata ilyen esetekben is hasznosítható.

Fentiek értelmében a továbbiakban az $Y_1 \ldots Y_i \ldots Y_k$ függvényeknek kódválasztással történő egyszerűsítésére mutatunk be módszereket. Korábban láttuk, hogy a fenti függvényekkel D flip-flop alkalmazása esetén egyúttal a vezérlő kombinációs hálózatot is megadtuk. Vizsgáljuk meg, hogy a kódválasztással miként befolyásolhatjuk a fenti függvények egyszerűségét.

3.9.1.1. *Szomszédos kódolás*

Nyilvánvaló, hogy az egyes Y_i függvények egyszerűsítése szempontjából kedvező, ha mintermjeik páronkénti szomszédosságára törekszünk a kódválasztáskor. Ennek egyik módja az, hogy páronként szomszédos kódokat rendelünk mindazon állapotokhoz, amelyeknek azonos következő állapotuk van valamilyen bemeneti kombináció mellett. Ilyenkor ugyanis az azonos következő állapotnak megfelelő $Y_1 \dots Y_i \dots Y_k$ kombináció 1-es értékeit mint függvényértékeket páronként szomszédos $y_1 \dots y_i \dots y_k$ kombinációk állítják elő az adott bemeneti kombinációk mellett. Így tehát az egyes Y_i függvényeknek az $y_1 \dots y_i \dots y_k$ változók értékei szerint szomszédos mintermpárjai keletkeznek, ami egyszerűsítési lehetőségeiket nyilvánvalóan növeli.

Az Y_i függvények egyszerűsítése szempontjából az is kedvező, ha páronként szomszédos kódokat rendelünk mindazon állapotokhoz, amelyek ugyanannak az állapotnak a következő állapotai valamilyen bemeneti kombinációi mellett. Ekkor az azonos $y_1 \dots y_i \dots y_k$ kombinációkhoz tartozó Y_i értékek legtöbbje azonos, hiszen a következő állapotok páronkénti szomszédos kódja következtében az illető $Y_1 \dots Y_i \dots Y_k$ kombinációk páronként csak egyetlen helyértéken különböznek egymástól. Így az $Y_1 \dots Y_i \dots Y_k$ függvényeknek nagy valószínűséggel keletkeznek közös mintermjeik, ami a több kimenetű vezérlő kombinációs hálózat egyszerűsítése szempontjából kedvezően hat, mert növeli a közös prímimplikánsok képzésének lehetőségét. Ezen túlmenően az ilyen kódválasztás nagy valószínűséggel azt eredményezi, hogy az egyes Y_i függvényértékek függetlenekké válnak, egy vagy több bemeneti változótól, ami a függvény algebrai alakját szintén egyszerűsíti. Ennek oka az, hogy az Y_i függvényértékek többsége a szomszédos kódolás következtében azonos $y_1 \dots y_i \dots y_k$ kombináció mellett azonos, vagyis nem függ a bemeneti kombinációtól.

A fenti meggondolások alapján összefoglalva, a kódválasztáskor az alábbi két követelményt célszerű szem előtt tartanunk:

a) Legyen páronként szomszédos mindazon állapotok kódja, amelyeknek azonos következő állapotuk van valamilyen bemeneti kombináció mellett.

b) Legyen páronként szomszédos mindazon állapotok kódja, amelyek ugyanannak az állapotnak a következő állapotai valamilyen bemeneti kombináció mellett.

Ha egy adott állapottábla esetén a két követelmény egyidejű teljesítése ellentmondáshoz vezetne, akkor ajánlatos az *a)* pontbelit előnyben részesíteni, mert ilyenkor ettől általában nagyobb egyszerűsítő hatást várhatunk. Sőt, az is előfordulhat, hogy vagy az *a)* pontbeli vagy a *b)* pontbeli követelmény önmagában olyan páronkénti szomszédossági viszonyokat ír elő, hogy azok mindegyike nem teljesíthető egyidejűleg. Ilyen esetben nem tehetünk mást, mint a szomszédossági előírások számát a lehető legkisebb mértékben csökkentve ellentmondásmentesen teljesíthető szomszédossági követelményeket hozunk létre.

Példaképpen kódoljuk a 3.112. ábrán szereplő állapottábla állapotait a fentiekben leírt módszerrel. Az állapottáblából kiolvashatjuk, hogy melyek azok az állapotok, amelyeknek azonos következő állapotuk van:

Következő állapot	Kiindulási állapot
a	*e, g*
b	*a*
c	*a*
d	*b, c*
e	*d, f*
f	*b*
g	*f*

Az *a)* pontbeli követelmény szerint tehát

e állapotnak	*g* állapottal
b állapotnak	*c* állapottal
d állapotnak	*f* állapottal

kell kódban szomszédosnak lennie.

Hasonló módon kiolvasható az állapottáblából az is, hogy melyek azok az állapotok, amelyek ugyanannak az állapotnak a következő állapotai:

Kiindulási állapot	Következő állapot
a	*b, c*
b	*d, f*
c	*d*
d	*e*
e	*a*
f	*e, g*
g	*a*

A *b)* pontbeli követelmény szerint tehát

b állapotnak	*c* állapottal
d állapotnak	*f* állapottal
e állapotnak	*g* állapottal

kell kódban szomszédosnak lennie.

Láthatjuk, hogy a *b)* pontbeli követelmény teljesítése az adott esetben nincs ellentmondásban az *a)* pontbelivel, hiszen ugyanazokat a szomszédossági viszonyokat írja elő.

A kapott szomszédossági előírásokat kielégítő kódok megválasztásához előnyösen alkalmazhatjuk a Karnaugh-táblát a szomszédossági viszonyok áttekintésére. A tábla peremezését az adott esetben feltételenül szükséges 3 db szekunder változó értékkombinációja szerint végezzük és így az egyes cellákba írhatjuk az adott szekunder kombináció által képviselt állapot szimbólumát. A 3.118. ábrán ennek egy lehetséges megoldása látható. Ennek alapján a választott kódolást a 3.119. ábrán foglaltuk össze.

		y_1		
	a	b	c	d
y_1	–	g	e	f
		y_2		

3.118. ábra. Az állapotkódok szomszédossági előírásainak szemléltetése Karnaugh-táblán

	y_1	y_2	y_3
a	0	0	0
b	0	1	0
c	1	1	0
d	1	0	0
e	1	1	1
f	1	0	1
g	0	1	1
Nem használt –	0	0	1

3.119. ábra. A szomszédos kódolás módszerével meghatározott kódolási változat a 3.112. ábrán szereplő állapottáblához

	$y_1y_2y_3$ \ x_1x_2	00	01	11	10
a	000	010 0	110 0	010 0	110 0
b	010	101 0	100 0	101 0	100 0
c	110	100 0	100 0	100 0	100 0
d	100	111 0	111 0	111 0	111 0
e	111	0000	0000	0000	0000
f	101	011 0	111 0	011 0	111 0
g	011	0001	000 0	0001	0000

3.120. ábra. Kódolt állapottábla a 3.119. ábra kódolási változata alapján

A 3.120. ábrán szereplő kódolt állapottábla alapján az ismert megvalósítási lépések után S—R flip-flop alkalmazása esetén az alábbi logikai függvényekhez jutunk:

$$R_1 = y_2y_2+\bar{x}_1\bar{x}_2y_3+x_1x_2y_3,$$

$$S_1 = y_2\bar{y}_3+\bar{x}_1x_2\bar{y}_3+x_1\bar{x}_2\bar{y}_3,$$

$$R_2 = y_2,$$

$$S_2 = \bar{y}_2,$$

$$R_3 = y_2y_3,$$

$$S_3 = y_1\bar{y}_2+\bar{x}_1\bar{x}_2\bar{y}_1y_2\bar{y}_3+x_1x_2\bar{y}_1y_2\bar{y}_3,$$

$$Z = \bar{x}_1\bar{x}_2\bar{y}_1y_3+x_1x_2\bar{y}_1y_3.$$

Ha az y_2y_3 prímimplikánst csak egyszer valósítjuk meg, akkor a szükséges kapubemetek száma: 47. A szomszédos kódolás alapján kapott hálózat tehát egyszerűbb, mint a 3.113. ábra kódolási változatával kialakult hálózat. Azt azonban természetesen nem állíthatjuk, hogy ez az optimális kódolás.

3.9.1.2. Önfüggő szekunder változócsoportok szerinti kódolás

Az eljárás alapgondolata az, hogy az $Y_i=f_i(x_1, \ldots, x_n, y_1, \ldots, y_k)$ függvények egyszerűsége szempontjából mindenképpen kedvező, ha azok minél kevesebb változótól függenek. Az állapotkódok alkalmas megválasztásával azt tudjuk befolyásolni, hogy az egyes Y_i függvényértékek mely y változóktól legyenek függetlenek. Nyilvánvalóan előnyös, ha az egyes Y_i függvények algebrai alakjaiban a lehető legkevesebb y változó szerepel. Természetesen nem minden esetben sikerül az y változók számát csökkenteni, de célszerű megvizsgálni ezt a lehetőséget az állapotkódolás megválasztásakor. Ennek egyik módja az lehet, hogy alkalmas kódválasztással megkíséreljük előidézni, hogy az Y_i függő változóknak legyenek olyan csoportjai, amelyekben levő Y változók csak a csoportbeli Y változóknak megfelelő y változóktól (és természetesen a bemeneti változóktól) függenek. A szekunder változók ilyen csoportjait *önfüggő csoportoknak* nevezik, amelyek létezésének vizsgálatára és megkeresésük módjára mutatunk be a továbbiakban egy eljárást.

Induljunk ki a 3.112. ábrán szereplő állapottáblához a 3.113. ábrán adott kódolási változatból, és vizsgáljuk meg, hogy az y_1 és y_2 szekunder változók önfüggő csoportot alkotnak-e. Ha y_1 és y_2 önfüggő csoportot alkot, akkor Y_1 és Y_2 értéket meg lehet adni y_1 és y_2, valamint a bemeneti kombináció ismeretében, vagyis y_3 ismerete nélkül. Az y_1 és y_2 változók értékkombinációi egy-egy állapotcsoportot jelölnek ki, a csoporton belüli állapotokat y_3 értéke különbözteti meg. A 3.113. ábra szerinti kódolás esetén ezek az állapotcsoportok az alábbiak:

y_1	y_2	
0	0	ab
0	1	cd
1	0	ef
1	1	g

Az y_3 szekunder változó értékének ismerete nélkül tehát csak az dönthető el, hogy a hálózat a fenti állapotcsoportok közül melyikben van és nem tudjuk a csoporton belüli állapotot meghatározni. Ha az y_1, y_2 önfüggő csoport létezik, akkor az Y_1 és Y_2 értékek által kijelölt következő állapotok csoportjának meghatározásához nincs szükségünk y_3 értékének ismeretére.

Az önfüggés feltétele az állapotátmenetekre nézve a következő: ismerve, hogy a hálózat melyik állapotcsoportban van (ismerve y_1 és y_2 értékét), de nem ismerve, hogy a csoport melyik állapotában (y_3 ismerete nélkül), a bemeneti kombináció ismeretében meg tudjuk határozni, hogy melyik következő állapotcsoportba kerül (meg tudjuk határozni Y_1 és Y_2 értékét). Más szóval, az y_1 és y_2 által meghatározott bármelyik állapotcsoport állapotaiból kiindulva bármelyik bemeneti kombináció hatására létrejövő következő állapotok is egyetlen állapotcsoporton belül szerepeljenek.

Teljesül-e ez például az $y_1y_2=00$ által meghatározott (ab) csoportra? A következő állapotok: $x_1x_2=00$ és 11 hatására (bf); 01 és 10 hatására pedig (cd). Mivel

b és f nem szerepel egy csoporton belül, a feltétel nem teljesül. (Különböző Y_1 és Y_2 értékek vannak előírva attól függően, hogy a csoporton belül az a vagy b állapotban van a hálózat, tehát y_3-tól is függ Y_1 és Y_2.) A választott kódolás mellett tehát y_1 és y_2 nem alkot önfüggő csoportot. Ezt természetesen az is jelzi, hogy a vizsgált kódolás mellett kapott S_1, R_1, S_2 és R_2 függvények y_1-en és y_2-n kívül y_3-tól is függenek.

A példa alapján megfogalmazhatjuk, hogy miként válasszunk olyan kódolást, amely a szekunder változóknak önfüggő csoportját eredményezi. Keressünk olyan állapotcsoportokat, amelyekre teljesül az állapotátmenetekre vonatkozó fenti feltétel, és az így kialakuló csoportokat kódoljuk a szükséges számú szekunder változóval. Ez a változócsoport önfüggő lesz. Ezután a csoportokon belüli állapotok megkülönböztetésére használjunk további szekunder változókat. Ezek már nem alkotnak önfüggő csoportot.

Figyeljük meg, hogy az állapotcsoportok keresése során hasonló feltételt kell kielégítenünk, mint az állapot-összevonás során: két állapot egy csoportba tartozik, ha bármelyik bemeneti kombináció hatására kialakuló következő állapotaik is egy csoportba tartoznak. A különbség csupán az, hogy a kimeneteknek jelenlegi vizsgálatunknál semmiféle jelentőségük nincs. A feltétel formailag azonos a kompatibilitási osztályokra megfogalmazott zártsági feltétellel, diszjunkt osztályok esetén. Az állapotcsoportok ezúttal ugyanis nyilvánvalóan nem tartalmazhatnak közös állapotokat. A kialakuló állapotcsoportokat egy partíció blokkjainak tekinthetjük.

Az állapotoknak olyan partícióját, amely kielégíti a fenti feltételt, vagyis amely alapján önfüggő szekunder változócsoportot eredményező kódolás választható, *helyettesítési tulajdonságú* partíciónak nevezik. Definíciószerűen tehát:

Egy sorrendi hálózat állapotainak egy partícióját helyettesítési tulajdonságú partíciónak nevezzük, ha egy tetszőleges blokkjának bármely két állapotából kiindulva bármely bemeneti kombináció hatására kialakuló két következő állapot is közös blokkban szerepel. Helyettesítési tulajdonságú például a 3.112. ábrán szereplő állapottáblán a $\Pi = \{(adf)(bceg)\}$ partíció.

Fentiek alapján általánosan is megfogalmazhatjuk: *Ha egy sorrendi hálózat állapottábláján helyettesítési tulajdonságú partíció mutatható ki, akkor ennek alapján biztosan választható olyan kódolás, amely a szekunder változók önfüggő csoportját eredményezi.*

Belátható, hogy nem minden állapottáblán található helyettesítési tulajdonságú (a továbbiakban: h. t.) partíció, vagyis nem mindig alkalmazható a kódválasztásra a bemutatott módszer. A h. t. partíciók keresését a következő módon végezhetjük.

Az eljárás lényege az, hogy kiindulunk az állapottábla két tetszőleges állapotából, amelyekről feltételezzük, hogy egy h. t. partíciónak ugyanabban a blokkjában vannak. Ezután megvizsgáljuk, hogy az így feltételezett blokk létezése a helyettesítési tulajdonság alapján milyen más blokkok létezését írja elő. Az ellenőrzést az összes keletkezett blokk bármely két-két állapot[illegible]el kell végeznünk a bemeneti kombinációnkénti következő állapotok egy blok[illegible]artozásának vizsgálatával. Ha a keletkezett blokkok között olyanok akadnak, a[illegible]y[illegible] azonos állapotokat is tartalmaznak (tehát

nem diszjunktak), akkor ezeket a blokkokat a további vizsgálat szempontjából egyesítenünk kell, hiszen a h. t. partíció blokkjainak diszjunktaknak kell lenniük.

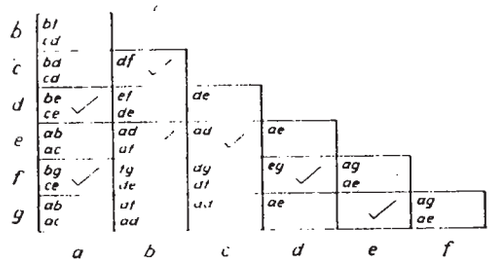
A kiindulási két állapotról az eljárás végén úgy derül ki, hogy nem lehetnek egy h. t. partíciónak ugyanabban a blokkjában, hogy a blokkok sorozatos egyesítése folytán az összes állapotot tartalmazó blokkhoz jutunk. Ez triviális h. t. partíciónak tekinthető és természetesen nem használható önfüggő szekunder változócsoportot eredményező kódolás céljából, ami úgy is fogalmazható, hogy csak olyan önfüggőcsoportot képezhetünk ez alapján, amely az összes szekunder változót tartalmazza.

A keresés folyamatát egy példán szemléltetjük. Keressünk h. t. partíciót a 3.112. ábrán szereplő állapottáblán. Az állapotok páronkénti vizsgálatához képezzük a 3.121. ábrán látható lépcsős táblát. Az egyes cellákba írt állapotpárok ezúttal azt jelentik, hogy az illető cellának megfelelő állapotpár egy blokkba tartozásához bemeneti kombinációnként mely következő állapotoknak kell egy blokkba tartozniuk. Például az a és b állapot által kijelölt cellába azért írtunk bf-et és cd-t, mert az állapottáblán a és b állapotokból kiindulva bemeneti kombinációnként b és f, valamint c és d következő állapotok adódnak, amelyeknek a helyettesítési tulajdonság értelmében a és b egy blokkba tartozása esetén szintén egy blokkba kell tartozniuk. Az így kitöltött lépcsős tábla alapján a keresést úgy végezhetjük, hogy kiindulunk két állapotból és feltételezzük, hogy ezek egy blokkban vannak. Példánkban legyen ez a két kiválasztott állapot először a és b. Az ezeknek megfelelő cellába a lépcsős táblán tegyünk $\surd$ jelet. A cellában szereplő állapotpároknak megfelelő cellákba (bf és cd) szintén tegyünk jelet. A bf és cd állapotpárok által kijelölt cellákban szereplő állapotpárok (gf és de) újabb cellákat jelölnek ki, amelyekbe szintén $\surd$ jelet teszünk. Az eljárást így folytatjuk mindaddig, amíg újabb $\surd$ jelet eredményező állapotpár nem adódik. Figyeljük meg, hogy pl. a bd állapotpár által kijelölt cellában szereplő fe és de állapotpárok miatt az fd állapotpár által kijelölt cellába is $\surd$ jelet kell tennünk. Ennek oka, hogy fe és de nem diszjunktak, egyesítésük folytán a def blokk adódik és ennek összes állapotpárjára — így fd-re is — fel kell tételezni a helyettesítési tulajdonságot. A ge állapotpár által kijelölt cellába azért került eleve feltétel nélkül $\surd$ jel, mert g és e egy blokkba tartozásának az állapottábla alapján csak az a feltétele, hogy az a állapot önmagával egy blokkba tartozzon, ami mindig teljesül.

A bizonyítást mellőzve a továbbiakban feltételezzük, hogy két állapot egy blokkba tartozása eddigi értelmezésünk szerint formálisan ekvivalenciarelációként

	a	b	c	d	e	f
b	bf cd ✓					
c	bd cd ✓	fd ✓				
d	be ce ✓	fe de ✓	de ✓			
e	ab ac ✓	ad af ✓	ad ✓	ae ✓		
f	bg ce ✓	gf de ✓	dg df ✓	eg ✓	ag ae ✓	
g	ab ac ✓	af ad ✓	ad	ae	✓	ag ae ✓

3.121. ábra. A 3.112. ábrán szereplő állapottábla h. t. partícióinak kereséséhez képzett lépcsős tábla az a és b állapotok egy blokkba tartozásának kiindulási feltételezésével



3.122. ábra. A 3.112. ábrán szereplő állapottábla h. t. partícióinak kereséséhez képzett lépcsős tábla az *a* és *d* állapotok egy blokkba tartozásának kiindulási feltételezésével

kezelhető. Így a lépcsős tábla alapján az állapot-összevonási eljárás során megismert formális módszerrel képezhető ekvivalenciaosztályok ezúttal a keresett h. t. partíciók blokkjait jelölik. A lépcsős tábla √ jel nélkül maradt celláiba képzelve az állapot-összevonási eljárás során alkalmazott × jeleket, formálisan alkalmazhatjuk az ott megismert eljárást a 3.121. ábra alapján:

$(abcdefg)$,

$(abdefg)(abcdef)$,

$(abefg)(abdef)(abcdef)$.

Mivel ekvivalenciarelációt tételeztünk fel, csak diszjunkt blokkokat értelmezhetünk. Ezért a kapott nem diszjunkt blokkokat egyesítenünk kell:

$(abcdefg)$,

vagyis triviális h. t. partícióhoz jutunk.

Megállapíthatjuk tehát, hogy kiindulási feltételünk nem teljesülhet, vagyis *a* és *b* nem lehet ugyanabban a h. t. partícióblokkban.

Tételezzük fel ezek után kiindulásképpen, hogy az *a* és *d* állapotok vannak egy blokkban. Ezzel a feltételezéssel a 3.122. ábrán látható módon alakulnak a lépcsős táblába írt √ jelek. Ebből az alábbi módon adódnak a blokkok:

$(abcedfg)$,

$(bcdefg)(adf)$,

$(cdefg)(bceg)(adf)$,

$(defg)(bce)(beg)(bce)(adf)$,

$(efg)(adf)(beg)(bce)(adf)$,

$(fg)(eg)(beg)(bce)(adf)$,

$(f)(g)(beg)(bce)(adf)$.

Elvégezve a szükséges blokkegyesítést az alábbi h. t. partíciót kapjuk.

$\Pi_1 = \{(adf)(bceg)\}$

Ezúttal tehát kiindulási feltételezésünk helyesnek bizonyult: *a* és *d* állapotok valóban egy blokkban vannak a kapott h. t. partícióban.

Mivel a lépcsős táblán az *eg* állapotpárnak megfelelő cellában feltétel nélküli √ jel szerepel, ezért közvetlenül kiolvashatjuk, hogy *eg* biztosan blokkja egy h. t. partíciónak. Így az alábbi h. t. partíciót írhatjuk fel:

$\Pi_2 = \{(a)(b)(c)(d)(eg)(f)\}$.

Ha a keresés kiindulásaként a *df* állapotpár egy blokkba tartozását tételezzük fel, akkor a fentiekben ismertetett eljárással végeredményként a

$\Pi_3 = \{(a)(b)(c)(df)(eg)\}$

h. t. partíciót kapjuk.

A *bc* állapotpár egy blokkba tartozásának kiindulási feltételezésével pedig a

$\Pi_4 = \{(a)(bc)(df)(eg)\}$

h. t. partíció adódik.

Az állapottáblához tartozó összes h. t. partíció megtalálásához elvileg az összes állapotpárból kiindulva végre kell hajtani a bemutatott keresést. Gyakorlatilag azonban erre általában nincs szükség, mert a partíciókra vonatkozó algebrai műveletek segítségével két h. t. partícióból továbbiak képezhetők. Ezt nem részletezzük, mert a gyakorlatban az ilyen módon kezelhető méretű állapottáblák esetén két-három h. t. partíció ismerete már elegendő a kódválasztással kapcsolatos további mérlegelésekhez.

A következő feladatunk az, hogy a kapott h. t. partíciók közül kiválasszuk azt, amelynek alapján meghatározzuk az állapotkódokat. Egyik szempontunk lehet az, hogy a lehető legkevesebb szekunder változót használjuk. Tudjuk, hogy a szekunder változók egyik csoportjával a h. t. partíció blokkjait, másik csoportjával pedig a blokkokon belüli állapotokat kell megkülönböztetnünk. Így ha blokkok számát *B*-vel, az egy blokkban előforduló állapotok legnagyobb számát *A*-val jelöljük, akkor a kódoláshoz szükséges szekunder változók számát (p) az alábbi összefüggés alapján határozhatjuk meg:

$$p = \lceil \log_2 B \rceil + \lceil \log_2 A \rceil,$$

ahol a $\lceil\ \rceil$ jelölés az értéknek a legközelebbi egész számra történő felkerekítésére utal.

A fentiekben meghatározott négy h. t. partíció esetén a kódoláshoz feltétlenül szükséges szekunder változók száma rendre a következőképpen alakul:

	A	B	p
Π_1:	2	4	3
Π_2:	6	2	4
Π_3:	5	2	4
Π_4:	4	2	3

Ha tehát a legkevesebb szekunder változó alkalmazására törekszünk, akkor Π_1 vagy Π_4 alapján célszerű a kódolást végeznünk. Π_1-hez az alábbi kódolást képezhetjük:

	y_1	y_2	y_3
a	0	0	0
b	1	0	0
c	1	0	1
d	0	1	0
e	1	1	0
f	0	1	1
g	1	1	1

ahol y_1 értékével különböztettük meg Π_1 blokkjait, y_2 és y_3 értékkombinációival pedig a blokkokon belüli állapotokat.

Figyeljük meg, hogy az így képzett kódolással a Π_4 partíció szerinti állapotcsoportosítást is elvégeztük, hiszen y_1 és y_2 értékkombinációi az alábbi megkülönböztetéseket okozzák:

	y_1	y_2
a	0	0
b, c	1	0
d, f	0	1
e, g	1	1

vagyis Π_4 blokkjait különböztetik meg.

Azt tapasztaltuk tehát, hogy a Π_1 és Π_4 által előírt csoportosítások megvalósíthatók ugyanazzal a kódolással. Ezt abból is észrevehettük volna, hogy Π_4-ből Π_1 előállítható blokkegyesítések révén. A partíciókra vonatkozó algebrai műveletek és tulajdonságok alapján ezt úgy is fogalmazhatjuk, hogy Π_1 tartalmazza Π_4-et: $\Pi_4 \leqq \Pi_1$. Ilyen esetben Π_1 alapján mindig választható olyan kódolás, amely egyúttal a Π_4 szerinti csoportosítást is megvalósítja. Az ilyen kódolás azért előnyös, mert az önfüggő szekunder változócsoportok nemcsak Π_1 szerint, hanem Π_4 szerint is keletkeznek. Így tehát y_1 önmagában önfüggő, valamint y_1 és y_2 együtt is önfüggő csoportot képez. A kódolás alapjául szolgáló h. t. partíció kiválasztásakor mindig érdemes mérlegelni a fentiekhez hasonló lehetőségeket.

	y_1	y_2	y_3
a	0	0	0
b	1	0	0
c	1	0	1
d	0	1	0
e	1	1	0
f	0	1	1
g	1	1	1
Nem használt	–	0	0	1

3.123. ábra. Önfüggő szekunder változócsoportok képzése alapján meghatározott kódolási változat a 3.112. ábrán szereplő állapottáblához

	$y_1y_2y_3$ \ x_1x_2	00	01	11	10
a	000	100 0	101 0	100 0	101 0
b	100	011 0	010 0	011 0	010 0
c	101	010 0	010 0	010 0	010 0
d	010	110 0	110 0	110 0	110 0
e	110	0000	0000	0000	0000
f	011	111 0	110 0	111 0	110 0
g	111	000 1	000 0	000 1	0000

3.124. ábra. Kódolt állapottábla a 3.123. ábra kódolási változata alapján

A választott kódolás (3.123. ábra) alapján a 3.124. ábrán láthatjuk a kódolt állapottáblát, amelyből az alábbi logikai függvények képezhetők:

$$R_1 = y_1,$$

$$S_1 = \bar{y}_1,$$

$$R_2 = y_1 y_2,$$

$$S_2 = y_1 \bar{y}_2,$$

$$R_3 = y_1 y_3 + \bar{x}_1 x_2 y_2 + x_1 \bar{x}_2 y_2,$$

$$S_3 = \bar{x}_1 x_2 \bar{y}_1 \bar{y}_2 + x_1 \bar{x}_2 \bar{y}_1 \bar{y}_2 + \bar{x}_1 \bar{x}_2 y_1 \bar{y}_2 \bar{y}_3 + x_1 x_2 y_1 \bar{y}_2 \bar{y}_3,$$

$$Z = \bar{x}_1 \bar{x}_2 y_1 y_2 y_3 + x_1 x_2 y_1 y_2 y_3.$$

A szükséges kapubemenetek száma tehát 49. Az algebrai alakok alapján jól megfigyelhetők az önfüggő szekunder változócsoportok.

A következőkben egy egyszerű példán szemléltetjük, hogy a feltétlenül szükségesnél több szekunder változó alkalmazása egyszerűsítheti a vezérlő kombinációs hálózatot.

Keressünk h. t. partíciókat a 3.125. ábrán látható állapottáblán. Az a és b állapotokat egy blokkba tartozónak feltételezve a 3.126. ábrán bemutatott lépcsős táblát kapjuk. Ennek alapján:

$$(abcd),$$

$$(bcd)(ab),$$

$$(cd)(b)(ab),$$

tehát a

$$\Pi_1 = \{(ab)(cd)\}$$

h. t. partícóhoz jutottunk

y \ x	0	1
a	c 0	b 1
b	d 0	–
c	b 0	–
d	b 1	a 0

3.125. ábra. Példaképpeni állapottábla a h. t. partíciók alapján történő állapotkódolás során felvett többlet szekunder változók hatásának szemléltetéséhez

	a	b	c
b	cd ✓		
c	bc	bd	
d	bc ab	bd	✓

3.126. ábra. A 3.125. ábrán szereplő állapottábla h. t. partícióinak kereséséhez képzett lépcsős tábla az *a* és *b* állapotok egy blokkba tartozásának kiindulási feltételezésével

	y_1	y_2
a	0	0
b	0	1
c	1	1
d	1	0

3.127. ábra. Kódolási változat a 3.125. ábrán szereplő állapottáblához

A lépcsős táblán láthatjuk, hogy *c* és *d* állapotok feltétel nélkül lehetnek egy blokkban, tehát létezik a

$$\Pi_2 = \{(a)(b)(cd)\}$$

h. t. partíció is. Π_1 alapján két szekunder változóval, Π_2 alapján hárommal kódolhatunk.

Először Π_1 alapján végezzük el a kódolást. Ennek egy lehetséges változata a 3.127. ábrán látható. A blokkokat y_1-gyel, a blokkokon belüli állapotokat pedig y_2-vel különböztettük meg. A kódolt állapottábla (3.128. ábra) alapján a megvalósítást végezzük el T, majd D flip-floppal. Az ismert lépéseket mellőzve csak a végeredményeket adjuk meg:

$$T_1 = \bar{x}+y_1,$$
$$T_2 = \bar{y}_1+\bar{x}\bar{y}_2,$$
$$Z = \bar{x}y_1\bar{y}_2+x\bar{y}_1,$$
$$D_1 = \bar{x}\bar{y}_1,$$
$$D_2 = \bar{x}y_1+\bar{y}_1\bar{y}_2,$$
$$Z = \bar{x}y_1\bar{y}_2+x\bar{y}_1.$$

T flip-flop esetén 13, D flip-flop esetén pedig 15 kapubemenetet igényel a hálózat.

Kódoljunk ezután Π_2 szerint. A 3.129. ábrán megfigyelhetjük, hogy y_1 és y_2 különbözteti meg a blokkokat, y_3 pedig a blokkokon belüli állapotokat. A *c* és *d* állapotok

y_1y_2 \ x	0	1
a 00	110	011
b 01	100	–
c 11	010	–
d 10	011	000

3.128. ábra. Kódolt állapottábla a 3.127. ábra kódolási változata alapján

	y_1	y_2	y_3
a	0	0	–
b	0	1	–
c	1	–	0
d	1	–	1

3.129. ábra. Kódolási változat többlet szekunder változó alkalmazásával a 3.125. ábrán szereplő állapottáblához

kódjában y_2 értéke tetszőleges lehet, mert $y_1=1$ elegendő (*cd*)-nek (*a*)-tól és (*b*)-től való megkülönböztetéséhez. Az *a* és *b* állapotok kódjában y_3 értékét azért választhatjuk tetszőlegesnek, mert (*a*) és (*b*) egyetlen állapotot tartalmazó blokkok, így ezeken belül nincs szükség az állapotok megkülönböztetésére.

A kódolt állapottábla (3.130. ábra) kitöltésekor ügyelnünk kell arra, hogy a nem rögzített szekunder változóértékek miatt esetünkben minden állapotnak két sor felel meg, így a következő állapotokat mindkettőbe be kell írnunk. Az ismert lépéseket ezúttal sem részletezzük, csak a végeredményt adjuk meg T, ill. D flip-flop alkalmazása esetén:

$$T_1 = \bar{x}+y_1,$$
$$T_2 = \bar{y}_1+\bar{x}\bar{y}_2+xy_2,$$
$$T_3 = \bar{y}_2y_3+y_2\bar{y}_3,$$
$$Z = \bar{x}y_1y_3+x\bar{y}_1,$$
$$D_1 = \bar{x}\bar{y}_1,$$
$$D_2 = \bar{y}_1+\bar{x},$$
$$D_3 = y_2,$$
$$Z = \bar{x}y_1y_3+x\bar{y}_1.$$

T flip-flop esetén 22, D flip-flop esetén pedig 11 kapubemenet szükséges.

Oldjuk meg ezek után a feladatot úgy is, hogy a Π_2 szerinti kódolásban rögzítsük minden állapotban az összes szekunder változó értékét. A fel nem használt kombinációkról tételezzük fel, hogy sohasem alakulnak ki, miáltal az ezeknek megfelelő sorokban az állapottáblán sem a kimeneti értékeket, sem az állapotokat nem kell specifikálnunk. Rögzítsük a kódokat a 3.131. ábra szerint. Így a 3.132. ábrán látható

	$y_1y_2y_3$ \ x	0	1
a	000	1-00	01-1
	001	1-00	01-1
b	011	1-10	-
	010	1-10	-
	110	01-0	-
c d	111	01-1	00-0
	101	01-1	00-0
	100	01-0	

3.130. ábra. Kódolt állapottábla a 3.129. ábra kódolási változata alapján

		y_1	y_2	y_3
	a	0	0	0
	b	0	1	0
	c	1	0	0
	d	1	0	1
Nem használt	–	0	0	1
	–	0	1	1
	–	1	1	0
	–	1	1	1

3.131. ábra. Kódolási változat többlet szekunder változó alkalmazásával, rögzített szekunder változóértékekkel a 3.125. ábrán szereplő állapottáblához

kódolt állapottábla adódik, amelynek alapján végeredményként az alábbi függvényeket kapjuk:

$$T_1 = \bar{x} + y_1,$$
$$T_2 = y_2 + \bar{x}y_1 + x\bar{y}_1,$$
$$T_3 = y_2 + y_3,$$
$$Z = x\bar{y}_1 + \bar{x}y_1y_3,$$
$$D_1 = \bar{x}\bar{y}_1,$$
$$D_2 = x\bar{y}_1 + \bar{x}y_1,$$
$$D_3 = y_2,$$
$$Z = x\bar{y}_1 + \bar{x}y_1y_3.$$

A kapubemenetek száma T flip-flop esetén 18, D flip-flop esetén 15.

A példa alapján a következőket állapíthatjuk meg:

a) Ha többlet szekunder változót igénylő h. t. partíció alapján választjuk meg a kódot, akkor ennek *lehet olyan hatása,* hogy a vezérlő kombinációs hálózat egyszerűsödik (D flip-flop esetén két szekunder változóval 15 kapubemenet, háromma az első kitöltési mód szerint 11 kapubemenet).

	$y_1y_2y_3$ \ x	0	1
a	000	100 0	010 1
–	001	–	–
–	011	–	–
b	010	101 0	–
–	110	–	–
–	111	–	–
d	101	010 1	000 0
c	100	010 0	–

3.132. ábra. Kódolt állapottábla a 3.131. ábra kódolási változata alapján

b) Ennek a hatásnak az érvényesülése *nem független* attól, hogy milyen flip-floppal realizáljuk a hálózatot. (T flip-flop esetén nem tapasztaltunk egyszerűsödést.)

c) A kialakuló hálózat egyszerűségét *befolyásolja,* hogy a nem rögzített kódokat hogyan használjuk ki (különböző bonyolultságú hálózatok adódtak a két kitöltési mód esetén).

d) A nem rögzített kódok felhasználásának célszerű módját az *alkalmazott flip-flop típus is befolyásolja.* (T flip-flop esetén a második, D flip-flop esetén az első kitöltési mód adott egyszerűbb hálózatot.)

A szinkron sorrendi hálózatok állapotkódolásának megválasztására bemutatott két eljárás jól szemlélteti, hogy a legkedvezőbb kódolás meghatározása mennyire nem egyértelműen megfogalmazható feladat. Mindkét módszerrel tulajdonképpen csak arra vagyunk képesek, hogy a kódválasztáskor néhány egyszerűsítő hatást érvényesítsünk a kialakuló hálózatra vonatkozóan és nagy valószínűséggel egyszerűbb hálózatot kapjunk, mint egy találomra felvett kódolással.

Léteznek olyan — számítógépes végrehajtást igénylő — kódolási eljárások, amelyek lényege az, hogy segítségükkel viszonylag sok kódolási változat várható hatását tudjuk előre becsülni valamilyen tapasztalati úton származtatott mennyiségi jellemző alapján, amelyet a kódolási változatokra rendre kiszámítunk. Így a figyelembe vett kódolási változatok közül kiválasztható a legkedvezőbbnek tűnő. Nyilvánvaló azonban, hogy ezekről a módszerekről sem állítható, hogy az optimális megoldást szolgáltatják, hiszen, mint láttuk, ezt csak próbálgatással dönthetjük el és az optimalitási követelmény sem mindig fogalmazható meg kezelhető módon. Például a vezérlő kombinációs hálózat egyszerűsítése gyakran lehetséges többlet szekunder változó alkalmazásával, ami viszont pótlólagos flip-flopokat igényel a szinkron sorrendi hálózatban.

Mint láttuk, a kódválasztás után megvalósított sorrendi hálózat állapotainak száma nem szükségképpen egyezik meg a kiindulási állapottábla állapotainak számával. Ennek oka, hogy a megvalósított hálózatnak mindenképpen 2^p számú állapota van, ha a kódolást p számú szekunder változóval végeztük. A kiindulási állapottáblához képest még akkor is több állapot adódhat, ha a kódolás nem tartalmaz többlet sze-

kunder változót, hiszen mindenképpen fennáll az alábbi egyenlőtlenség:

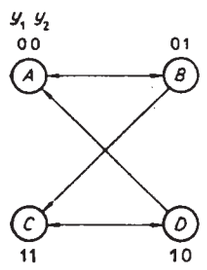
$$2^p \geqq r > 2^{p-1},$$

ahol r jelöli a kiindulási állapottábla állapotainak (sorainak) számát.

A kódolt állapottábla és a vezérlési tábla kitöltése egyértelmű ha $2^p=r$, vagy a megvalósított hálózatnak nincsenek pótlólagosan kiadódott állapotai. Ha vannak ilyenek, vagyis $2^p>r$, akkor kétféleképpen járhatunk el, amint azt a 3.130. és 3.132. ábrákon bemutattuk. Azt is szemléltettük, hogy a két kitöltési mód közötti választás során sem nélkülözhetjük a próbálgatást, hiszen az esetenként kialakuló hálózat egyszerűségét csak így tudjuk megítélni és az egyes kitöltési módok hatása az alkalmazott flip-flop típustól is függ.

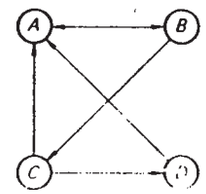
3.9.2. Aszinkron sorrendi hálózatok állapotkódolása

Az aszinkron sorrendi hálózatok tervezési lépéseinek bemutatásakor láttuk, hogy az állapotkódok helyes megválasztásával tudjuk kiküszöbölni a kritikus versenyhelyzetből fakadó hibás működést. A kódválasztás elsődleges szempontja tehát aszinkron esetben nem a kialakuló hálózat egyszerűsége, hanem a helyes működés biztosítása a szekunder változók tetszőleges egymáshoz képesti működésisebesség-aránya esetén is. Könnyen belátható, hogy a kritikus versenyhelyzetet biztosan kiküszöböljük, ha a közvetlenül egymás utáni állapotokhoz szomszédos kódokat rendelünk. Ilyenkor ugyanis az összes állapotváltozás során mindig csak egyetlen szekunder változó változtatja meg értékét, ami nyilvánvalóan kizárja a versenyhelyzet kialakulásának lehetőségét. Ezt a megoldást választottuk a tervezési lépések bemutatásakor a 3.73. ábra szerinti állapotkódolás alkalmazásával. A kódok szomszédossági követelményeinek szemléltetésére felhasználhatjuk az állapotgráfnak egy egyszerűsített változatát, amely csak az állapotátmenetek tényét ábrázolja, vagyis nem jelzi a bemeneti és kimeneti kombinációkat, valamint a stabil állapotokat. Ennek megfelelően a 3.71. ábra állapotgráfját a 3.133. ábrán látható alakra hozhatjuk. Az ábrán feltüntettük a 3.73. ábrán bemutatott versenyhelyzetmentes kódolású állapottáblát létrehozó állapotkódokat is. Látszik, hogy a szomszédos kódok hozzárendelésekor az egyszerűsített állapotgráf alapján az a formális szabály érvényes,



3.133. ábra. A 3.71. ábrán szereplő állapotgráf egyszerűsített változata a közvetlen állapotátmenetek, vagyis a kódok szomszédossági feltételeinek szemléltetése céljából

y \ x_1x_2	00	01	11	10
A	A 0	A 0	A 0	B 0
B	C -	–	A 0	B 0
C	C 1	A -	–	D 1
D	A -	–	A -	D 1



3.134. ábra. Példa olyan aszinkron állapottáblára, amely nem teszi lehetővé az összes szomszédossági feltétel teljesítését az állapotok kódolásakor

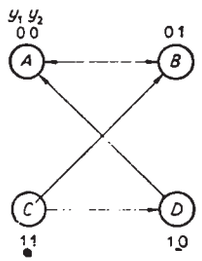
hogy egymáshoz képest szomszédosan kell kódolni mindazokat az állapotokat, amelyeket közvetlen élek kötnek össze, vagyis amelyek között közvetlen átmenet játszódhat le. Az így megfogalmazható szomszédossági feltétel azonban nem mindig teljesíthető olyan egyszerűen, mint a 3.133. ábra esetében. Nyilvánvaló, hogy az egyszerűsített állapotgráf elrendeződése, vagyis maga a megoldandó logikai feladat lehet olyan, hogy nem tudjuk az összes szomszédossági feltételt kielégíteni a kódválasztáskor. Ilyen esetet szemléltet a 3.134. ábrán szereplő állapottáblával adott feladat. Az egyszerűsített állapotgráf alapján láthatjuk, hogy a szomszédossági feltételeket a négy állapot által igényelt két szekunder változóval már csak azért sem tudjuk teljesíteni, mert pl. az A állapotnak három szomszédja van. Ugyanakkor pótlólagos szekunder változók alkalmazásával sem hozhatjuk létre az előírt kódolást, mert az egyszerűsített állapotgráfon megfigyelhető A—B—C és A—C—D háromszögek mindegyike egyidejűleg teljesíthetetlen szomszédossági követelményeket ábrázol.

A továbbiakban először két eljárást mutatunk be a fentiekben vázolt nehézségek leküzdésére és a helyes aszinkron működést biztosító szomszédos kód választására. Ezután szisztematikus módszert ismerünk meg a helyes kódválasztásra, amely nem a szomszédosságon, hanem a kritikusversenyhelyzet-mentesség szükséges és elégséges feltételén alapul.

3.9.2.1. Instabil állapotok módosítása

A 3.134. ábra egyszerűsített állapotgráfja módosul, ha az állapottáblában módosítunk néhány instabil állapotot. A 3.135. ábrán bemutatott állapottábla csupán a *-gal jelölt instabil állapotokban különbözik a 3.134. ábrán megadott állapottáblától. Ez a módosítás megszünteti a C és A állapotok közötti közvetlen átmenetet. Így az egyszerűsített állapotgráf is a 3.135. ábra szerint módosul, miáltal a szomszédos-

y \ x_1x_2	00	01	11	10
A	Ⓐ0	Ⓐ0	Ⓐ0	B 0
B	C –	A 0*	A 0	Ⓑ0
C	Ⓒ1	B –*	–	D 1
D	A –	–	A –	Ⓓ1



3.135. ábra. A 3.134. ábrán szereplő állapottábla módosítása a szomszédossági feltételek teljesíthetősége céljából

sági feltételek teljesíthetőkké válnak pl. az alábbi kódválasztással:

	y_1	y_2
A	0	0
B	0	1
C	1	1
D	1	0

Az eljárás lényege tehát az, hogy az instabil állapotokat és közömbös bejegyzéseket célszerűen módosítva olyan közvetlen állapotátmeneteket szüntetünk meg, azaz teszünk közvetettekké, amelyek gátolták a szomszédossági feltételek teljesítését. Könnyen meggyőződhetünk arról, hogy példánkban más instabil állapotok módosításával is hasonló eredményt kaptunk volna. A bemutatott megoldásban megfigyelhetjük, hogy az $x_1x_2=01$ oszlop B sorában a közömbös bejegyzést A0-ra rögzítettük, holott logikailag A– is megfelelne a célnak. Ennek oka az, hogy a közömbös kimeneti kombinációk a kétszeri instabilállapot-változás során kimeneti impulzust hozhatnak létre, ha a megvalósításkor tetszőlegesen rögzíthetnénk azokat. Mindez lényegében annak a következménye, hogy a bemutatott eljárás alkalmazásával megszüntetjük a hálózat normál jellegét, hiszen a módosított állapottáblán már lesznek olyan állapotátmenetek is, amelyek során egynél több instabil állapot lép fel. Ez egyrészt lényegében lassítja a hálózat működését, másrészt pedig megnehezíti a hazárdjelenségek vizsgálatát.

A fentiekben szemléltetett módszer intuitív jellegű, általában a próbálgatást sem mellőzhetjük és eredményessége az állapottáblától, vagyis a hálózat által megoldandó logikai feladattól függ. Könnyen képezhetünk olyan állapottáblát, amelyhez az ismertetett módon nem tudunk állapotkódokat hozzárendelni. Nem mindig kínálkozik lehetőség az instabil állapotok olyan módosítására vagy rögzítésére, amely bizonyos

közvetlen állapotátmenetek megszüntetésével a szomszédossági feltételeket teljesíthetővé teszi.

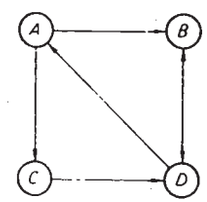
A következőkben olyan eljárást mutatunk be, amely minden esetben lehetővé teszi — általában többlet szekunder változó alkalmazásával — a szomszédossági feltételek teljesítését.

3.9.2.2. Átvezető állapotok felvétele

Az ún. *átvezető állapotok* rendeltetése az, hogy a szomszédossági feltételeket megfelelően módosítsák. Példaként induljunk ki a 3.136. ábrán szereplő állapottáblából. A 3.137. ábrán mutatunk be egy lehetséges megoldást az átvezető állapotok felvételére. Az egyszerűsített állapotgráfon jól megfigyelhetjük a Q_1 és Q_2 állapotok szerepét az állapotátmenetek módosításában és így a szomszédossági feltételek átalakításában. Az átvezető állapotok természetesen nem változtatják meg a kiindulási állapottáblán a stabil állapotok alapján követhető működést, hiszen a közbeiktatott átvezető állapotok soha nem stabilizálódnak, hanem mindig a megfelelő eredeti stabil állapotba vezetik tovább a hálózatot. Éppen az így közbeiktatódó instabil állapotok teszik lehetővé a szomszédos kódolást, amelynek egy lehetséges változata a 3.137. ábra alapján

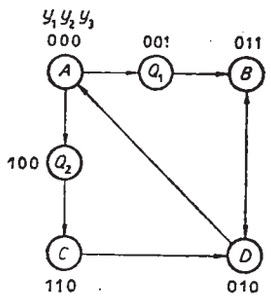
	y_1	y_2	y_3
A	0	0	0
B	0	1	1
C	1	1	0
D	0	1	0
Q_1	0	0	1
Q_2	1	0	0

y \ x_1x_2	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 –



3.136. ábra. Aszinkron állapottábla az átvezető állapotok felvételével történő állapotkódolás bemutatásához

x_1x_2 / y	00	01	11	10
A	Ⓐ0	Q_2-*	$Q_1$0	Ⓐ0
B	D -	D -	Ⓑ0	Ⓑ0
C	D1	Ⓒ1	Ⓒ1	Ⓒ1
D	Ⓓ1	Ⓓ1	B -	A -
Q_1	–	–	B0	–
Q_2	–	C-*	–	–



3.137. ábra. A 3.136. ábrán szereplő állapottábla módosítása a Q_1 és Q_2 jelű átvezető állapotok felvétele révén

A felvett átvezető állapotok megkülönböztetésének szükségessége miatt általában több szekunder változóra van szükség, mint amennyit a kiindulási állapottábla sorainak száma igényelne. A szekunder változók kihasználatlan kombinációi szomszédos kódolás esetén természetesen nem léphetnek fel a hálózat működése során, ezért az ezeknek megfelelő sorok a kódolt állapottáblában csupa közömbös bejegyzéssel tölthetők ki. A 3.137. ábrán megfigyelhetjük, hogy a Q_1 és Q_2 sorokban levő bejegyzések formailag megengednék a Q_1 és Q_2 állapot összevonását. Ezt természetesen nem végezzük el, hiszen ezáltal a szomszédossági feltételeket ismét módosítanánk és újra lehetetlenné tennénk a szomszédos kódolást.

A 3.137. ábra alapján történő megvalósításkor ügyelnünk kell arra, hogy a 01 oszlopban *-gal jelölt közömbös kimeneti értékeket nem rögzíthetjük egymástól függetlenül. El kell kerülni ugyanis, hogy az Ⓐ 0, Q_2–, C–, Ⓒ 1 állapotátmenet Ⓐ 0, $Q_2$1, C0, Ⓒ 1 formájában rögzítődjön, ami a kimeneten hibás impulzust okozhat az $x_1x_2=01$ bemeneti kombináció fellépésekor.

Megfelelő számú átvezető állapot felvételével mindig módosíthatók úgy a szomszédossági feltételek, hogy azok teljesíthetők legyenek a kódválasztáskor. Az átvezető állapotok minimális számának és szükséges helyének meghatározása általában próbálgatást igényel. Az instabil állapotok módosításának módszeréhez hasonlóan ezúttal is megszűnik a hálózat normál jellege, ami hátrányos a sebesség és a hazárdjelenségek vizsgálhatósága szempontjából.

3.9.2.3. Állapotkódolás megkülönböztető partíciók alapján (Tracey—Unger-módszer)

Az eddigiekben láttuk, hogy a kritikus versenyhelyzeteket biztosan kiküszöböljük, ha a közvetlenül egymás utáni állapotokhoz szomszédos kódokat rendelünk. Az ilyen szomszédoskód-választás tehát a kritikusversenyhelyzet-mentesség elégséges feltétele. A továbbiakban bemutatjuk, hogy ez a feltétel nem szükséges, vagyis a kritikusversenyhelyzet-mentességhez nem kell feltétlenül a fenti értelemben szomszédosaknak lenniük az állapotkódoknak. Egy bemeneti kombináció fellépésekor az annak oszlopában lejátszódó állapotátmenetek attól függenek, hogy melyik előző bemeneti kombináció állt fenn és annak oszlopában melyik stabil állapotban volt a hálózat. Kritikus versenyhelyzet akkor, és csak akkor alakulhat ki, ha egy bemeneti kombináció fellépésekor lejátszódó állapotátmenet során átmenetileg létrejöhet olyan szekunder kombináció, amely megegyezik az oszlopban levő valamelyik másik állapotátmenet során létrejöhető vagy az oszlopban levő valamelyik másik stabil állapotnak megfelelő szekunder kombinációval.

Példaként vizsgáljuk meg a 3.136. ábrán szereplő állapottáblát. Pl. az $x_1x_2=00$ fellépésekor lejátszódhat

a) a B—D állapotátmenet, ha a megelőző bemeneti kombináció $x_1x_2=10$ volt és B stabil állapotban volt a hálózat;

b) a C—D állapotátmenet, ha a megelőző bemeneti kombináció $x_1x_2=01$ vagy $x_1x_2=10$ volt és C stabil állapotban volt a hálózat;

c) az A stabil állapot fennmaradása, ha a megelőző bemeneti kombináció $x_1x_2=10$ volt és az A stabil állapotban volt a hálózat,

d) a D stabil állapot fennmaradása, ha a megelőző bemeneti kombináció $x_1x_2=01$ volt és a D stabil állapotban volt a hálózat.

Ugyanígy foglalhatjuk össze a többi bemeneti kombináció fellépésekor lejátszódó változásokat. A mindenkori megelőző bemeneti kombináció sorravételekor természetesen ügyelnünk kell arra, hogy az aszinkron hálózatok esetén kizárólag szomszédos bemeneti változásokat tételeztünk fel már a tervezéskor is. Ezért pl. az $x_1x_2=00$ fellépésekor nem vizsgáltuk az $x_1x_2=11$ oszlopbeli stabil állapotokból kiinduló átmeneteket, bár az adott esetben ez nem okozott volna a felsoroltakon kívül további állapotátmeneteket.

Fogalmazzuk meg először csak az $x_1x_2=00$ oszlopbeli, fentiekben összefoglalt állapotváltozásokra vonatkozóan, hogy milyen állapotkódok szükségesek az $x_1x_2=00$ oszlopbeli kritikus versenyhelyzet elkerüléséhez:

a) A B—D állapotátmenet során ne alakulhasson ki az A állapotnak megfelelő szekunder kombináció;

b) a C—D állapotátmenet során ne alakulhasson ki az A állapotnak megfelelő szekunder kombináció.

A B—D és C—D átmenetek során kialakulhatnak közös szekunder kombinációk,

hiszen mindkét változás a D stabil állapotba vezet, vagyis legfeljebb nemkritikus versenyhelyzet léphet fel.

Az állapotkódolásnak tehát a fenti *a)* és *b)* pontbeli feltételt kell kielégítenie. Az *a)* pontbeli feltétel biztosan teljesül, ha *legalább egy szekunder változó értéke megkülönbözteti a kódban a BD állapotpárt az A állapottól.* Ekkor ugyanis ennek a megkülönböztető szekunder változónak az értéke a B—D átmenet során állandó és ellentétes az A állapot kódjában felvett értékével, vagyis csak olyan átmeneti (tranziens) szekunder kombináció alakulhat ki a B—D átmenet során, amely eltérő az A állapot kódjától.

Így nem következhet be hibás stabilizálódás az A állapotban. Az így feltételezett megkülönböztető szekunder változó tulajdonképpen az A, B, D állapotok egy kétblokkos partícióját hozza létre, amelyben a megkülönböztetendő állapotok kerülnek külön blokkba:

$$\Pi_a = (A, BD).$$

Az előző gondolatmenettel a *b)* pontbeli feltétel biztosan teljesül, *ha legalább egy szekunder változó értéke megkülönbözteti a kódban a CD állapotpárt az A állapottól.* Ez a megkülönböztető szekunder változó az alábbi állapotpartíciót hozza létre:

$$\Pi_b = (A, CD).$$

A feltételezett megkülönböztető szekunder változók által létrehozott állapotpartíciókat a továbbiakban *megkülönböztető partíciókat* nevezzük.

Nyilvánvaló, hogy a hálózat biztosan mentes lesz a kritikus versenyhelyzettől, ha az állapottábla minden egyes oszlopában a bemutatott gondolatmenettel megvizsgáljuk a lehetséges állapotátmeneteket és a kiadódó megkülönböztető partíciók előírásai alapján képezzük az állapotkódokat. Ha ugyanis minden megkülönböztető partíciót legalább egy szekunder változó értékével megvalósítunk az állapotkódokban, akkor minden szükséges állapotváltozás-megkülönböztetést elvégzünk. Példánkban az alábbiakban foglalhatjuk össze a bemeneti kombinációnkénti megkülönböztető partíciókat:

$x_1x_2 = 00$	$x_1x_2 = 01$	$x_1x_2 = 11$	$x_1x_2 = 10$
$\Pi_a = (A, BD)$	$\Pi_c = (AC, D)$	$\Pi_f = (AB, C)$	$\Pi_g = (AD, B)$
$\Pi_b = (A, CD)$	$\Pi_d = (AC, BD)$		$\Pi_h = (AD, C)$
	$\Pi_e = (BD, C)$		

Ha egy partíció több oszlopban is kiadódik, akkor természetesen elegendő egyszer figyelembe venni a felíráskor. Így pl. az $x_1x_2=11$ oszlop alapján képezhető (BD, C) partíciót azért nem írtuk fel újra, mert az $x_1x_2=01$ oszlopban Π_e-vel jelölve már figyelembe vettük.

Figyeljük meg, hogy a Π_d partíció két állapotátmenetet különböztet meg az $x_1x_2=01$ oszlopban. A kritikus versenyhelyzet kiküszöböléséhez ugyanis szükséges, hogy az állapotkódokban legalább egy szekunder változó értéke állandó, de ellentétes legyen a két állapotátmenet során. Csak ekkor állíthatjuk ugyanis, hogy az egyik állapotátmenet során nem alakulhat ki olyan átmeneti szekunder kombináció, amely a másik állapotátmenet során is létrejöhet. Ha felléphetnének a két átmenet során azonos átmeneti szekunder kombinációk, akkor a kialakuló stabil állapot a szekunder változók egymáshoz képesti működési sebességétől függően a két átmenet stabil állapotainak bármelyike lehetne, amit kritikus versenyhelyzetként definiáltunk.

Az összes megkülönböztető partíció felírása után kritikusversenyhelyzet-mentes kódokat kaphatunk, ha minden megkülönböztető partíció előírását külön szekunder változóval valósítjuk meg az alábbi módon:

	y_a	y_b	y_c	y_d	y_e	y_f	y_g	y_h
A	0	0	0	0	—	0	0	0
B	1	—	—	1	0	0	1	—
C	—	1	0	0	1	1	—	1
D	1	1	1	1	0	—	0	0

(3.9.21.)

A $\Pi_a \ldots \Pi_h$ partíciókat létrehozó szekunder változókat rendre $y_a \ldots y_h$-val jelöltük. A 0 és 1 értékek hozzárendelését az egyes blokkokhoz természetesen ellentétesen is végezhettük volna bármelyik partíció esetén. A megkülönböztetés ténye ugyanis nem függ a megkülönböztető szekunder változó kódbeli értékétől, ha az a megkülönböztetendő átmeneteket, ill. stabil állapotokat megfelelően elkülöníti egymástól. Az y_c szekunder változó például a Π_c partíciót valósítja meg és önkényesen rendeltünk hozzá az A, C állapotokhoz 0-t és a D állapothoz 1-et, az ellentétes hozzárendelés ugyanígy megvalósította volna Π_c-t. A partícióban nem szereplő állapotok kódjában a partíciót megvalósító szekunder változó értéke természetesen tetszőleges lehet. Ezért írtunk ezekre a helyekre közömbös bejegyzést.

Az így kiadódó állapotkódok tehát biztosítják a kritikusversenyhelyzet-mentességet, de általában feleslegesen sok szekunder változót igényelnek. Egyetlen szekunder változóval ugyanis általában több megkülönböztető partíciót is megvalósíthatunk. Figyeljük meg például, hogy ha y_d-vel Π_d-t létrehozzuk, akkor ezzel egyúttal a Π_c-ben előírt elkülönítés is teljesül, vagyis y_c-re nincs szükség. Az állapotkódok fenti felírásából ezt formálisan úgy is észrevehetjük, hogy y_c és y_d oszlopában egyetlen állapothoz sem tartozik ellentétes y_c és y_d érték, vagyis a két oszlop *kompatibilis.* Ezért az y_c és y_d oszlopok összevonhatók a kompatibilitás ismert szabályai szerint:

	$\ldots y_{cd} \ldots$
A	0
B	1
C	0
D	1

ahol y_{cd} a Π_d partíciót állítja elő.

Hasonló módon látható be Π_g és Π_h egyetlen szekunder változóval történő teljesíthetősége:

	$\ldots y_{gh} \ldots$
A	0
B	1
C	1
D	0

ahol y_{gh} a $\Pi_{gh}=(AD, BC)$ partíciót hozza létre, amely a kiindulási megkülönböztető partíciók között nem szerepelt ugyan, de összefoglalja Π_g és Π_h elkülönítő előírásait. Ugyancsak felesleges az y_e szekunder változó felvétele, mert Π_d megvalósítása esetén Π_e elkülönítő előírása is teljesül. A kódok fenti felírása esetén az y_e oszlop úgy válhat kompatibilissá az y_d oszloppal, hogy rendre y_e negált értékeit értelmezzük. Ezt természetesen mindig megtehetjük, hiszen a negált szekunder változóértékek az elkülönítő előírásokat nem befolyásolják. Megfigyelhetjük, hogy eleve a negált értékeket kaptuk volna az eredetileg alkalmazott 1—0 hozzárendeléssel, ha Π_e blokkjait fordított sorrendben írjuk fel, ami a megkülönböztetés ténye szempontjából valóban egyenértékű megoldás lett volna. Így tehát y_d és $\bar{y}_e$ oszlopok összevonásának eredménye:

	$\ldots y_{d\bar{e}} \ldots$
A	0
B	1
C	0
D	1

ahol $y_{d\bar{e}}$ a Π_d partíciót állítja elő.

A továbbiakban arra keresünk szisztematikus eljárást, hogy az *összes megkülönböztető partíciót a lehető legkevesebb szekunder változóval valósítsuk meg*. Ehhez felhasználhatjuk a kompatibilitás és az állapot-összevonási eljárás megismert formális szabályait. A (3.9.21.) állapotkódokat formálisan mátrixsoroknak tekintve, a feladat megoldásának első lépése az alábbi módon fogalmazható meg.

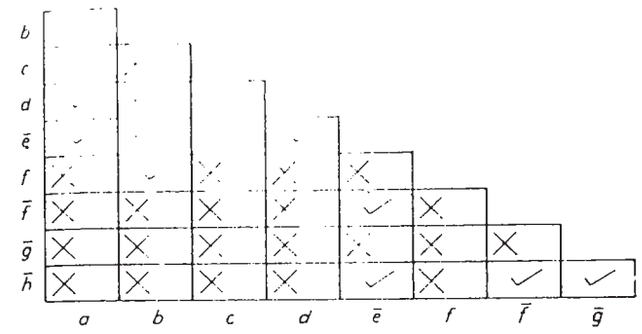
1. Keressük meg a (3.9.21) mátrix oszlopainak maximális kompatibilitási osztályait. Mint láttuk a páronkénti kompatibilitási vizsgálathoz az egyes oszlopok negáltját is figyelembe kell vennünk. Emiatt a lépcsős táblán minden oszlop negáltját is fel kellene tüntetni. Ez elkerülhető, ha az alábbi eljárást követjük.

— Válasszuk ki a mátrixnak azt a sorát, ahol a legkevesebb közömbös bejegyzés van. Példánkban ilyen az A vagy a D sor. Válasszuk D-t.

— Képezzük a megfelelő oszlopok negáltját úgy, hogy a kiválasztott sorban az összes határozott érték azonos legyen. Például a kiválasztott D sorban legyen az összes határozott érték 1. Ehhez az y_e, y_g és y_h oszlopok negáltjait kell képezni:

	y_a	y_b	y_c	y_d	$\bar{y}_e$	y_f	$\bar{y}_g$	$\bar{y}_h$
A	0	0	0	0	—	0	1	1
B	1	—	—	1	1	0	0	—
C	—	1	0	0	0	1	—	0
D	1	1	1	1	1	—	1	1

(3.9.22.)



3.138. ábra. Lépcsős tábla a 3.136. ábrán megadott hálózat állapotkódolásának meghatározásához a megkülönböztető partíciók alapján

Figyeljük meg, hogy ha az A sort választottuk volna, akkor nem lett volna szükség oszlopok tagadására, mert az A sorban az összes specifikált bejegyzés azonos. Csak azért választottuk a D sort, hogy a tagadott oszlopok képzésére vonatkozó lépést is bemutassuk.

Ezek után igaz az az állítás, hogy kompatibilitásvizsgálathoz a (3.9.22.) mátrix alapján képezhetjük a lépcsős táblát, és csak azokat az oszlopokat kell tagadás nélkül és tagadottan is feltüntetni, amelyek a kiválasztott sorban (példánkban a D-ben) közömbös bejegyzést tartalmaznak (példánkban csak az y_f oszlop ilyen). Az állítás helyessége viszonylag egyszerű gondolatmenettel bizonyítható. Mivel ez a további vizsgálatainkhoz nem feltétlenül szükséges, ezért mellőzzük.

A (3.9.22.) mátrix alapján a fenti meggondolásokkal képezhető lépcsős tábla a 3.138. ábrán látható. Az egyes oszlopok jelölésére csak a megfelelő szekunder változó indexét használtuk. A páronkénti kompatibilitásvizsgálat ismert szabályait alkalmazva ezúttal természetesen csak $\surd$ vagy $\times$ bejegyzést kaphatunk, hiszen az összehasonlítandó oszlopokban most nem állapotszimbólumok, hanem logikai értékek szerepelnek, így feltételes kompatibilitás nem állhat fenn.

Az oszlopok maximális kompatibilitási osztályait a kitöltött lépcsős tábla alapján az ismert módszerrel határozhatjuk meg:

$(abcd\bar{e}f\bar{f}\bar{g}\bar{h})$,

$(bcd\bar{e}f\bar{f}\bar{g}\bar{h})(abcd\bar{e})$,

$(cd\bar{e}f\bar{f}\bar{g}\bar{h})(bf)(acd\bar{e})(ab)$,

$(d\bar{e}f\bar{f}\bar{g}h)(cd\bar{e})(bf)(acd\bar{e})(ab)$,

$(\bar{e}f\bar{f}\bar{g}\bar{h})(d)(bf)(acd\bar{e})(ab)$,

$(f\bar{f}\bar{g}\bar{h})(\bar{e}\bar{f}\bar{h})(bf)(acd\bar{e})(ab)$,

$(\bar{f}\bar{g}\bar{h})(\bar{f})(\bar{e}\bar{f}\bar{h})(bf)(acd\bar{e})(ab)$,

$(\bar{g}\bar{h})(\bar{f}\bar{h})(\bar{e}\bar{f}\bar{h})(bf)(acd\bar{e})(ab)$.

2. A maximális kompatibilitási osztályok ismeretében rátérhetünk a lehető legkevesebb szekunder változóval megvalósított kódolás meghatározásának második lé-

Max. komp. osztályok \ Partíciók	Π_a	Π_b	Π_c	Π_d	Π_e	Π_f	Π_g	Π_h
(ab) : 1	×	×						
* $(acd\bar{e})$: 2	×		×	×	×			
(bf) : 3		×				×		
$(\bar{e}\bar{f}\bar{h})$: 4					×	×		×
* $(\bar{g}\bar{h})$: 5							×	×

3.139. ábra. Lefedési tábla a 3.136. ábrán megadott hálózat állapotkódolásának meghatározásához

pésére. A maximális kompatibilitási osztályokkal végezzük el a kiindulási megkülönböztető partíciók optimális lefedését. Ehhez felhasználhatjuk a prímimplikánstábla lefedésére megismert eljárás formális szabályait. A feladat ugyanis lényegében azonos, csupán a lefedő prímimplikánsok helyett a maximális kompatibilitási osztályokat, a lefedendő mintermek helyett pedig a kiindulási megkülönböztető partíciókat kell értelmeznünk. Esetünkben az így képezhető lefedési tábla a 3.139. ábrán látható. A lefedési segédfüggvény felírásához a maximális kompatibilitási osztályokat a melléírt számokkal jelöltük. A lefedési segédfüggvényt az ismert módon írhatjuk fel:

$$S = 2\cdot 5(1+2)(1+3)(2+4)(3+4)(4+5) = \ldots + 2\cdot 3\cdot 5 + \ldots$$

Eszerint tehát az $(acd\bar{e})$, a (bf) és a $(\bar{g}\bar{h})$ maximális kompatibilitási osztályok elegendőek az összes kiindulási megkülönböztető partíció lefedéséhez. Ezután az eljárás harmadik lépéseként képezhetjük az állapotkódokat.

3. Vizsgáljuk meg az optimális lefedést biztosító maximális kompatibilitási osztályokat diszjunktivitás szempontjából. Ha ugyanis egy oszlop több osztályban is szerepel, akkor azt egyetlen osztályban elegendő szerepeltetnünk, a többitől elhagyhatjuk. Az elhagyások célja ezúttal kizárólag az, hogy feleslegesen ne rögzítsünk közömbös bejegyzéseket a kódokban. A zártság ellenőrzésével természetesen nem kell foglalkoznunk, hiszen most a fogalom nem is definiálható, mert nem létezhet az oszlopok között feltételes kompatibilitás. Példánkban az optimális lefedést biztosító maximális kompatibilitási osztályok diszjunktak, így közvetlenül alkalmasak az állapotkódok felírására. A legegyszerűbben úgy járhatunk el, hogy a (3.9.22.) mátrix alapján képezzük a kapott lefedő maximális kompatibilitási osztályoknak megfelelő eredő oszlopokat:

	2	3	5
	$(y_a y_c y_d \bar{y}_e)$	$(y_b y_f)$	$(\bar{y}_g \bar{y}_h)$
	y_1	y_2	y_3
A	0	0	1
B	1	0	0
C	0	1	0
D	1	1	1

(3.9.23.)

A további egyszerűbb hivatkozás érdekében a lefedő maximális kompatibilitási osztályokat megvalósító szekunder változókat rendre y_1-gyel, y_2-vel és y_3-mal jelöltük. Az így kapott szekunder kombinációk a bemutatott eljárásból következően olyan állapotkódoknak tekinthetők, amelyek lehetetlenné teszik a kritikus versenyhelyzetből fakadó hibás állapotátmenetek létrejöttét.

Megfigyelhetjük, hogy a kapott állapotkódok nem szomszédosak, az eljárás teljesen szisztematikus, mindig eredményre vezet és megmarad az állapottábla normál jellege. A végeredményként kapott állapotkódokkal az y_1, y_2, y_3 szekunder változók a (3.9.23.) mátrixból kiolvashatóan az alábbi állapotpartíciókat valósítják meg:

$$y_1\colon\ \Pi_1 = (AC, BD); \qquad y_2\colon\ \Pi_2 = (AB, CD);$$
$$y_3\colon\ \Pi_3 = (AD, BC).$$

Ezek a partíciók az eljárásból következően a kiindulási megkülönböztető partíciók összes elkülönítő előírását tartalmazzák. Ezt úgy is fogalmazhatjuk, hogy a Π_1, Π_2, Π_3 partíciók lefedik a kiindulási megkülönböztető partíciók teljes listáját.

Példánkban a kritikusversenyhelyzet-mentes kódoláshoz több szekunder változóra van szükség, mint amennyit az állapottábla sorainak megkülönböztetése igényelne. Ha a kiindulási megkülönböztető partíciók olyan kevés elkülönítő előírást tartalmaznának, hogy kevesebb szekunder változót igényelnének a kódoláshoz mint a kiindulási állapottábla sorainak megkülönböztetése, akkor természetesen a sorok megkülönböztetésére pótlólagos szekunder változókat kell felvenniük.

Mivel a kapott állapotkódok általában nem teljesítik a szomszédossági követelményeket, a kódolt állapottábla kitöltésekor az esetleges kihasználatlan kódok soraira is ügyelnünk kell. A 3.140. ábrán láthatjuk a példánkban vizsgált hálózat kódolt állapottábláját a (3.9.23.) állapotkódok felhasználásával.

A kihasználatlan kódok sorait azért kellett megfelelő módon kitöltenünk, mert a nem szomszédos kódú állapotátmenetek során kialakulhatnak a kihasználatlan állapotkódok is. Ha az ezekhez tartozó rovatok mindegyikébe közömbös bejegyzést írtunk volna, akkor a megvalósítás során olyan rögzítések történhetnek, amelyek hibás stabil állapotba is vezethetnék a hálózatot. A kitöltés gondolatmenetének szemlélteté-

	$y_1y_2y_3$ \ x_1x_2	00	01	11	10
A	001	$y_1y_2y_3$ z (001) 0	010 –	100 0	(001) 0
B	100	111 –	111 –	(100) 0	(100) 0
C	010	111 1	(010) 1	(010) 1	(010) 1
D	111	(111) 1	(111) 1	100 –	001 –
Kihasználatlan kódok	000	—	010 –	100 0	—
	011	111 –	010 –	—	001 –
	101	111 –	111 –	100 0	001 –
	110	111 –	111 –	100 0	—

3.140. ábra. A 3.136. ábrán megadott hálózat kódolt állapottáblája megkülönböztető partíciók alapján meghatározott állapotkódok felhasználásával és a kihasználatlan kódok sorainak kitöltésével

sére vizsgáljuk meg a hálózat működését az $y_1y_2y_3=001$ stabil állapotból kiindulva, ha a bemeneti kombináció $x_1x_2=00$-ról $x_1x_2=01$-re változik.

A 3.140. ábrán folytonos nyilak jelzik az ilyenkor előírt állapotátmenetet, amely az $y_1y_2y_3=010$ stabil állapotba vezet az $x_1x_2=01$ oszlopban. A nem szomszédos kódú állapotátmenet miatt azonban a szekunder változás során — ismert okokból — átmenetileg létrejöhet az $y_1y_2y_3=000$ vagy az $y_1y_2y_3=011$ kihasználatlan kód (szaggatott nyilak). Emiatt ezen sorok $x_1x_2=01$ oszlopbeli rovatában a megvalósítás során nem szabad olyan bejegyzést rögzítenünk, amely nem az előírt $y_1y_2y_3=010$ stabil állapotba vezetné a hálózatot. Példánkban ezért írtunk ezekbe a rovatokba $Y_1Y_2Y_3=010$ bejegyzést. Megfigyelhetjük, hogy a szekunder változók különböző működési sebessége miatt így két stabil állapot között a megvalósított hálózatban több instabil állapot léphet fel annak ellenére, hogy a kiindulási kódolatlan állapottábla normál jellegét a kódolási eljárás során nem szüntettük meg. Így az egymás után következő instabil állapotokhoz tartozó kimeneti kombinációk sem választhatók meg egymástól függetlenül a hibás kimeneti impulzusok fellépésének veszélye miatt ugyanúgy, mint azt az átvezető állapotok felvételével végzett kódolás esetére bemutattuk.

A kihasználatlan kódok soraiba írt többi bejegyzés szükségessége hasonló módon követhető.

Felmerülhet a kérdés ezek után, hogy mi a lényegi különbség a most bemutatott eljárással meghatározott és az átvezetőállapotok felvételével kapott kódolási eredmény között, hiszen — annak ellenére, hogy erre nem törekedtünk — most is több szomszédos instabil állapot léphet fel két nem szomszédos kódú stabil állapot közötti átmenet során. A lényeges különbség az, hogy a megkülönböztető partíciók felhasználásával végzett állapotkódolás esetén a kiindulási állapottábla normál jellegét nem kell megváltoztatnunk. Ezáltal a nem szomszédos kódú állapotátmenetek során az összes új értéket felvevő szekunder változó értékmódosítása már az első instabil állapotban megindul. Ezzel ellentétben az átvezetőállapotok felvételén alapuló eljárás eredményeként minden instabil állapotban csak egyetlen szekunder változó értékmódosítása indul meg, vagyis két nem szomszédos kódú stabil állapot közötti átmenet szekunder változásai csak időben sorosan indulhatnak, ami lassúbb működést okoz, mintha már az első instabil állapotban megindulhattak volna. A megkülönböztető partíciókon alapuló eljárás tehát megtartja a kiindulási állapottábla normál jellegét, gyorsabb működést eredményez és a szükséges szekunder változók számát optimális lefedés útján szolgáltatja.

3.9.2.4. A bemeneti és a szekunder változások érzékelési sorrendjének hatása az állapotkódolásra

A lényeges hazárd jelenségének bemutatásakor láttuk, hogy integrált áramköri építőelemek esetén nem tételezhetjük fel, hogy az aszinkron sorrendi hálózat működése során a hálózat minden pontján előbb érvényesül a bemeneti változás (az állapottáblán oszlopváltás) és csak ezután a szekunder változások (az állapottáblán sorváltás). Előfordulhat, hogy a késleltetési viszonyok miatt a bemeneti és a szekunder változások az ok—okozati összefüggés által meghatározott időrenddel ellentétesen hatnak a hálózat egyes pontjain. Bemutattuk, hogy bizonyos felépítésű állapottáblák esetében éppen ez a jelenség okozhatja hibás stabil állapot kialakulását szomszédos kódolás mellett is (lényeges hazárd).

Az alábbiakban azt szemléltetjük, hogy nem szomszédos állapotkódok esetén a bemeneti és a szekunder változások hatásának időrendi felcserélődése a kódválasztáskor pótlólagos megfontolásokat igényel. A 3.141. ábrán adott állapottábla mellé felírtuk a megkülönböztető partíciók alapján meghatározott állapotkódokat. Az eljárás ismert lépéseit nem részletezzük, csak a kiindulási megkülönböztető partíciókat soroljuk fel:

$x_1x_2=00$	$x_1x_2=01$	$x_1x_2=11$	$x_1x_2=10$
$\Pi_a=(BC, DE)$		$\Pi_d=(B, CD)$	$\Pi_h=(AE, CD)$
$\Pi_b=(A, BC)$		$\Pi_e=(AD, B)$	
$\Pi_c=(A, DE)$		$\Pi_f=(BE, CD)$	
		$\Pi_g=(AD, BE)$	

A kódolt állapottábla a 3.142. ábrán látható.

$y_1y_2y_3$ \ y \ x_1x_2	00	01	11	10
011	A	(A) 0	(A) 0	D 0	E 0
111	B	C 0	(B) 0	(B) 0	E 0
101	C	(C) 0	(C) 0	D 0	(C) 0
000	D	E –	(D) 0	(D) 0	C 0
110	E	(E) 1	(E) 0	B 0	(E) 0

3.141. ábra. Példa a bemeneti és szekunder változások érzékelésének időrendi felcserélődése miatt szükséges pótlólagos kódolási megfontolások szemléltetéséhez nem szomszédos állapotkódok esetén

$y_1y_2y_3$ \ x_1x_2	00	01	11	10
011	$Y_1Y_2Y_3$ Z (011) 0	(011) 0	000 0	110 0
111	101 0	(111) 0	(111) 0	110 0
101	(101) 0	(101) 0	000 0	(101) 0
000	110 –	(000) 0	(000) 0	101 0
110	(110) 1	(110) 0	111 0	(110) 0

3.142. ábra. A 3.141. ábrának megfelelő kódolt állapottábla

Vizsgáljuk meg a hálózat működését a 3.141. ábrán jelölt állapotátmenet esetén, vagyis az $x_1x_2=00$ mellett fennálló A stabil állapotból kiindulva $x_1x_2=10$ fellépésének hatására. Az állapottábla által előírt A—E állapotátmenetet a 3.141. ábrán nyíllal szemléltettük. Kövessük ezt az állapotváltozást a 3.142. ábra kódolt állapottábláján. Az $x_1x_2y_1y_2y_3=00011$ stabil állapotból az $x_1x_2=10$ bemeneti kombináció hatására a hálózatnak az állapottábla szerint az $x_1x_2y_1y_2y_3=10110$ stabil állapotba kellene jutnia.

Ez az előírt állapotátmenet nem játszódik le, ha a késleltetési viszonyokról az alábbiakat tételezzük fel:

— Y_1 szekunder változó visszahatásának útjában kisebb késleltetések vannak, mint Y_3 útjában,

— az Y_2-t előállító hálózatrész később érzékeli a bemeneti változást, mint ugyanennek a bemeneti változásnak a hatására létrejött szekunder változást.

Az első feltételezés értelmében a szóban forgó állapotátmenet során kialakul az $Y_1Y_2Y_3=111$ szekunder kombináció.

A második feltételezés azt jelenti, hogy az Y_2-t előállító hálózatrészhez, az $x_1=0\rightarrow 1$ változás nem jutott még el, de a bemeneti változás által — más hálózatrészekben — előidézett szekunder változás $y_1y_2y_3=111$ már igen. Az Y_2 hálózatrész bemenetén tehát az $x_1x_2y_1y_2y_3=00111$ kombináció van, amely a kódolt állapottábla előírása szerint az $Y_2=1\rightarrow 0$ átmenet eredményezi. Ha az $x_1x_2=10$ még most sem hatásos az Y_2 hálózatrész bemenetén, akkor kialakul az $x_1x_2y_1y_2y_3=00101$ stabil állapot. Ezután már hiába érzékeli ez a hálózatrész is a bemeneti kombináció megváltozását, a hálózat az állapottábla által előírt $x_1x_2y_1y_2y_3=10110$ stabil állapot helyett, az $x_1x_2y_1y_2y_3=10101$ stabil állapotba kerül, ami hibás működést jelent.

A helytelen működést tehát az okozza, hogy a bemeneti és szekunder változók változási sebességének összemérhetősége miatt az állapotátmenetek *nem minden esetben egy oszlopban* (az adott esetben nem az $x_1x_2=10$ oszlopon belül) játszódnak le. A leírt hibás állapotátmenetet a 3.142. ábrán szaggatott nyilak szemléltetik.

Esetünkben az $x_1x_2=00$ oszlopban a B—C átmenet, az $x_1x_2=10$ oszlopban pedig az A—E átmenet együttes hatását figyelhettük meg. E két állapotátmenetet az eddigi módszerrel felírt kiindulási megkülönböztető partíciók egyike sem különítette el egymástól, vagyis az így kapott állapotkódok esetén nem biztos, hogy található olyan szekunder változó, amelynek értéke a B és C állapotokban ellentétes az A és E állapotokban felvett értékéhez képest. Ennek az a következménye, hogy a késleltetési viszonyoktól függően mind a B—C, mind az A—E állapotátmenet során létrejöhet az $Y_1Y_2Y_3=111$ szekunder kombináció, ami lényegében a bemutatott hibás működés alapvető oka.

A hibás állapotátmenet lehetőségét nyilvánvalóan megszüntethetjük, ha a kiindulási megkülönböztető partíciók közé pótlólag felvesszük azokat, amelyek az azonos oszlopbeli állapotátmeneteket és stabil állapotokat rendre elkülönítik az összes szomszédos bemeneti kombináció oszlopaiban előforduló állapotátmenetektől, ill. stabil állapotoktól.

Példánkban az alábbi pótlólagos megkülönböztető partíciók írhatók fel ilyen módon:

$x_1x_2=00-10$	$x_1x_2=00-01$	$x_1x_2=11-01$	$x_1x_2=10-11$
$\Pi_i=(AE, BC)$			$\Pi_j=(AD, BE)$
			$\Pi_k=(CD, BE)$
			$\Pi_l=(AE, CD)$

Látható, hogy csak Π_i ír elő új megkülönböztetést, hiszen $\Pi_j=\Pi_g$; $\Pi_k=\Pi_f$ és $\Pi_l=\Pi_h$. Így esetünkben csupán Π_i-t kell pótlólag felvennünk a kiindulási megkülönböztető partíciók közé és az ismert lépésekben új állapotkódokat határozhatunk meg:

	y_1	y_2	y_3
A	1	0	0
B	0	0	1
C	0	0	0
D	0	1	0
E	1	1	1

Meggyőződhetünk róla, hogy a vizsgált állapotátmenet ilyen állapotkódok esetén hibátlanul játszódik le tetszőleges késleltetési viszonyok mellett is. Az állapotkódokban ugyanis az y_1 szekunder változó értéke elkülöníti egymástól a BC és az AE állapotpárokat. Emiatt az A—E átmenet során ezúttal nem jöhet létre olyan instabil állapot, amelynek kódja azonos valamelyik $x_1x_2=00$ oszlopbeli, E-től különböző stabil állapotéval. Kialakulhat viszont az átmenet során az $y_1y_2y_3=101$ vagy az $y_1y_2y_3=110$ szekunder kombináció, amely kihasználatlan a kapott kódolásban. A kihasználatlan kódok sorainak kitöltésekor ezért az előzőekben bemutatott szempontokon túlmenően arra is ügyelnünk kell, hogy az így fellépő instabil állapotok is mindig a helyes stabil állapotba vezessék a hálózatot, mindkét érintett bemeneti kombináció oszlopában. A kimeneti impulzusok elkerülésére változatlanul törekednünk kell a kihasználatlan kódok soraiba írt kimeneti kombinációk rögzítésekor. Ez azonban Mealy-modell esetén a szomszédos oszlopok kölcsönhatása miatt ellentmondó bejegyzéseket igényelhet.

Az olvasóra bízzuk ennek szemléltetését a vizsgált példánk kódolt állapottáblája (3.143. ábra) alapján, a kihasználatlan kódok sorainak kitöltésével.

$y_1y_2y_3$ \ x_1x_2	00	01	11	10
100	$Y_1Y_2Y_3Z$ 100 0	100 0	010 0	111 0
001	000 0	001 0	001 0	111 0
000	000 0	000 0	010 0	000 0
010	111 –	010 0	010 0	000 0
111	111 1	111 0	001 0	111 0

3.143. ábra. A 3.141. ábrán szereplő állapottáblának megfelelő kódolt állapottábla a $\Pi_i=(AE, BC)$ megkülönböztető partíció figyelembevételével meghatározott állapotkódok esetén

y \ x_1x_2	00	01	11	10
A	A 1	B 0	A 0	A 0
B	C 0	B 0	B 0	D 0
C	C 0	B 0	A 0	C 0
D	C 0	D 1	B 0	D 0

3.144. ábra. Példa a bemeneti és szekunder változások érzékelésének időrendi felcserélődése miatt szükséges pótlólagos kódolási megfontolások szemléltetéséhez, nem szomszédos állapotkódok esetén

$y_1y_2y_3$ \ x_1x_2	00	01	11	10
000	$Y_1Y_2Y_3$ Z 000 1	011 0	000 0	000 0
011	101 0	011 0	011 0	110 0
101	101 0	011 0	000 0	101 0
110	101 0	110 1	011 0	110 0

3.145. ábra. A 3.144. ábrán szereplő állapottáblának megfelelő kódolt állapottábla

A kimeneti impulzusok kiküszöbölésére Moore-modell esetén természetesen nem adódnak ellentmondó követelmények a szomszédos oszlopok kölcsönhatásából. Ekkor ugyanis a kimeneti kombinációk egy sor minden rovatában azonosak.

A továbbiakban bemutatjuk, hogy a szomszédos bemeneti kombinációk oszlopainak kölcsönhatását figyelembe véve kiegészített kiindulási megkülönböztető partíciók alapján kapható állapotkódok sem biztosítják minden esetben az aszinkron sorrendi hálózat helyes működését, ha a bemeneti és szekunder változások érzékelési sorrendje felcserélődhet.

Határozzuk meg a 3.144. ábrán szereplő állapottáblához az állapotkódokat a szomszédos oszlopok kölcsönhatása miatt szükséges kiegészítő megkülönböztető partíciók figyelembevételével. Így az összes kiindulási megkülönböztető partíció az alábbi:

$x_1x_2 = 00$	$x_1x_2 = 01$	$x_1x_2 = 11$	$x_1x_2 = 10$
$\Pi_a = (A, BC)$	$\Pi_c = (AB, D)$	$\Pi_e = (AC, BD)$	$\Pi_f = (A, BD)$
$\Pi_b = (A, CD)$	$\Pi_d = (BC, D)$		$\Pi_g = (BD, C)$
$x_1x_2 = 00-01$	$x_1x_2 = 00-10$	$x_1x_2 = 01-11$	$x_1x_2 = 10-11$
$\Pi_h = (AB, CD)$			

Az ismert eljárással a következő állapotkódok adódnak:

	y_1	y_2	y_3
A	0	0	0
B	0	1	1
C	1	0	1
D	1	1	0

A kódolt állapottábla a 3.145. ábrán látható. Kövessük ezen a következő állapotátmenetet. Vizsgálatunk kezdetekor a hálózat legyen az $x_1x_2y_1y_2y_3=00000$ stabil állapotban. A bemeneti kombináció változzék $x_1x_2=00$-ról $x_1x_2=01$-re.

Tételezzük fel, hogy a késleltetési viszonyok olyanok, hogy az Y_1-et előállító hálózatrészhez előbb jut el a megváltozott y_2y_3, mint az Y_2Y_3 megváltozását kiváltó bemeneti változás, ugyanakkor az Y_2, Y_3 hálózatrészek bemenetén a megváltozott y_2, y_3 még nem hatásos. Ebben az esetben a hálózat átmenetileg az $x_1x_2y_1y_2y_3=00011$ rovatba kerül, amely Y_1-nek is $0\to1$ átmenetet ír elő. Így ebben a pillanatban mindhárom szekunder változó az 1-es érték felé tart.

A három szekunder változó 1-es értékének a hálózat bemenetére való visszajutási sorrendjét a késleltetési viszonyok határozzák meg. Így az állapottáblából nem lehet kiolvasni, hogy abban a pillanatban, amikor az $x_1x_2=01$ a hálózat minden részében hatásossá válik, akkor az $y_1y_2y_3$ szekunder kombinációnak a 000 és 111 között éppen milyen közbenső értéke érvényes. Ez azt jelenti, hogy a hálózat az $x_1x_2=01$ oszlopban levő stabil állapotok bármelyikébe eljuthat, ez pedig hibás működést okozhat. A jelenség magyarázatához célszerű bevezetni az alábbi definíciót.

Egy aszinkron sorrendi hálózat ún. *d-triót* tartalmaz, ha léteznek olyan X^i és X^j szomszédos bemeneti kombinációk és y^k, y^l, y^m állapotok, hogy

$$f_y(X^i, y^k) \Rightarrow Y^k,$$
$$f_y(X^j, y^k) \Rightarrow Y^l,$$
$$f_y(X^j, y^l) \Rightarrow Y^l,$$
$$f_y(X^i, y^l) \Rightarrow Y^m,$$
$$f_y(X^i, y^m) \Rightarrow Y^m,$$
$$f_y(X^j, y^m) \Rightarrow Y^l.$$

A d-triót a fenti jelölések alkalmazásával a 3.146. ábrán látható állapottábla-részlet szemlélteti. A 3.144. ábrán az $x_1x_2=00$ és 01 oszlopokban könnyen felismerhetjük az A, B és C állapotokból álló d-triót, amely éppen a kódolt állapottáblán vizsgált állapotátmenetből indul ki.

Láttuk, hogy a d-trión belül lejátszódó állapotátmenet során olyan szekunder kombinációk alakulhatnak ki, amelyek a d-trió állapotaitól eltérnek, és így hibás stabil állapotba vezethetik a hálózatot, ha a bemeneti és szekunder változások érzékelési időpontjai felcserélődhetnek. Ezt a nibalehetőséget az állapotkódolás során úgy küszöbölhetjük ki, hogy a kiindulási megkülönböztető partíciók közé felvesszük azokat is, amelyek elkülönítik a d-trió állapotait az érintett két szomszédos bemeneti kombináció oszlopában levő egyéb stabil állapotoktól, állapotátmenetektől, ill. eset-

y \ X		X^i	X^j
y^k		Y^k	Y^l
y^l		Y^m	Y^l
y^m		Y^m	Y^l

3.146. ábra. A d-trió szemléltetése az állapottáblán

leges más d-triók állapotaitól. Így a fenti jelöléseinkkel az alábbi felépítésű kiindulási megkülönböztető partíciók pótlólagos figyelembevétele szükséges:

$$(y^k y^l y^m, y^n); \quad (y^k y^l y^m, y^p y^q), \quad (y^k y^l y^m, y^r y^s y^t),$$

ahol y^n az érintett két oszlopban levő stabil állapot; $y^p y^q$ az érintett két oszlopban levő állapotátmenet állapotai; $y^r y^s y^t$ az érintett két oszlopban levő d-trió állapotai.

Az így adódó állapotkódokban legalább egy szekunder változó értéke megkülönbözteti a d-trió állapotait az érintett két oszlop összes egyéb állapotától, ill. állapotátmenetétől, tehát nem alakulhatnak ki a d-trión belüli állapotátmenet során a d-trión belüli állapotoktól eltérő kihasznált szekunder kombinációk. A kihasználatlan szekunder kombinációk sorainak kitöltésére és a kimeneti impulzusok elkerülhetőségére természetesen ezúttal is érvényesek a korábbi megállapításaink.

Fentiek alapján példánkban a

$$\Pi_j = (ABC, D), \qquad \text{és} \qquad \Pi_k = (BCD, A)$$

kiindulási megkülönböztető partíciókat kell pótlólagosan figyelembe vennünk. Figyeljük meg, hogy Π_k felvétele azért szükséges, mert a B, C és D állapotok is d-triót alkotnak az $x_1x_2=00$ és $x_1x_2=01$ oszlopokban, így az A stabil állapottól való elkülönítésük szintén követelmény. Ezáltal az állapotkódok a következőképpen alakulnak:

	y_1	y_2	y_3	y_4
A	0	0	0	0
B	1	0	0	1
C	0	1	0	1
D	1	1	1	1

A d-triók miatti pótlólagos megkülönböztető partíciók által előírt elkülönítések tehát további szekunder változót igényeltek.

Nem vizsgáltuk azonban eddig, hogy ezek a pótlólagos partíciók nem tesznek-e feleslegessé korábban felvett más megkülönböztető partíciókat. Pedig pl. az A, B, C d-trió állapotátmenetei alapján könnyen észrevehetjük, hogy a $\Pi_i=(ABC, D)$ partíció pótlólagos felvétele feleslegessé teszi a $\Pi_h=(AB, CD)$ partíciót.

A d-trió jellegéből következően ugyanis a Π_h elkülönítés által kiküszöbölendő hibás állapotátmenetet a Π_i elkülönítés megvalósításával eleve kiküszöböljük. A 3.144. ábra állapottábláján ezt úgy magyarázhatjuk, hogy a d-trió szerinti állapotelrendeződés következtében az AB állapotpárt elegendő a D állapottól megkülönböztetni (amit Π_i meg is tesz), hiszen ha az A—B állapotátmenet során ki is alakulna a C állapot kódja, akkor is láthatóan a d-trión belüli B állapotba juthat csak a hálózat. Így a Π_h által előírt megkülönböztetés d-trió esetén szükségtelenül szigorú, több, mint ami feltétlenül szükséges, ezért az optimális megoldás érdekében elhagyandó.

Általában is kimondhatjuk, hogy a d-triók pótlólagos megkülönböztető partíciókat igényelnek, de más partíciók elhagyását is előírhatják. Példánkból levonhatjuk azt

a következtetést, hogy a d-triók miatt azokat a megkülönböztető partíciókat kell elhagynunk, amelyek a szomszédos oszlopok kölcsönhatását veszik figyelembe és egy blokkban tartalmaznak az érintett két oszlopban levő d-trión belüli állapotátmeneteket.

Esetünkben csak $\Pi_h=(AB, CD)$ elhagyása ~~szükséges~~ lehetséges, ami az ismert eljárással, az alábbi állapotkódokhoz vezet:

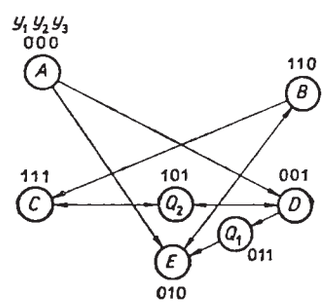
	y_1	y_2	y_3
A	0	0	0
B	0	1	1
C	0	1	0
D	1	1	1

Az így képezhető kódolt állapottáblán (3.147. ábra) megfigyelhetjük, hogy az állapotkódok kiküszöbölik az összes hibás állapotátmenet lehetőségét. A kihasználatlan kódok sorainak kitöltését az előzőekben vázolt gondolatmenettel végezhetjük. A szom-

$y_1y_2y_3$ \ x_1x_2	00	01	11	10
000	$y_1y_2y_3$ z 000 1	011 0	000 0	000 0
011	010 0	011 0	011 0	111 0
010	010 0	011 0	000 0	010 0
111	010 0	111 1	011 0	110

3.147. ábra. A 3.144. ábrán szereplő állapottáblának megfelelő kódolt állapottábla a d-trió hatásának figyelembevételével

$y_1y_2y_3$	y \ x_1x_2	00	01	11	10
000	A	A 0	A 0	D 0	E 0
110	B	C 0	B 0	B 0	E 0
111	C	C 0	C 0	Q_2 0	C 0
001	D	Q_1 *	D 0	D 0	Q_2 0
010	E	E 1	E 0	B 0	E 0
011	Q_1	E *	E 0	—	—
101	Q_2	—	D 0	D 0	C 0



3.148. ábra. A 3.141. ábrán szereplő állapottábla kódolása átvezető állapotok felvételével

szédos oszlopok kölcsönhatásának figyelembevétele miatt Mealy-modell esetén nem mindig tudjuk kiküszöbölni a kimeneti impulzusokat.

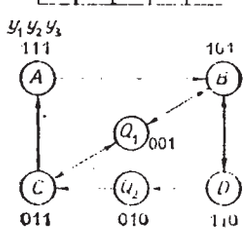
Belátható, hogy az eddig bemutatott hibás működési lehetőségek a szomszédossági feltételek szerinti állapotkódok esetén nem állnak fenn, hiszen ekkor egy állapotátmenet során csak egyetlen szekunder változó értéke változik. Így hiába cserélődik fel a hálózat egyes részein a bemeneti és a szekunder változások észlelési sorrendje, a bemeneti változás érvényesülésekor az eddig bemutatott felépítésű állapottáblákkal leírható hálózatokban végül mindig a helyes stabil állapot alakul ki. Ennek szemléltetése céljából a 3.148., ill. a 3.149. ábrán bemutattuk a 3.141., ill. a 3.144. ábrán szereplő állapottáblák kódolását átvezetőállapotok felvételével. Az átvezetőállapotok sorainak kitöltésekor természetesen nem szabad megfeledkeznünk arról, hogy a szekunder változás elsődleges érzékelése esetén még a kiindulási bemeneti kombináció oszlopában is eljuthatunk az átvezetőállapot sorába.

Ezekben a rovatokban tehát nem hagyhatjuk meg a közömbös bejegyzést, hiszen így a tervezés további lépései esetleg hibásan rögzítenék az ekkor létrejövő következő állapotot. Ezért írtunk elő például a 3.148. ábrán a Q_1 sor $x_1x_2=01$ rovatában E 0-t és a Q_2 sor $x_1x_2=01$ rovatában D 0-t.

Megfigyelhetjük, hogy a hálózat akkor is helyesen működne, ha a Q_1 sor $x_1x_2=01$ rovatába D 0-t írtunk volna. A 3.144. ábrán szereplő állapottábla instabil állapotok módosítása révén végrehajtható szomszédos kódolását a 3.150. ábrán mutatjuk be. A helyes működésről ezúttal is könnyen meggyőződhetünk az állapottáblán formálisan követett változásokkal, a bemeneti és szekunder változások érzékelési sorrendjének felcserélődését is feltételezve.

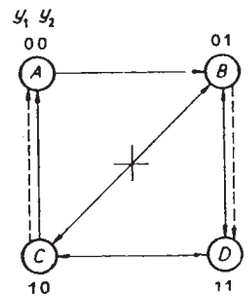
A továbbiakban bemutatjuk, hogy vannak olyan aszinkron sorrendi hálózatok, amelyek esetében a bemeneti és szekunder változások érzékelési sorrendjének felcserélődése által okozott hibás működési lehetőséget *semmiféle kódválasztással nem*

$y_1y_2y_3$	$y \backslash x_1x_2$	00	01	11	10
111	A	(A)1	B 0	(A)0	(A)0
101	B	Q_1 0	(B)0	(B)0	D 0
011	C	(C)0	Q_1 0	A 0	(C)0
110	D	Q_2 0	(D)0	B 0	(D)0
001	Q_1	C 0	B 0	–	–
010	Q_2	C 0	D 0	–	–



3.149. ábra. A 3.144. ábrán szereplő állapottábla kódolása átvezető állapotok felvételével

y_1y_2	$y \backslash x_1x_2$	00	01	11	10
00	A	(A)1	B 0	(A)0	(A)0
01	B	C 0	(B)0	(B)0	D 0
11	C	(C)0	A 0	A 0	(C)0
10	D	C 0	(D)1	B 0	(D)0



3.150. ábra. A 3.144. ábrán szereplő állapottábla kódolása instabil állapotok módosításával

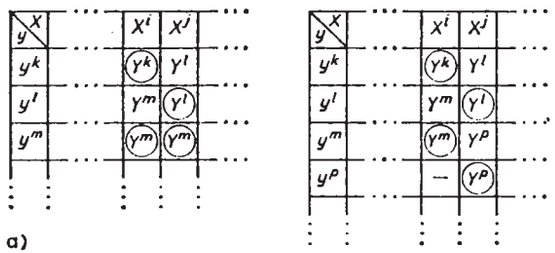
lehet kiküszöbölni. Az ilyen hálózatok állapottábláin tipikus állapotelrendeződések figyelhetők meg, ami természetesen magából a megoldandó logikai feladatból következik. A 3.5.6. pontban bemutatott példa is ilyen volt és a jelenséget lényeges hazárdnak neveztük.

3.9.2.5. A lényeges hazárd vizsgálata

Tételezzük fel, hogy egy normál aszinkron sorrendi hálózat állapottábláján a 3.151. ábrán vázolt állapotelrendeződések valamelyike figyelhető meg két szomszédos bemeneti kombináció oszlopában. Ismert jelöléseinket alkalmazva a 3.151. ábra alapján:

$$f_y(X^i, y^k) \Rightarrow Y^k,$$
$$f_y(X^j, y^k) \Rightarrow Y^l,$$
$$f_y(X^j, y^l) \Rightarrow Y^l,$$
$$f_y(X^i, y^l) \Rightarrow Y^m,$$
$$f_y(X^i, y^m) \Rightarrow Y^m,$$
$$f_y(X^j, y^m) \Rightarrow \begin{cases} Y^m, & \text{3.151}a\text{) ábra alapján,} \\ Y^p, & \text{3.151}b\text{) ábra alapján.} \end{cases}$$

Hasonlítsuk össze ezeket az állapotelrendeződéseket a d-trióval (3.146. ábra). Megfigyelhetjük, hogy a másodszori bemeneti változás hatására a d-trióval ellentétben nem kerül vissza a hálózat az y^l stabil állapotba. Éppen ebből a tényből követ-



a)

b)

3.151. ábra. Tipikus állapotelrendeződések lényeges hazárd esetén

$y \backslash x_1x_2$	00	01	11	10
A	A 0	A 0	A 0	B 0
B	A 0	—	C –	B 0
C	D 1	C 1	C 1	C 1
D	D 1	A –	—	C 1

3.152. ábra. Példa lényeges hazárdot okozó állapotelrendeződésre

kezik, hogy állapotkódolással nem tudjuk kiküszöbölni a hibás működés lehetőségét a bemeneti és szekunder változások érzékelési időrendjének felcserélődése esetén, vagyis a hálózatban lényeges hazárd van. Ennek szemléltetése céljából vizsgáljuk meg újból a 3.5.6. pontban megoldott feladatnak a 3.71. ábrán szereplő állapottábláját, amelyet a 3.152. ábrán újból felrajzoltunk, folytonos nyíllal bejelölve rajta a 3.5.6. pontban követett állapotátmenetet. Láthatjuk, hogy az $x_1x_2=11$ és az $x_1x_2=10$ oszlopokban az *A*, *B* és *C* állapotok elrendeződése a 3.151*a)* ábrának felel meg. Ha a hálózat egyes pontjain előbb jut érvényre a szekunder változás, mint az azt előidéző bemeneti változás, akkor előfordulhat, hogy még $x_1x_2=11$ mellett hat a *B* állapot szekunder kombinációja, ami a *C* állapot kialakulását eredményezheti még $x_1x_2=10$ észlelése előtt. Így a szaggatott nyíllal jelölt állapotváltozások játszódhatnak le, aminek következtében hibásan a *C* stabil állapot alakul ki az $x_1x_2=10$ oszlopban. A jelenség nyilvánvalóan teljesen független az állapotkódoktól, ezért kódválasztással a hibalehetőséget megszüntetni nem lehet. A megoldás egyedüli módja, hogy a 3.5.6. pontban már említett módon helyreállítjuk a hálózat minden részében az ok—okozati összefüggés érvényesülését, azaz szekunder változók visszahatásának útjába megfelelő értékű késleltetéseket építünk be. Elvileg nem mindig szükséges az összes szekunder változó visszahatását késleltetni, csupán azokat, amelyek valamely lényeges hazárdot okozó elrendeződés (3.151. ábra) kiindulási állapotátmenetében változnak. Ez természetesen már függ a választott állapotkódoktól. A 3.5.6. pontban érvényes állapotkódok esetén ezért kellett csupán Y_2 visszahatását késleltetni. A gyakorlatban azonban gyakran az összes visszacsatoló ágba elhelyezik a késleltetést, ha az állapottáblán csupán egyetlen olyan állapotelrendeződés is kimutatható, amely lényeges hazárdot okoz. Így ugyanis már nem kell többé foglalkozni azokkal az előzőekben bemutatott hibalehetőségekkel, amelyeket nem a lényeges hazárd okoz ugyan, de a

bemeneti és szekunder változó észlelési sorrendjének felcserélődésére vezethetők vissza.

Fentiek alapján az állapottáblát úgy vizsgálhatjuk formálisan lényeges hazárd szempontjából, hogy rendre minden egyes stabil állapotból kiindulva az összes szomszédos bemeneti változás hatására ellenőrizzük, nem mutatható-e ki a 3.151. ábrán vázolt állapotelrendeződések valamelyike (Unger-tétel). Ha nem, akkor kódválasztással mindig biztosíthatjuk a helyes állapotátmeneteket. Ha igen, akkor nem kerülhetjük el késleltetések beépítését a visszacsatoló ágakba. Ha az állapottábla nem normál, akkor természetesen nem alkalmazhatjuk a fenti formális vizsgálatot, hiszen a hálózat egyes részei az instabil állapotváltozások megindulása után bármelyik instabil állapotban észlelhetik a bemeneti változást. Így nem normál hálózatok esetén a tipikus állapotelrendeződések keresésén túlmenően az instabil állapotokból kiinduló változásokat is vizsgálnunk kell a bemeneti és szekunder változások észlelési sorrendjének felcserélődéséből fakadó hibalehetőségek felderítéséhez.

3.9.2.6. Összefoglaló következtetések az aszinkron sorrendi hálózatok állapotkódolásával kapcsolatban

A 3.9.2.1...3.9.2.5. pontok alapján megállapíthatjuk, hogy a hibás állapotátmenetek és a hibás kimeneti impulzusok kiküszöbölésére a bemutatott kódolási módszerek az alábbi lehetőségeket nyújtják.

1. A szomszédossági feltételek alapján választott állapotkódokkal (instabil állapotok módosítása, átvezetőállapotok felvétele):

— Kiküszöbölhető a kritikus versenyhelyzet.

— Moore-modell esetén elkerülhetők az instabil állapotok fellépésekor a hibás kimeneti impulzusok. (Mealy-modell esetén nem, hiszen az $f_z(X, y) \Rightarrow Z$ kimeneti leképezés argumentumában nem szomszédos változások léphetnek fel szomszédos y változások mellett is, ami a kimeneti függvény megvalósításakor funkcionális hazárdot okoz.)

— Lényeges hazárdot nem tartalmazó hálózatok esetén megszüntethetők a bemeneti és szekunder változások érzékelési sorrendjének felcserélődéséből származó hibalehetőségek.

— Megszűnik a hálózat normál jellege.

2. A megkülönböztető partíciók alapján választott állapotkódokkal:

— Kiküszöbölhető a kritikus versenyhelyzet.

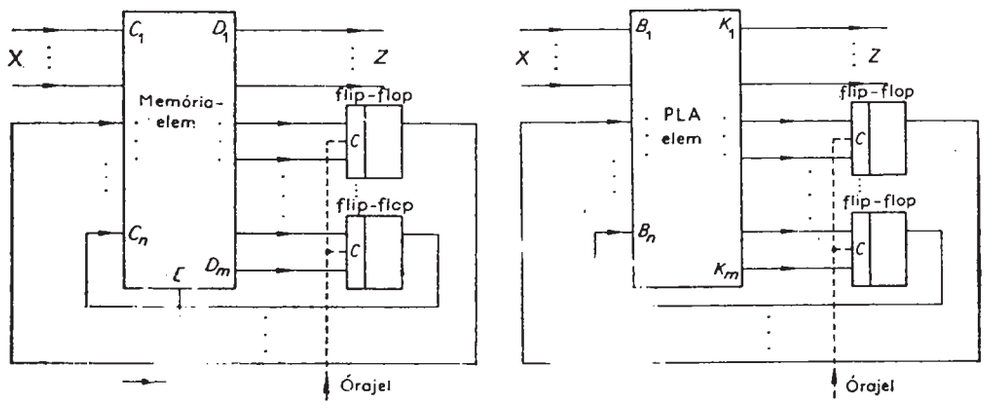
— Moore-modell esetén sem kerülhetők el mindig az instabil állapotok fellépésekor a hibás kimeneti impulzusok, mivel az állapotkódok nem szomszédosak, így az $f'_z(y) \Rightarrow Z$ leképezés megvalósításakor funkcionális hazárd keletkezhet. Ez természetesen még akkor is így van, ha a kihasználatlan kombinációk sorainak kitöltésekor a kimeneti kombinációk értékeit ellentmondás-mentesen tudjuk rögzíteni.

— Lényeges házárdot nem tartalmazó hálózatok esetén megszüntethetők a bemeneti és szekunder változások érzékelési sorrendjének felcserélődéséből származó hibás állapotátmenetek, de a kihasználatlan kombinációk sorainak kitöltésekor a kimeneti kombinációk értékeire ellentmondásos rögzítési feltételek adódhatnak, ami hibás kimeneti impulzusokat okozhat átmeneti állapotban.
— Megmarad a hálózat normál jellege.

Fentiekből következik, hogy a hibás kimeneti impulzusokat általános esetben csak Moore-modell tervezésével és szomszédos kódolással lehet elkerülni. Ilyenkor azonban megszűnik a hálózat normál jellege, ami lényegében lassítja a működést. Ha a hálózat állapottábláján kimutatható a lényeges hazárdot okozó állapotelrendeződés, akkor a megfelelő visszacsatoló ágakba késleltetéseket kell beépíteni. A késleltetéseket legegyszerűbben páros számú inverter sorba kapcsolásával, vagy két inverter közé kapcsolt *RC* taggal valósíthatjuk meg.

3.10. Sorrendi hálózatok megvalósítása memória- és PLA elemekkel

A 3.67. ábrán látható általános felépítési vázlaton láttuk, hogy a sorrendi hálózatok flip-flopok alkalmazása esetén több kimenetű kombinációs hálózatból és flip-flopokból állnak. (Az aszinkron hálózatok flip-flopok nélkül, visszacsatolt kombinációs hálózattal is előállíthatók.) A 2.6. és 2.7. fejezetekben megismertük, hogy miként lehet a kombinációs hálózatokat memória- és PLA elemekből felépíteni. Ezeket az elemeket tehát sorrendi hálózatok megvalósításához is felhasználhatjuk, ha a 3.67. ábra több kimenetű kombinációs hálózatát építjük meg segítségükkel az ismert módon a 3.166. ábra szerint. Az ábrán egyetlen memória- és PLA elem szerepel, de természetesen helyükre képzelhetjük a 2.6. és 2.7. fejezetekben vázolt módokon kialakuló több elemes hálózatokat is. Az órajelbemeneteket szaggatottan jelöltük, utalva ezzel arra, hogy flip-flopok alkalmazása esetén, akár szinkron, akár aszinkron hálózatok felépítésének szemléltetésére alkalmas az ábra, hiszen a 3.67. ábrán az órajel elhagyásával aszinkron hálózatot is értelmezhetünk. Memóriaelem alkalmazásakor azonban nem szabad megfeledkeznünk arról, hogy a címváltozás hatására létrejövő adatváltozás csak a memóriaelemre jellemző ciklusidő múlva észlelhető biztonságosan. A ciklusidő letelte előtti átmeneti adatértékekre a katalógusok általában nem tartalmaznak előírásokat, így definiálatlanoknak kell tekintenünk a flip-flopokra jutó vezérlőjelek átmeneti értékeit. Emiatt aszinkron hálózatok megvalósítása esetén a flip-flopok az előre nem ismert átmeneti vezérlés hatására hibás stabil állapotot hozhatnak létre. Ezért aszinkron hálózatokat csak akkor tanácsos memóriaelem segítségével megépíteni, ha mérések útján pontosan meg tudjuk ismerni az adott memóriaelem ciklusidőn belüli átmeneti viselkedését.



3.166. ábra. [illegible]rendi hálózatok felépítésének ál[illegible]s vázlata memória- il[illegible]LA elemek alkalmazása esetén

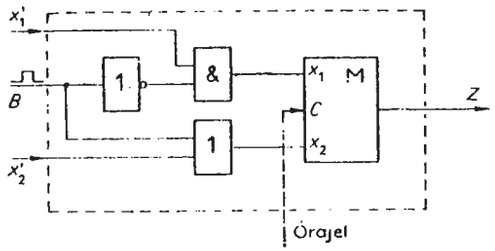
PLA elem alkalmazásakor a fenti nehézség nem merül fel, hiszen az tulajdonképpen kétszintű kombinációs hálózatot tartalmaz. Így a tervezéskor az összes megismert állapotkódolási és hazárdmentesítési szempontot figyelembe tudjuk venni akár szinkron, akár aszinkron esetben.

Tudjuk, hogy az aszinkron sorrendi hálózatok megépíthetők visszacsatolt kombinációs hálózatként is, tehát flip-flopok alkalmazása nélkül. A memóriaelem közvetlen visszacsatolása esetén nyilvánvalóan jelentkeznek a fent említett átmeneti adatértékek káros hatásai a címbemeneteken, ami szintén lehetetlenné teszi az aszinkron hálózatok esetén szükséges kódolási és hazárdmentesítési előírások betartását. PLA elemet alkalmazva ilyen nehézségek nem lépnek fel, de a közvetlen visszacsatolás csak akkor végezhető el, ha ez az adott PLA típusra áramkörileg megengedett.

Léteznek olyan nagy integráltsági fokú programozható építőelemek, amelyek a PLA elemen kívül flip-flopokat is tartalmaznak, lényegében a 3.166. ábra elrendezése szerint. Az ilyen elemeket PLS (Programmable Logic Sequencer) elemeknek nevezik és legtöbbször a kimeneti kombináció tárolására alkalmas flip-flopokat is tartalmaznak.

3.11. Sorrendi hálózatok kiindulási állapotának biztosítása

Az eddigiekben nem foglalkoztunk azzal a fontos követelménnyel, hogy a sorrendi hálózatok bekapcsolásakor (a tápfeszültség megjelenésekor) valamilyen előírt, ún. *kiindulási (vagy alap-)* állapotba kerüljenek. Véletlenszerű ugyanis, hogy a sorrendi hálózatok a bekapcsoláskor lejátszódó tranziens áramköri jelenségek után melyik állapotba kerülnek. A helyes működéshez természetesen feltétlenül szükséges,



3.167. ábra. A kiindulási állapotot beállító bemeneti kombináció előállításának szemléltetése

hogy a hálózat a logikai feladatban előírt alapállapotból induljon. Ezért bekapcsolás után mindig elő kell állítani egy impulzust, amelynek hatására minden sorrendi hálózat előírt alapállapotába jut. Ha egy sorrendi hálózat állapottáblája olyan, hogy egy adott bemeneti kombináció hatására csak az alapállapot alakulhat ki, akkor a bekapcsolás után ennek a bemeneti kombinációnak a létrehozása elegendő az előírt kiindulási állapot biztosítása céljából. A 3.167. ábrán B-vel jelöltük az alapállapotba hozó impulzust, és az ábrázolt sorrendi hálózatról feltételeztük, hogy az $x_1x_2=01$ bemeneti kombináció fellépése esetén mindig létrejön az előírt kiindulási állapot. Az ábrán szaggatottan bekeretezett sorrendi hálózat $B=0$ esetén x_1' és x_2' bemeneteket tekintve M-mel azonos működésű. $B=1$ esetén viszont (az impulzus időtartama alatt) az x_1' és x_2' bemenetekre jutó jelértékektől függetlenül az az $x_1x_2=01$ bemeneti kombináció jut M-re, ami feltételezésünk szerint az előírt kiindulási állapotot biztosítja.

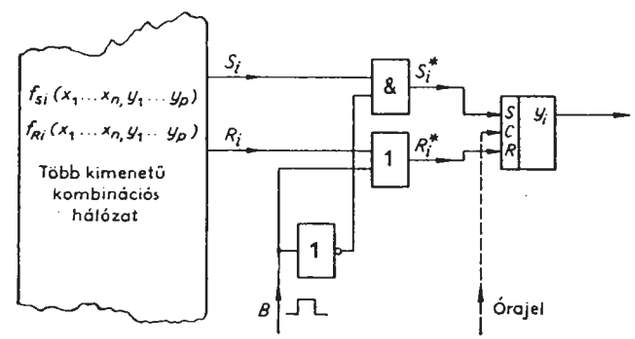
Természetesen nem minden sorrendi hálózatra adható meg olyan bemeneti kombináció, amelynek hatására minden esetben az előírt alapállapot jön létre. Elegendő olyan állapottáblára gondolnunk, amelynek minden egyes oszlopában egynél több állapot fordul elő. Ilyenkor a kiindulási állapothoz tartozó bemeneti kombináción kívül az alapállapotba hozó impulzussal közvetlenül be kell állítanunk az egyes szekunder változók értékeit a kiindulási állapot kódjainak megfelelően.

Ezt egyszerűen elvégezhetjük, ha a tervezés során megvalósított f_y leképezést az alapállapot-beállító impulzussal a kívánt szekunder változó értékétől függően módosítjuk. A 3.168. ábrán látható táblázatban összefoglaltuk a módosítás különböző eseteit visszacsatolt kombinációs hálózatra és a megismert flip-flop típusokra vonatkozóan. A táblázat alapján 3.169. ábrán egy olyan hálózatrészt rajzoltunk fel, amelyben egy S—R flip-flop $y=0$ alapállapotba állítása követhető. Szinkron flip-flop esetén természetesen a B impulzus időtartamát úgy kell megválasztanunk, hogy tartama alatt legalább egy óraimpulzus beérkezzen.

Mint említettük, a legtöbb integrált áramköri flip-flop rendelkezik Preset és Clear elnevezésű bemenetekkel, amelyek előnyösen használhatók az alaphelyzetbe állítás céljára, hiszen segítségükkel a többi bemeneti jeltől és az órajeltől teljesen függetlenül beállítható a flip-flop állapota. Így ilyenkor nincs szükség az f_y leképezés fenti módosításaira.

f_y leképzés megvalósításának módja	A tervezés során kapott függvények	Az alapállapot-beállító impulzussal (B) módosított függvények	
		ha az előírt alapállapotban $Y=0$	ha az előírt alapállapotban $Y=1$
Visszacsatolt kombinációs hálózat	$Y=f(x_1\dots x_n, y_1\dots y_p)$	$Y^*=Y\bar{B}$	$Y^*=Y+B$
S-R flip-flop	$S=f_S(x_1\dots x_n, y_1\dots y_p)$ $R=f_R(x_1\dots x_n, y_1\dots y_p)$	$S^*=S\bar{B}$ $R^*=R+B$	$S^*=S+B$ $R^*=R\bar{B}$
J-K flip-flop	$J=f_J(x_1\dots x_n, y_1\dots y_p)$ $K=f_K(x_1\dots x_n, y_1\dots y_p)$	$J^*=J\bar{B}$ $K^*=K+B$	$J^*=J+B$ $K^*=K\bar{B}$
T flip-flop	$T=f_T(x_1\dots x_n, y_1\dots y_p)$	$T^*=T\bar{B}+yB$	$T^*=T\bar{B}+\bar{y}B$
D-G flip-flop	$D=f_D(x_1\dots x_n, y_1\dots y_p)$ $G=f_G(x_1\dots x_n, y_1\dots y_p)$	$D^*=D\bar{B}$ $G^*=G+B$	$D^*=D+B$ $G^*=G+B$
D flip-flop	$D=f_d(x_1\dots x_n, y_1\dots y_p)$	$D^*=D\bar{B}$	$D^*=D+B$

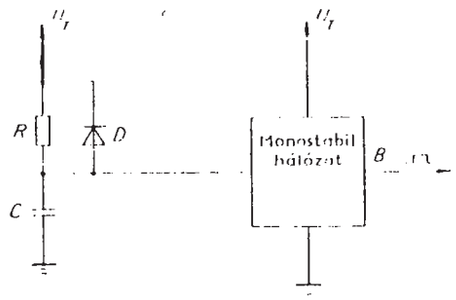
3.168. ábra. A szekunder változók alapállapotnak megfelelő közvetlen beállításához szükséges módosított vezérlő függvények összefoglalása



3.169. ábra. S—R flip-floppal megvalósított szekunder változó $y=0$ alapállapotba állításának szemléltetése

A megfelelő kiindulási állapotot természetesen úgy is biztosíthatjuk a sorrendi hálózatokban, hogy az előbbiekben említett B impulzust már az előzetes állapottábla felírásakor pótlólagos bemeneti jelnek tekintjük. Így az alapállapot létrehozását a megoldandó logikai feladat részeként értelmezzük, miáltal az állapottábla oszlopainak száma kétszeres lesz. Az így megnövekedett állapottábla kitöltésekor a $B=1$-t tartalmazó bemeneti kombinációknak megfelelő oszlopokra a kívánt alapállapotot kell beírnunk következő állapotként.

Elképzelhető, hogy egy állapottáblához — felépítéséből következően — meghatározható egy olyan bemeneti kombinációsorozat, amely bármely állapotból kiindulva a kívánt alapállapotot hozza létre. Ilyen állapottábla esetén az alapállapotot úgy is létrehozhatjuk, hogy az előbb említett bemeneti kombinációsorozatot minden bekapcsoláskor előállítjuk.



3.170. ábra. Az alapállapotot beállító impulzus előállításának egy változata

Az alapállapotot beállító impulzust létrehozhatjuk egy erre a célra fenntartott nyomógomb megnyomásával. Ilyenkor a tápfeszültség bekapcsolása után mindig meg kell nyomni ezt a nyomógombot és csak ezután tekinthető működőképesnek egy megépített hálózat. Könnyen építhetünk olyan áramkört, amely az alapállapot-beállító impulzust a tápfeszültség megjelenésekor minden esetben létrehozza, és így a nyomógomb alkalmazását feleslegessé teszi. A 3.170. ábrán egy lehetséges áramköri megoldást láthatunk erre. A megfelelően méretezett *RC* tag biztosítja, hogy az ún. monostabil hálózat bemenetére késleltetve jusson az U_T tápfeszültség megjelenése által előállított logikai 1 érték. A monostabil hálózat csak azután állítja elő a *B* impulzust, miután a tápfeszültséget a logikai hálózat minden része érzékeli. Az egyes sorrendi hálózatok tehát már működőképesek, amikor megkapják a monostabil hálózat által megfelelő szélességűre előállított *B* impulzust, amely alapállapotukat az említett módon beállítja. Az *R* ellenállással párhuzamosan kapcsolt dióda a tápfeszültség eltűnésekor a *C* kondenzátor gyors kisülését teszi lehetővé, miáltal az impulzust előállító hálózat viszonylag gyorsan válik újra működőképessé.

4. Vezérlőegységek tervezése

Az 1.4., 1.5. pontokban az 1.4. ábra alapján értelmeztük a logikai rendszerek felépítését. Az eddigiekben megismerkedtünk a logikai hálózatok tervezési módszereivel, amelyek segítségével létrehozhatjuk a funkcionális egységen belül szükséges „látszólagos" építőelemeket. Ha az építőelem-készlet fejlettségi szintje, ill. a megoldandó logikai feladat nem igényli ezeknek a látszólagos építőelemeknek az alkalmazását, mert a funkcionális egységek mindegyike egyetlen építőelemmel vagy néhány építőelem összekapcsolásával előállítható, akkor a logikai rendszerben csak a vezérlőegység megvalósításához szükségesek tervezési eljárások. Ekkor ugyanis feltételezzük, hogy a többi funkcionális egység felépítése az építőelemek ismeretében adottnak tekinthető. Természetesen a gyakorlat és a találékonyság fontos az így kialakított funkcionális egységek egyszerűsége szempontjából, mert az építőelemek kiválasztása és azok összekapcsolásának módja jelentős mértékben befolyásolja a kialakuló egységek egyszerűségét.

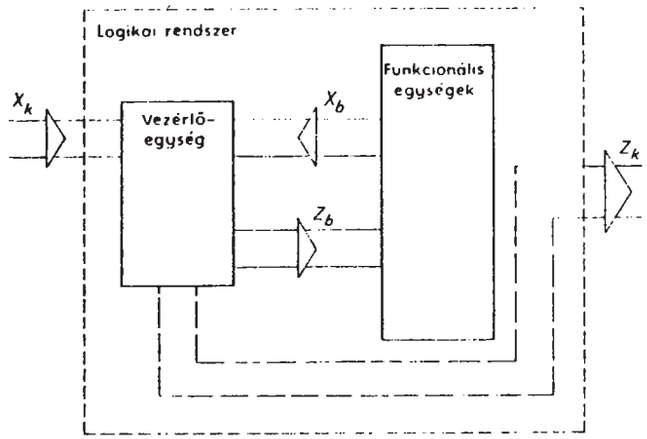
Az 1.4. ábra értelmezése szerint a vezérlőegységről is feltételeztük, hogy a funkcionális egységekre bemutatott háromfajta lehetséges felépítési mód valamelyike jellemző rá. A vezérlőegységek különleges szerepéből fakadó logikai feladat jellege azonban lehetővé teszi, hogy tervezésükre eljárásokat fogalmazzunk meg, amelyek természetesen figyelembe veszik az építőelem-készlet fejlettségét, a működési sebességre vonatkozó követelményeket és egyéb peremfeltételeket.

Vizsgáljuk meg kissé részletesebben, hogy milyen jellegű a vezérlőegység által megoldandó logikai feladat. A vezérlőegységnek össze kell hangolnia, ütemeznie kell a többi funkcionális egység működését abból a célból, hogy a logikai rendszer a környezetéből érkező bemeneti kombinációk hatására mindig azt a kimeneti kombinációt állítsa elő a környezet számára, amelyet a logikai rendszerre megfogalmazható logikai feladat előír. A 4.1. ábrán

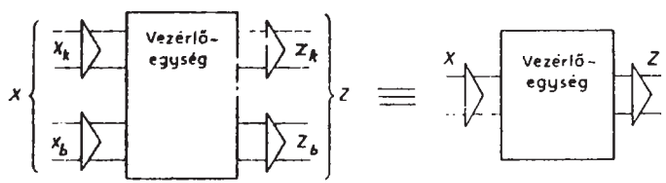
X_k a környezetből érkező bemeneti jelek összessége;
X_b a funkcionális egységekről érkező bemeneti jelek összessége;
Z_k a környezetbe jutó kimeneti jelek összessége;
Z_b a funkcionális egységeket vezérlő jelek összessége.

Nyilvánvaló, hogy az X_b jelekre általában szükség van, hiszen a funkcionális egységek megfelelő vezérléséhez rendszerint ismerni kell azok mindenkori állapotát.

Az X_k és Z_k jelek útját a logikai rendszeren belül azért jelöltük szaggatottan, mert a gyakorlatban kiadódhat, hogy a Z_k jelek egy részét a funkcionális egységekről köz-



4.1. ábra. A vezérlőegység bemeneti és kimeneti jeleinek szemléltetése



4.2. ábra. A vezérlőegység bemeneti és kimeneti jelei egységes jelölésének szemléltetése

vetlenül kivezethetjük, valamint az is előfordulhat, hogy az X_k jelek egy részére a vezérlőegységnek nincs közvetlenül szüksége, így azok közvetlenül a funkcionális egységekre vezethetők. Általános vizsgálatunkban az ilyen eseteket figyelmen kívül hagyhatjuk, hiszen közvetve (X_b és Z_b belső jelek közvetítésével) minden környezeti jel kapcsolatban van a vezérlőegységgel. Ilyen feltételezéssel a továbbiakban a vezérlőegység bemeneti és kimeneti jeleit a szokásos módon a 4.2. ábra szerint értelmezzük, ahol X és Z mindenfajta bemeneti, ill. kimeneti jelet magában foglal.

A vezérlőegységnek sorrendi hálózatnak kell lennie, mert a funkcionális egységek működésének összehangolása, ütemezése olyan logikai feladat, amely nem oldható meg kombinációs hálózattal. Így elvileg alkalmazhatnánk a sorrendi hálózatok tervezésére megismert eljárásokat, a gyakorlatban azonban már viszonylag egyszerű vezérlőegységek esetén is nehézségekbe ütköznénk. A legnagyobb gondot az jelentené, hogy a vezérlőegységekre a fenti szemléltetésből következően is általában sok bemeneti jel jut. A sorrendi hálózatok működésének leírására megismert állapottábla és állapotgráf viszont gyakorlatilag kezelhetetlenné válik bizonyos számú bemeneti jelen felül még akkor is, ha számítógépes támogatással tervezünk. Elegendő arra gondolnunk, hogy az állapottábla oszlopainak száma exponenciálisan növekszik a bemeneti jelek számának függvényében. Így a vezérlőegységek működésének leírására más módszert kell választanunk.

Ebben a fejezetben először bemutatjuk a vezérlőegységek működésleírásának egy módszerét, amely a továbbiakban ismertetendő tervezési eljárások kiindulási alapja. Ezek közül a szinkron fázisregiszteres megoldás lényegében olyan előre rögzített felépítésű szinkron sorrendi hálózatot eredményez, amelyben a kombinációs részre vonatkozó logikai függvényeket az állapottábla megszerkesztése nélkül felírhatjuk. Az aszinkron fázisregiszteres megvalósításnak pedig ezen kívül az a legfontosabb jellemzője, hogy integrált áramköri szinkron flip-flopot használunk aszinkron részhálózatként. A fázisregiszteres megoldások felépítésekor különös figyelmet fordítunk a jelkésleltető hatások által okozott hibák elkerülésére.

A fejezet további részében a vezérlőegység mikroprogramozott felépítéséből kiindulva jutunk el a mikroprocesszorok lényegének egyfajta megközelítéséig.

A vezérlőegységek szisztematikus tervezésére a szakirodalomban nem találhatók olyan általánosan ismert eljárások, mint az állapottáblával könnyen megadható sorrendi hálózatok tervezésére megismert módszerek. A jelen fejezetben bemutatott módszerek lényegét a Budapesti Műszaki Egyetem Folyamatszabályozási Tanszékének egy kutatócsoportja dolgozta ki. Ezen belül a szinkron fázisregiszteres eljárás dr. Kalmár Péter [20], az aszinkron fázisregiszteres eljárás pedig dr. Terplán Sándor [21] kutató munkájának eredménye.

4.1. A vezérlőegységek működésének leírása folyamatábrával

A sorrendi hálózatok működésének leírásakor az

$$f_z(X, y) \Rightarrow Z, \qquad \text{és} \qquad f_y(X, y) \Rightarrow Y$$

leképezések ismeretét feltételezve töltöttük ki az állapottáblát és ennek alapján megszerkesztettük az állapotgráfot. A leképezések alapján formálisan képzett állapottáblából és állapotgráfból csak akkor tudtuk egyértelműen kiolvasni a működést, ha azt is megadtuk, hogy a sorrendi hálózat aszinkron vagy szinkron működésű-e.

A vezérlőegységek működésére általában az jellemző, hogy a sok bemeneti jelük miatt képezhető igen nagyszámú lehetséges, egymástól különböző bemeneti kombinációsorozat közül csak viszonylag kevéshez tartozik előírt kimeneti, ill. szekunder kombinációsorozat. Nevezzük az ilyen bemeneti kimeneti, ill. szekunder kombinációsorozatokat *specifikációs* bemeneti, kimeneti, ill. szekunder sorozatoknak. Ezek száma néha még a sok bemeneti jelből képezhető bemeneti kombinációk számánál is kisebb.

Tételezzük fel, hogy a vezérlőegység előírt működése kiindulásképpen az összetartozó specifikációs bemeneti és kimeneti kombinációsorozatokkal adott. Nem ismertek tehát előzetesen az állapotok. Csupán a kiindulási állapotot értelmezzük az által, hogy minden előírt bemeneti kombinációsorozat első kombinációját kiindulási

bemeneti kombinációnak tekintjük és ehhez természetesen a vezérlőegység ún. kiindulási állapota tartozik.

A specifikációs bemeneti kombinációsorozatokat az alábbi két csoportba sorolhatjuk:

1. *Visszatérő bemeneti kombinációsorozatoknak* nevezzük azokat, amelyek lejátszódása után a vezérlőegység a kiindulási állapotába tér vissza.

2. *Ciklusos bemeneti kombinációsorozatoknak* nevezzük azokat, amelyek során a vezérlőegység nem tér vissza kiindulási állapotába, hanem a sorozat egy része egy adott kombinációjától kezdődően ciklikusan ismétlődik anélkül, hogy az egyes ismétlődések során létrejövő kimeneti kombinációsorozat eltérő volna.

A fenti definíciók alapján a visszatérő sorozat a ciklusos sorozatok olyan speciális esetének tekinthető, amelyben a teljes sorozat ismétlődik ciklikusan.

Azokat a visszatérő sorozatokat, amelyek során csak egyszer történik visszatérés a kiindulási állapotba, *egyszeres visszatérő bemeneti kombinációsorozatoknak* nevezzük. Ha a ciklusos sorozatok ismétlődő részeit csak egyszer írjuk le, akkor formálisan ún. *egyszeres ciklusos bemeneti kombinációssorozatokat* értelmezhetünk.

Egy vezérlőegység működését egyértelműen megadhatjuk azáltal, hogy az összes egyszeres visszatérő és egyszeres ciklusos specifikációs bemeneti kombinációsorozathoz megadjuk a specifikációs kimeneti kombinációsorozatokat.

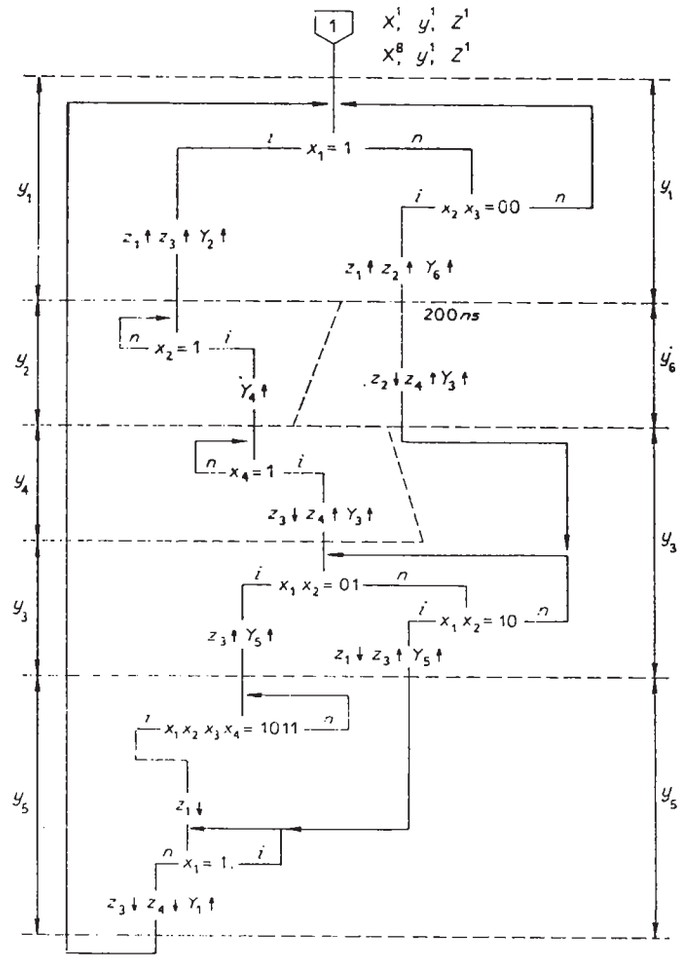
Az összes lehetséges specifikációs bemeneti kombinációsorozat ugyanis egyértelműen összeállítható az egyszeres visszatérő és egyszeres ciklusos sorozatok kizárólagos felhasználásával, amelyekhez külön-külön adottak a specifikációs kimeneti kombinációsorozatok. Így az összes lehetséges specifikációs bemeneti kombinációsorozat mindegyikéhez is adott a specifikációs kimeneti kombinációsorozat.

A vezérlőegységek működésének leírásához célszerű az f_z és f_y leképezésekből olyan leképezéseket előállítani, amelyek a specifikációs kombinációsorozatokban egymást közvetlenül követő kombinációpárok halmazai közötti kapcsolatokat írják le:

$$\varphi_z(\Delta X, y) \Rightarrow \Delta Z, \quad \varphi_y(\Delta X, y) \Rightarrow \Delta Y,$$

ahol ΔX a specifikációs bemeneti kombinációsorozatokban egymást közvetlenül követő kombinációpárok — a továbbiakban *specifikációs bemeneti változások* — halmaza, ΔZ specifikációs kimeneti kombinációsorozatokban egymást közvetlenül követő kombinációpárok — a továbbiakban *specifikációs kimeneti változások* — halmaza, ΔY a specifikációs szekunder kombinációsorozatokban egymást közvetlenül követő kombinációpárok — a továbbiakban *specifikációs szekunder változások* — halmaza.

A továbbiakban feltételezzük, hogy a vezérlőegységnek csak a specifikációs bemeneti változások hatására kell előírt módon működnie. Ezt a feltételezést úgy is értelmezhetjük, hogy a vezérlőegység bemenetére csak a specifikációs változások jutnak a megadott alapállapothoz tartozó valamelyik kiindulási bemeneti kombinációt követően.



4.4. ábra. A vezérlőegység működésének leírása folyamatábrával, a 4.3. ábrán szereplő specifikációs változásokat jelváltozásokkal jellemezve

A szimbólumból kiinduló *i* jelű ág (igen) $x_1 = 1$ esetén, az *n* jelű (nem) pedig $x_1 = 0$ esetén mutatja a vezérlési folyamat folytatódását. Az *n* jelű ágon a következő vizsgálat az $x_2 x_3 = 00$ értékkombinációra vonatkozik. Az ebből kiinduló *n* jelű ág azt jelenti, hogy sem az $x_1 = 1$-re, sem az $x_2 x_3 = 00$-ra vonatkozó vizsgálatok nem eredményezték a vezérlési folyamat folytatódását, vagyis a ΔX^{12}, ΔX^{82}, ΔX^{13}, ΔX^{83} változások egyike sem játszódott le. Az $x_1 = 1$ és az $x_2 x_3 = 00$ szimbólumokból kivezető *i* jelű ágak rendre megegyeznek a 4.3. ábra ΔX^{12}, ΔX^{82}, ill. ΔX^{13}, ΔX^{83} változásokra várakozó ágaival. Megfigyelhetjük, hogy az $x_1 = 1$-re és az $x_2 x_3 = 00$-ra

vonatkozó vizsgálatok sorrendje a 4.4. ábrához képest felcserélhető a leírt vezérlési folyamat megváltozása nélkül.

Általánosan is megállapíthatjuk, hogy a specifikációs gráfon VAGY kapcsolatban szereplő specifikációs bemeneti változásokat jellemző vizsgálatok sorrendje tetszőleges lehet. Ez természetesen nem igaz a specifikációs gráfon egymást követő specifikációs bemeneti változásokra, amelyek a specifikációs bemeneti kombinációsorozatokat képviselik, és így általában jelértékekkel történő jellemzésük esetén sem változtathatjuk meg sorrendjüket az előírt működés megváltozása nélkül.

A 4.3. ábra bal oldali ágán haladva tovább ΔZ^{13} és ΔY^{12} változásokat kell előállítania a vezérlőegységnek. A feltételezett specifikációs kimeneti változások alapján ΔZ^{13} egyértelműen jellemezhető azzal, hogy a z_1 és a z_3 kimeneti jelek 0-ról 1-re változnak. A 4.4. ábrán ezt felfelé mutató nyilakkal jelöltük. Ugyancsak felfelé mutató nyíllal jelöltük Y_2-nek 0-ról 1-re történő változását, ami a szekunder kombinációkra feltételezett n-ből 1 kódok miatt ΔY^{12} változásnak felel meg. Az Y_1 1-ről 0-ra történő változását azért nem tüntettük fel, mert az n-ből 1 kód feltételezése lényegében kényszerkapcsolatot jelent a szekunder változók között: ha valamelyiknek az értéke 0-ról 1-re változik, akkor az n-ből 1 kód megőrzése megköveteli, hogy a korábban 1 értékű szekunder változó 0 értékűvé váljon. Így esetünkben $Y_2=1$ fellépésekor Y_1 értékének 0-ra történő változását is feltételezzük, mivel előzőleg ennek értéke volt 1. A vezérlőegység megvalósításakor természetesen gondoskodnunk kell majd a tervezési folyamat egyszerűsítése miatt feltételezett kényszerkapcsolat biztosításáról.

A 4.3. ábrán a bal oldai ág a ΔX^{25} változásra történő várakozással folytatódik. Ezt a specifikációs bemeneti változást egyértelműen jellemezhetjük az x_2 bemeneti jel értékének 0-ról 1-re történő változásával:

$$\Delta X^{25}: \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ 1 & 0 & 1 & 0 \quad X^2 \\ 1 & 1 & 1 & 0 \quad X^5 \end{array}$$

A 4.4. ábrán ezért elegendő a vezérlési folyamat folytatódásaként az $x_2=1$ értékre történő várakozás feltüntetése. A 4.4. ábra megszerkesztését ilyen gondolatmenettel folytathatjuk.

Figyeljük meg, hogy a $\boxed{\Delta X^{79}}$ specifikációs szakaszváltozás jellemzésére az összes bemeneti jelet fel kellett tüntetnünk a vizsgálatban az X^9-nek megfelelő jelértékek szerint. Csak így biztosítható ugyanis, hogy az X^7 megjelenése utáni összes lehetséges, X^9-et nem tartalmazó bemeneti kombinációsorozatra érzéketlenné váljon a vezérlőegység. Az X^9 megjelenésére történő várakozással egyúttal a ΔX^{89} specifikációs bemeneti változást is jellemezzük.

Általánosan is igaz, hogy a bemeneti specifikációs szakaszváltozások azonosításához az összes bemeneti jelet fel kell használnunk, ha az érzéketlenség folyamán bármilyen bemeneti kombináció felléphet. Ugyanis csak így tudjuk megkülönböztetni ezektől az új bemeneti specifikációs szakasz kezdetét. Ha ismerjük az érzéketlenség

idején lejátszódó bemeneti kombinációsorozatokat, akkor természetesen előfordulhat, hogy a specifikációs szakaszváltozás jellemzéséhez nincs szükség az összes bemeneti jelre.

Példaként még vizsgáljuk meg, hogy miként jellemezhető a 4.3. ábrán szereplő ΔX^{38}, ΔX^{67} és ΔX^{34}, ΔX^{64} bemeneti változásoktól függő elágazás a 4.4. ábrán:

$$\Delta X^{38}: \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ 0 & 0 & 0 & 0 \quad X^3 \\ 0 & 1 & 0 & 0 \quad X^8 \end{array} \qquad \Delta X^{67}: \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ 1 & 1 & 1 & 1 \quad X^6 \\ 0 & 1 & 1 & 1 \quad X^7 \end{array}$$

$$\Delta X^{34}: \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ 0 & 0 & 0 & 0 \quad X^3 \\ 1 & 0 & 0 & 0 \quad X^4 \end{array} \qquad \Delta X^{64}: \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ 1 & 1 & 1 & 1 \quad X^6 \\ 1 & 0 & 0 & 0 \quad X^7 \end{array}$$

A ΔX^{38} és ΔX^{67} közös jellemzője $x_1x_2=01$ megjelenése, a ΔX^{34} és ΔX^{64} változásokat pedig az $x_1x_2=10$ kombinációval azonosíthatjuk. E két kombinációra történő várakozással meg is különböztethetjük a két ágat. A két kombinációra vonatkozó vizsgálat sorrendjét a 4.4. ábrához képest természetesen fordított sorrendben is rajzolhattuk volna, hiszen az n jelű ágak mindaddig várakozást jelölnek, ameddig $x_1x_2=01$ és $x_1x_2=10$ valamelyike fel nem lép.

A 4.3. ábrán bemutatott ábrázolásmódból a fenti gondolatmenettel mindig eljuthatunk a 4.4. ábrán látható működésleíráshoz, amelyet a továbbiakban *folyamatábrának* nevezünk.

A folyamatábra a bemutatott megszerkesztési módjából következően lényegében az alábbi háromfajta műveletből épül fel:

1. Bemeneti jelek adott értékkombinációinak vizsgálata, amelynek eredménye alapján várakozás vagy elágazás történik. A továbbiakban az ilyen műveleteket *bemeneti műveleteknek* nevezzük. A bemeneti műveletet formálisan *várakozásnak* tekintjük, ha a vizsgált értékkombináció jelenlététől függően a műveletnek közvetlenül ismétlődik a végrehajtása. A 4.4. ábrán ilyen értelmezésben várakozásnak tekinthető például az y_4 fázisban levő $x_4=1$-re vonatkozó bemeneti művelet, amely $x_4=0$ esetén közvetlenül ismétlődik. A bemeneti műveletet formálisan akkor értelmezzük *elágazásként*, ha az illető művelet egyik ágon sem ismétlődik közvetlenül. A 4.4. ábrán így formálisan elágazásnak tekintjük például az y_1 fázisban az $x_1=1$-re vagy az $x_2x_3=00$-ra vonatkozó bemeneti műveletet, mert egyikük sem ismétlődik közvetlenül. A 4.4. ábra alapján azonban azt is megállapíthatjuk, hogy a várakozásokat csupán formai ismérvek alapján különböztettük meg. A specifikációs gráfból képzett folyamatábra bemeneti műveleteivel ugyanis a specifikációs bemeneti változásokra történő várakozásokat jellemeztük. Így tartalmilag a bemeneti műveletek mindegyike valamelyik specifikációs bemeneti változásra történő várakozás jellemzésének része. A formailag elágazásoknak tekintett y_1 fázisbeli $x_1=1$-re és $x_2x_3=00$-ra vonatkozó bemeneti műveletek például együttesen lényegében várakozást képviselnek a ΔX^{12}, ΔX^{82}, ΔX^{13}, ΔX^{83} specifikációs bemeneti változások valamelyikére.

A várakozások és az elágazások formai megkülönböztetését azért célszerű bevezetni, mert a későbbiekben, különösen a folyamatábra specifikációs gráf nélküli megszerkesztése esetén, tartalmi különbségeket is tulajdonítunk azoknak és az elvi logikai rajz meghatározásakor külön kezelést is igényelnek.

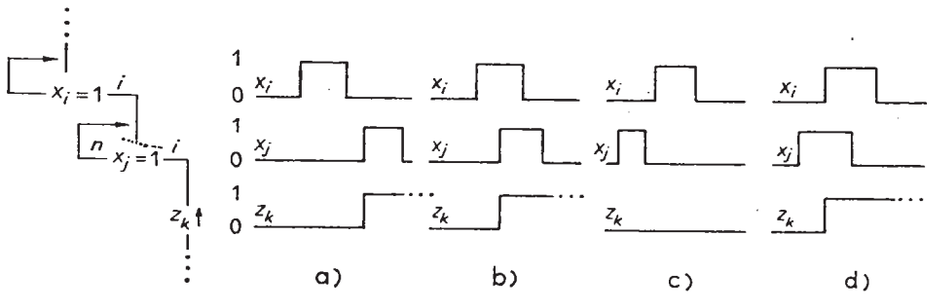
2. Kimeneti jelek értékét beállító műveletek, a továbbiakban *kimeneti műveletek*.

3. Szekunder változó 1 értékét beállító műveletek, a továbbiakban *szekunder műveletek*, amelyek az állapotváltozásokat, azaz a fázisátmeneteket okozzák.

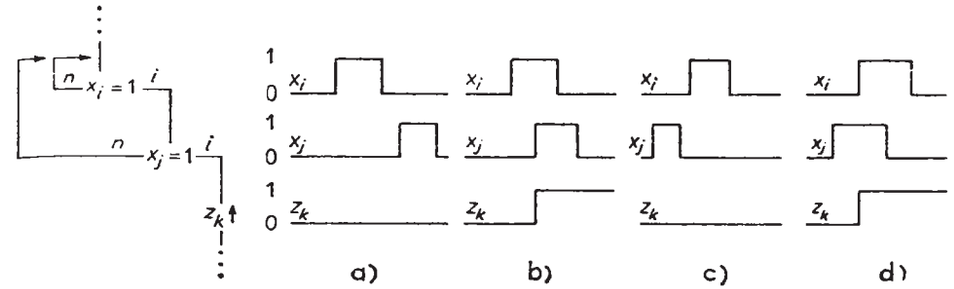
A szekunder kombinációkra feltételezett n-ből 1 kódból következő kényszerkapcsolat miatt az egyes működési fázisokat kijelölő y^i kombinációkat egyértelműen jellemezhetjük azzal az y_i változóval, amely az adott y^i kombinációban 1 értékű. A 4.4. ábrán ennek megfelelően azonosítottuk az egyes fázisokat, vagyis állapotokat. A folyamatábrán a műveleteket összekötő vonalak értelmezése a specifikációs gráfhoz képest változatlan.

Az ilyen módon megszerkesztett folyamatábrából a specifikációs bemeneti változások hatására az előírt működést olvashatjuk ki. Akadhatnak azonban olyan bemeneti kombinációsorozatok, amelyek nem tartoznak a specifikációs bemeneti változásokat alkotók közé, mégis egyértelműen követhetjük a hatásukra lejátszódó kimeneti és szekunder változásokat a folyamatábra alapján. Ennek oka nyilvánvalóan az, hogy a bemeneti műveletekben általában nem szerepel az összes bemeneti jel megváltozásának vizsgálata és a bemeneti specifikációs szakaszváltozások során sincsenek rögzítve a hatástalan bemeneti kombinációsorozatok.

Egy adott bemeneti műveletben nem szereplő jelek értékét tetszőlegesen rögzítve több olyan bemeneti kombinációsorozatot kaphatunk, amelyre a folyamatábra egyértelmű működést ír le. Természetesen adott esetben könnyű olyan bemeneti kombinációsorozatot is képezni, amelynek hatására a folyamatábra nem ad egyértelmű működésleírást. A folyamatábra formális értelmezése ugyanis a specifikációs bemeneti változások ismerete nélkül nem mindig egyértelmű. Elegendő ehhez a 4.5. ábrán bemutatott folyamatábra-részletre gondolnunk. Formálisan értelmezve azt mondhatjuk, hogy $z_k=1$ fellépésének előzménye az $x_i=1$, ill. az $x_j=1$ értékre vonatkozó várakozóműveletek i jelű ágán való továbbhaladás, vagyis az $x_i=1$ és $x_j=1$ értékek fellépése. Ha az x_i és x_j bemeneti jelek változásaira nem ismerjük a



4.5. ábra. Példa a folyamatábra értelmezésére egymás utáni várakozóműveletek esetén

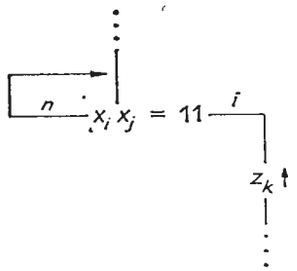


4.6. ábra. Példa a folyamatábra értelmezésére egymásba ágyazott várakozóműveletek esetén

specifikációs bemeneti változások, ill. szakaszváltozások által megfogalmazott megkötéseket, akkor a 4.5. ábrán idődiagramokkal ábrázolt esetek mindegyikére meg kell vizsgálnunk a működést. Ezek közül a folyamatábra alapján csak a *c)* esetben nem teljesül $z_k=1$ beállításának feltétele, ami a folyamatábrából formálisan következik, ha x_i és x_j ellenőrzését, vagyis a két várakozást az ábrázolt sorrendben tételezzük fel. Az *a)* esetben éppen azért teljesül $z_k=1$ beállításának feltétele, mert — bár $x_i=1$ és $x_j=1$ egyidejűleg ekkor sem áll fenn — a jelváltozások sorrendje megegyezik a folyamatábrán szereplő műveleti sorrenddel.

Vizsgáljuk meg, mi a hatása annak, ha a 4.5. ábra folyamatábra-részletét a 4.6. ábra szerint rajzoljuk át. Láthatjuk, hogy a 4.5. ábrához képest csak az *a)* esetben van eltérés. Az egymás utáni várakozóműveleteket ugyanis egymásba ágyazott várakozó (formailag elágazó) műveletekké alakítottuk át, aminek következtében $z_k=1$ beállításához feltétlenül szükségessé válik $x_i=1$ és $x_j=1$ egyidejű fennállása. A 4.6. ábra folyamatábra-részletéből az is látszik, hogy csakis ez az egyidejű fennállás okozza $z_k=1$ beállítását (az *a)* és *c)* esetben z_k egyaránt 0 értékű marad). Azt is megállapíthatjuk tehát, hogy a 4.6. ábra alapján a leírt működés nem függ a bemeneti jelek változási sorrendjétől, ellentétben a 4.5. ábrával, ahol az *a)* és *c)* esetben másmás működést olvashattunk ki a folyamatábrából. Az egymásba ágyazott várakozóműveletek esetében tehát a folyamatábrán történő továbblépés feltételeként mindig elegendő az eredő bemeneti kombináció fennállását vizsgálnunk, hiszen ilyen ábrázolásban a bemeneti műveletek sorrendje felcserélhető. A 4.6. ábra alapján a továbblépés feltétele tehát mindazon bemeneti kombinációk fellépése, amelyekben x_i és x_j egyaránt 1 értékű. Ezért a 4.6. ábra folyamatábra-részletét pl. a 4.7. ábra szerint is ábrázolhatnánk.

Fentiek alapján láthatjuk, hogy a folyamatábrából csak akkor tudjuk egyértelműen kiolvasni a vezérlőegység működését, ha ismerjük a specifikációs bemeneti változásokat. Csak így tudjuk eldönteni például azt, hogy a 4.5. vagy 4.6. ábra szerinti ábrázolás milyen bemeneti jelváltozások feltételezésén alapul. Később látni fogjuk, hogy a gyakorlatban általában a megoldandó vezérlési feladat ismeretében közvet-



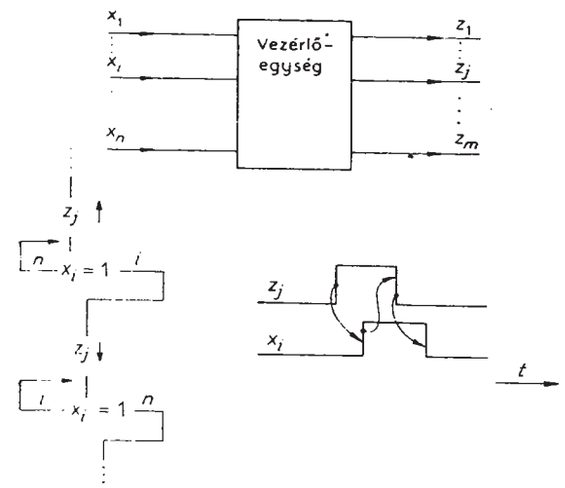
4.7. ábra. A 4.6. ábra folyamatábra-részletének ábrázolása a két bemeneti jelnek egy műveletben történő vizsgálatával

lenül is megszerkeszthetjük a folyamatábrát a specifikációs gráf nélkül is. Természetesen ilyenkor is ismernünk kell a bemeneti jelek mindenkor feltételezhető változásait, ha nem is éppen a specifikációs bemeneti változások, ill. szakaszváltozások formájában adottak. A folyamatábrán ugyanis mindig a specifikációs változásokat kell a jelváltozásokkal jellemezni, még akkor is, ha nem a specifikációs gráfból indulunk ki.

Sok esetben nem célszerű a vezérlési feladatot leíró jelváltozások ismeretében a folyamatábra megszerkesztése előtt a specifikációs bemeneti változásokat rögzíteni és a specifikációs gráfot létrehozni. A működést leíró egyes jelváltozások nyilvánvalóan nem szükségszerűen írnak elő megkötést az összes bemeneti jelre. Így a belőlük visszakövetkeztetett specifikációs bemeneti változások kombinációiban sok közömbös jelérték adódna, ami feleslegesen megnövelné a specifikációs bemeneti kombinációk számát, és ezáltal viszonylag bonyolult specifikációs gráf keletkezne egyszerű vezérlési feladatok esetén is.

A bemeneti jelek változásainak elemzésével és a specifikációs bemeneti változások jellemzésével akkor van a legkevesebb gondunk, ha feltételezhetjük, hogy a vezérlőegység működése ún. *„párbeszéd" (dialógus, hand-shaking) jellegű.* Ennek lényege az, hogy minden bemeneti változás létrehoz egy kimeneti változást és új bemeneti változás csak azután következik be, miután az azt közvetlenül megelőző kimeneti változás a vezérlőegység környezetében már kifejtette hatását. Az ilyen működés természetesen megköveteli, hogy a vezérlőegység környezete (a logikai rendszer környezete és a funkcionális egységek) is az ilyen szervezésnek megfelelően válaszoljon a kimeneti változásokra. A folyamatábrán a párbeszéd jellegű működés formálisan azt eredményezi, hogy a bemeneti várakozóműveletek és a kimeneti műveletek váltakozva következnek egymás után. Önmagában ebből a formális ismérvből azonban természetesen nem következik a párbeszéd jelleg, hiszen ehhez a bemeneti és kimeneti változások fentiekben leírt kölcsönhatása, vagyis a környezet megfelelő reagálása is szükséges, amelynek hiányában is kialakulhat a bemeneti és kimeneti műveletek formális váltakozása.

A párbeszéd jellegű működés előnye, hogy különböző és időben változó késleltető hatású vezetékeken is biztonságosan és mégis nagy sebességgel játszódhat le a jelforgalom két egység között. Egy kimeneti kombináció ugyanis csak akkor változhat meg, ha a környezetből megérkezett a vezérlőegységhez az a bemeneti kombináció, amely nyugtázza, hogy a környezetnek már nincs szüksége az illető kimeneti kombi-



4.8. ábra. A jelenkénti párbeszéd (hand-shaking, dialógus) jellegű működés szemléltetése

náció fennállására. Így nem fordulhat elő, hogy a kimeneti kombináció megváltozik, mielőtt a környezet azt értelmezni tudná. A párbeszéd jellegű működéshez természetesen elengedhetetlen, hogy a vezérlőegység környezete megfelelően visszajelzéseket adjon a vezérlőegység kimeneti kombinációira, vagyis rendre előállítsa számára az *ún. nyugtázó* bemeneti kombinációkat, amelyek észlelése után a vezérlőegység változtathat a kimeneti kombinációján. Ez a változtatás viszont a környezet számára jelenti a nyugtázást arra vonatkozóan, hogy már megváltoztathatja az általa előállított nyugtázó bemeneti kombinációt. Így a vezérlőegység és a környezet kölcsönösen letiltják egymás kimeneti kombinációinak változásait, mindaddig, amíg azok fennállására kölcsönösen szükségük van. A gyakorlatban legtöbbször jelenként érvényesül a párbeszéd jellegű működés, vagyis minden kimeneti jelnek van egy nyugtázó (hand-shaking) párja a bemeneti jelek között. Ez a megoldás igen egyszerűen értelmezhető és írható le a folyamatábra műveletei segítségével. A 4.8. ábrán a jelenkénti párbeszéd jellegű működés lényegét szemléltetjük folyamatábrával és idődiagrammal. A szekunder változásokat nem tüntettük fel, mert a működés magyarázata szempontjából nincs szerepük. A kizárólag jelenkénti párbeszéd jellegű lépéseket tartalmazó folyamatábra szerinti működés formálisan megvalósítható úgy, hogy a környezet az egyes kimeneti jeleket közvetlenül visszavezeti nyugtázójelekként. Az ilyen megoldásnak azonban nincs sok gyakorlati jelentősége, mert a kölcsönös nyugtázások megjelenése csak a jelterjedési késleltetésektől függ és a jelértékek így kiadódó időtartama általában túl rövid a kölcsönösen biztonságos észleléshez, ill. jelfeldolgozáshoz. Vizsgálatunkat a 4.8. ábrán a z_j kimeneti jel 1 értékre állításakor kezdjük, tehát ez az első vizsgált kimeneti kombinációváltozás. Ezt megelőzően természetesen várakozóműveletnek kell szerepelnie a folyamatábrán, ha a jelenkénti párbeszéd jelleg a teljes működésre érvényesül, de ettől az előzménytől ezúttal tekintsünk el. Ha feltételezzük, hogy z_j nyugtázó párja a bemeneti jelek közül x_i, éspedig oly módon,

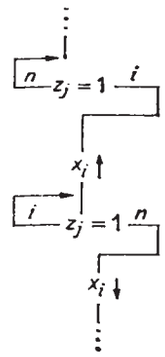
hogy $z_j=1$-re $x_i=1$ a válasz, akkor $z_j=1$ beállítása után nyilvánvalóan $x_i=1$ ellenőrzésének és az erre történő várakozásnak kell következnie a folyamatábrán. Miután a vezérlőegység az $x_i=1$-et tartalmazó bemeneti kombinációt észlelte, beállíthatja $z_j=0$-t, amit a környezet $x_i=0$ előállításával vesz tudomásul. A további működés párbeszéd jellegének biztosításához tehát szükséges, hogy a folyamatábrán csak akkor történhessen továbblépés $z_j=0$ beállítása után, ha $x_i=0$-t mint nyugtázást már észlelte a vezérlőegység. Ezért követi $z_j=0$ beállítását az $x_i=0$-ra történő várakozás.

A 4.8. ábra idődiagramján a vizsgált folyamatábra-részlet szerinti jelváltozásokat követhetjük. Az egyes jelváltozások közötti fentiekben részletezett ok—okozati összefüggéseket nyilakkal jelöltük. Megfigyelhetjük, hogy a nyugtázásokra történő kölcsönös várakozás következtében automatikusan mindig olyan impulzusszélességek alakulnak ki, amelyek az adott pillanatban érvényes jelkésleltetések és a kölcsönösen biztonságos észlelés, ill. feldolgozás időigénye miatt feltétlenül szükségesek. A jelek észlelése és feldolgozása így kölcsönösen biztonságossá válik. Ha vezérlőegység és a környezete közötti jelterjedési késleltetést, valamint az észlelési, ill. feldolgozási időket azonosnak tételezzük fel a bemeneti és a kimeneti jelekre vonatkozóan, akkor z_j és x_i összetartozó megváltozásai között mindig ennek az eredő késleltetési időnek a kétszerese telik el az oda-vissza történő jelterjedésre való várakozás miatt. A jelenkénti párbeszéd jellegű jelforgalom lehetséges legnagyobb sebességét tehát ennek figyelembevételével határozhatjuk meg. Nyilvánvaló, hogy egyszeres jelterjedési és észlelési idők alatt nyugtázó jelek nélkül, nem párbeszéd jelleggel is le lehetne bonyolítani az adott információtartalmú jelforgalmat, de ekkor a lehető legnagyobb sebesség esetén szigorú megkötéseket kellene tennünk a késleltető hatások időbeli állandóságára és az észlelési időigényekre vonatkozóan, ami különleges áramköri megoldásokkal sem mindig teljesíthető megbízhatóan.

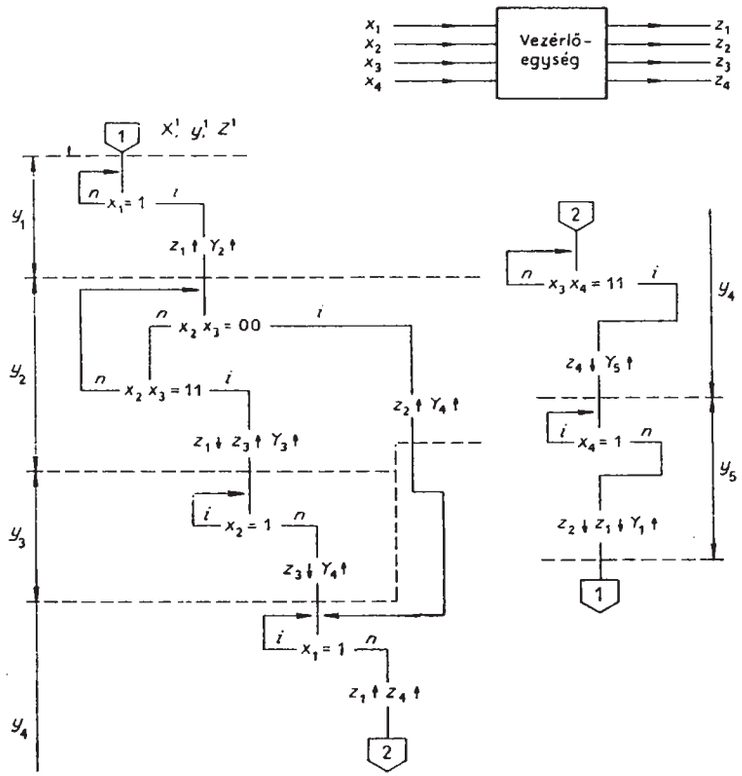
A párbeszéd jellegű jelforgalom elsősorban akkor ajánlott mint vezérlő jelkapcsolat, ha viszonylag nagy késleltető hatású összeköttetéseken keresztül történik információátvitel az összeköttetésekhez képest viszonylag gyors működésű egységek között. Ha az összeköttések jelkésleltető hatása kicsi, vagyis az egységek működési sebességéhez képest a jelterjedés gyors, akkor a nyugtázójel túl gyors megérkezése még átmeneti állapotban érheti a kezdeményező egységet, ami hazárdjelenségeket okozhat. Különösen az aszinkron működésű egységek közötti gyorsműködésű összeköttetéseken kell tehát a párbeszéd jellegű szervezést kerülni.

A jelenkénti párbeszéd jellegű jelforgalom tehát nagy késleltetésű összekötettések esetén teljes biztonságot nyújt azon az áron, hogy nyugtázójelekről kell gondoskodnunk és kétszeres jelterjedési időkkel kell számolnunk.

A párbeszéd jellegű jelforgalom során ún. *fennakadás* lép fel, ha olyan hiba keletkezik (pl. vezetékszakadás, adó-vevő erősítő hibája), amely megakadályozza a jelváltozás továbbítását. Ilyenkor a folyamatábrán valamelyik várakozás esetén nem teljesül a továbblépés feltétele. A gyakorlatban az ilyen hibák ellen ún. *időfigyelő-hálózatokkal (watch-dog)* védekeznek, amelyek adott időt meghaladó várakozás esetén



4.9. ábra. A 4.8. ábrán szereplő folyamatábra-részlet szerinti működés a vezérlőegység környezete szempontjából értelmezve



4.10. ábra. Példa párbeszéd jellegű működést leíró folyamatábrára

beavatkoznak a jelforgalomba és pl. beállítják a kiindulási helyzetet vagy hibajelzést adnak.

A 4.9. ábrán azt mutattuk be, hogy a vezérlőegység párbeszéd jellegű folyamatábrája alapján hogyan adható meg a környezet működésének folyamatábrája.

Megfigyelhetjük, hogy csupán a jelek bemeneti és kimeneti voltát kell ellentétesen értelmeznünk.

A 4.10. ábrán példaként megadtuk egy párbeszéd jelleggel működő vezérlőegység folyamatábráját. Figyeljük meg, hogy az ábrázolt működés értelmezésünk szerint nem teljes egészében jelenkénti párbeszéd jellegű, hiszen néhány esetben egy-egy várakozást követően több kimeneti jel megváltozása is bekövetkezik és több bemeneti jel értékkombinációjára történő várakozás is előfordul. A 4.10. ábra azt is szemlélteti, hogy a folyamatábra egyes külön rajzolt részeinek összefüggését címkékkel adhatjuk meg (2-es címke). Ezáltal a folyamatábrák áttekinthetőbben szerkeszthetők meg és az elágazások ábrázolása rajztechnikailag egyszerűbb.

A továbbiakban olyan módszereket ismerünk meg, amelyek segítségével a folyamatábrából kiindulva határozhatjuk meg a vezérlőegység elvi logikai rajzát.

4.2. Fázisregiszteres vezérlőegység tervezése a folyamatábra alapján

Az előzőekben bevezetett műveletekből felépített folyamatábra alapján kíséreljük meg létrehozni a vezérlőegység elvi logikai rajzát a következő gondolatmenettel. Az y_i változók és a működési fázisok értelmezése alapján egyszerűen kiolvashatjuk a folyamatábrából az egyes kimeneti jelek 1 értékre, ill. 0 értékre állításának (a továbbiakban megjelenésének és eltűnésének) feltételeit. A 4.4. ábra alapján az alábbi logikai függvények adódnak:

$$z_1{:}1 = y_1(x_1+\bar{x}_1\bar{x}_2\bar{x}_3) = y_1(x_1+\bar{x}_2\bar{x}_3),$$

$$z_1{:}0 = y_3x_1\bar{x}_2+y_5x_1\bar{x}_2x_3x_4,$$

$$z_2{:}1 = y_1\bar{x}_2\bar{x}_3,$$

$$z_2{:}0 = y_6,$$

$$z_3{:}1 = y_1x_1+y_3(\bar{x}_1x_2+x_1\bar{x}_2),$$

$$z_3{:}0 = y_4x_4+y_5(x_1\bar{x}_2x_3x_4\bar{x}_1+x_1\bar{x}_2\bar{x}_1) = y_4x_4+y_5x_1\bar{x}_2\bar{x}_1?$$

$$z_4{:}1 = y_4x_4+y_6$$

$$z_4{:}0 = y_5(x_1\bar{x}_2x_3x_4\bar{x}_1+x_1\bar{x}_2\bar{x}_1) = y_5x_1\bar{x}_2\bar{x}_1?,$$

ahol a kettőspont utáni 1 érték a kimeneti jel megjelenésére, a 0 érték pedig az eltűnésére utal. Kérdőjellel jelöltük meg azokat a függvényeket, amelyekre a formális felírás során ellentmondásos kifejezés adódott. Ennek okát és a kiküszöbölés módját később részletezzük.

A függvények formális kiolvasása és értelmezése a folyamatábra és a szekunder

kombinációk feltételezett tulajdonságai következtében igen egyszerű. Például a z_1 kimeneti jel esetén:

— z_1 megjelenik, ha $y_1=1$ melett $x_1=1$ vagy $x_2x_3=00$ fellép.

— z_1 eltűnik, ha $y_3=1$ mellett $x_1x_2=10$, vagy $y_5=1$ mellett $x_1x_2x_3x_4=1011$ fellép.

A $z_1{:}1$ függvényre felírt kifejezés követi a bemeneti műveletek ábrázolt sorrendjét, amely felcserélhető. A második tagban kezdetben szerepel az $\bar{x}_1$ tényező, amelyet nemcsak az algebrai egyszerűsíthetőség miatt hagyhatunk el, hanem azért is, mert az $x_1=1$-re és az $x_2x_3=00$-ra vonatkozó bemeneti műveleteket fordítva is rajzolhattuk volna. Az $z_2{:}1$ kifejezés felírásakor ezért az $\bar{x}_1$ tényezőt eleve el is hagytuk.

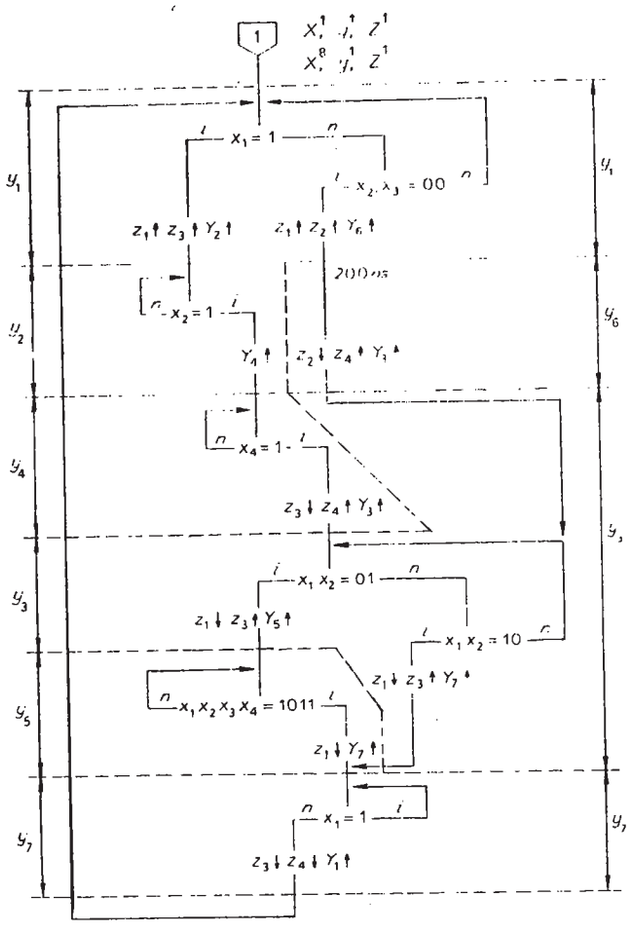
Az egyes kifejezések formális képzésekor nem szabad megfeledkeznünk arról, hogy az adott kimeneti műveletig esetleg több folyamatábraágon is eljuthatunk. Ilyenkor minden egyes ág mentén fel kell írnunk a feltételeket és ezeket VAGY kapcsolatba kell hoznunk. Ezt szemlélteti például $z_1{:}0$ kifejezése.

Gyakran előfordul, hogy egy adott fázis a folyamatábra több ágát tartalmazza. Ilyenkor nagyon fontos a fázison belüli ágak biztonságos megkülönböztetése a függvények felírásakor. Sokszor nem az adott fázisban, hanem valamelyik megelőző fázisban találunk olyan bemeneti műveleteket, amelyek alapján ezt a megkülönböztetést elvégezhetjük. Ilyen esetben azonban még fokozottabban kell ügyenünk arra, hogy a megelőző fázisban érvényes megkülönböztető jelértékek a fázisátmenet után is változatlan értékűek-e. Esetünkben például az y_5 fázis a folyamatábra két ágát foglalja magában. Például a $z_3{:}0$ függvény felírásakor a bal oldali ágat az $x_1x_2x_3x_4=1011$ kombinációval, a jobb oldali ágat pedig az ágat még az y_3 fázisban megkülönböztető $x_1x_2=10$ bemeneti kombináció fennállásával jellemeztük. A két ág találkozása után kimeneti változás feltételeként az egyes ágak mentén felírható feltételek VAGY kapcsolata az alábbi módon adódott:

$$y_5(x_1\bar{x}_2x_3x_4\bar{x}_1+x_1\bar{x}_2\bar{x}_1) = y_5x_1\bar{x}_2\bar{x}_1,$$

így ellentmondáshoz jutottunk, mert a formálisan képzett $x_1\bar{x}_1$ szorzat állandóan 0 függvényértéket állít elő. Az $x_1\bar{x}_1$ szorzat úgy jött létre, hogy az y_5 fázisban egymás után szerepel az $x_1=1$ értékre és az $x_1=0$ értékre vonatkozó várakozóművelet, anélkül, hogy közöttük állapotváltozás, vagyis fázisátmenet volna kijelölve. Így lényegében a 4.5*a)* ábrán vázolt helyzet áll fenn, amelyre nem érvényes a 4.7. ábra szerinti értelmezés. Az egymás utáni ellentétes x_1 értékekre vonatkozó várakozások esetén ugyanis biztosan nem tételezhetjük fel a 4.7. ábra szerinti értelmezéshez szükséges átfedést a várt jelértékek között. A $z_4{:}0$ függvény formális felírásakor viszont éppen a 4.7. ábra szerint értelmeztük a két egymás utáni várakozóműveletet. Ugyanilyen okból kaptuk az ellentmondásos kifejezést $z_4{:}0$ formális felírásakor.

A hibát kiküszöbölhetjük azáltal, hogy a két várakozóművelet közé állapotváltozást, ill. fázisátmenetet helyezünk el a folyamatábrán. Az így kiadódó szekunder változó értékével szétválaszthatjuk egymástól az x_1 és $\bar{x}_1$ tényezőket a $z_3{:}0$ és $z_4{:}0$ kifejezések formális felírásakor. Ehhez természetesen pótlólagos szekunder változóra



4.11. ábra. A 4.4. ábrán szereplő folyamatábra módosítása az x_1 jellel kapcsolatos két várakozóművelet közé felvett pótlólagos fázisátmenettel

van szükségünk. A 4.11. ábrán ennek megfelelően módosítottuk a folyamatábrát. Ezáltal a kimeneti jelek megjelenésének és eltűnésének feltételei a következőképpen írhatók fel:

$$
\begin{aligned}
z_1{:}1 &= y_1(x_1+\bar{x}_2\bar{x}_3),\\
z_1{:}0 &= y_3x_1\bar{x}_2+y_5x_1\bar{x}_2x_3x_4,\\
z_2{:}1 &= y_1\bar{x}_2\bar{x}_3,\\
z_2{:}0 &= y_6,\\
z_3{:}1 &= y_1x_1+y_3(\bar{x}_1x_2+x_1\bar{x}_2),\\
z_3{:}0 &= y_4x_4+y_7\bar{x}_1,\\
z_4{:}1 &= y_4x_4+y_6,\\
z_4{:}0 &= y_7\bar{x}_1.
\end{aligned}
\tag{4.1.}
$$

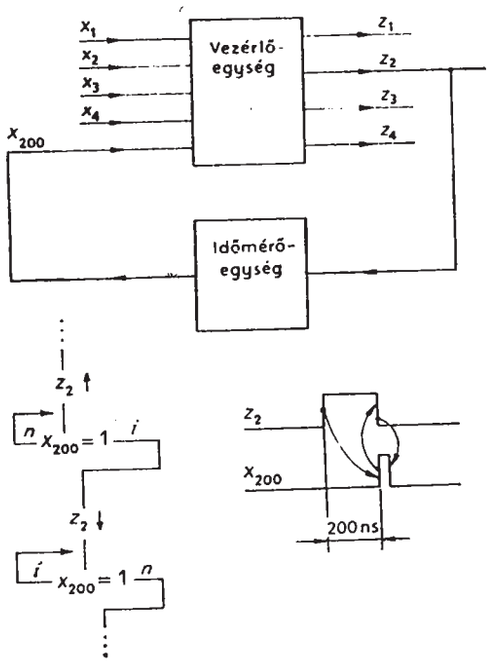
A pótlólagos fázisátmenet felvételével kapcsolatos módosítások természetesen visszahatnak a 4.3. ábrára is, hiszen módosítottuk a specifikációs szekunder változásokat. A 4.3. ábrán ez annyi módosítást jelent, hogy ΔZ^{56} mellett ΔY^{57}-et és ΔY^{51} helyett ΔY^{71}-et kell feltüntetnünk. Ezzel nyilvánvalóan megváltoztatjuk a 4.3. ábra megszerkesztésekor ismertnek feltételezett φ_z és φ_y, ill. f_z és f_y leképezéseket és egy pótlólagos szekunder változóra (y_7) is szükségünk van. Ezek az átalakítások azonban nyilvánvalóan nem változtatják meg a specifikációs bemeneti és kimeneti változások közötti kapcsolatokat, amelyeket a 4.3. ábra eredeti formájában is rögzít.

Ha minden kimeneti jel előállítását S—R flip-floppal képzeljük el, akkor az S bemeneteken a megfelelő z_i:1 függvényeket, az R bemeneteken pedig a z_i:0 függvényeket kell megvalósítanunk.

Ezek után gondoskodnunk kell az y_i változók előállításáról ahhoz, hogy a folyamatábrával leírt működés az y_i változók értékváltozásai révén lejátszódjon, feltételezve a folyamatábra kiindulási pontjáról (1-es címke) történő indítást. Az y_i változók értékeire rögzített kényszerkapcsolat miatt elegendő azt megadni, hogy az egyes $y_i=1$ értékek megjelenésének mi a feltétele. Az eltűnés ugyanis a kényszerkapcsolat révén következik be, hiszen feltételezésünk szerint bármelyik $y_i=1$ érték megjelenésének az összes többit 0 értékre kell állítania. Az y_i szekunder változó 1 értékének megjelenési feltételét nyilvánvalóan $Y_i=1$ fellépésének feltétele képezi, hiszen az $y_i=1$ érték mindig az Y—y visszahatás révén jön létre. A 4.11. ábra alapján az alábbi logikai függvényeket írhatjuk fel az egyes szekunder változók 1 értékének megjelenési feltételeire:

$$
\begin{aligned}
Y_1 &= y_7\bar{x}_1,\\
Y_2 &= y_1x_1,\\
Y_3 &= y_6+y_4x_4,\\
Y_4 &= y_2x_2,\\
Y_5 &= y_3\bar{x}_1x_2,\\
Y_6 &= y_1\bar{x}_2\bar{x}_3,\\
Y_7 &= y_3x_1\bar{x}_2+y_5x_1\bar{x}_2x_3x_4.
\end{aligned}
$$

A felírás gondolatmenete szerint például $y_1=1$ akkor jelenik meg, ha a folyamatábra 1-es címkéjére ér a vezérlési folyamat. Az 1-es címkére viszont a működés közben akkor jutunk, ha $y_7=1$ mellett $x_1=0$ lép fel. Az 1-es címkére, mint kiindulási pontra kell jutnunk természetesen akkor is, ha a vezérlőegységet alaphelyzetbe állítjuk. Az ehhez szükséges kiegészítéssel majd a vezérlőegység megvalósításakor foglalkozunk. Egyelőre az Y_i függvények felírásakor nem törődünk az alaphelyzetbe állítással. A folyamatábrán továbbhaladva például $y_2=1$ akkor jelenik meg, ha $y_1=1$ fennáll és $x_1=1$. Ezután $y_3=1$ megjelenésének feltételét írtuk fel, amely $y_6=1$ fennállásakor bemeneti jelre vonatkozó feltétel nélkül teljesül. A folyamatábrán $y_6=1$ és $y_3=1$ között ugyanis nincs bemeneti jelre vonatkozó művelet. Így azonban

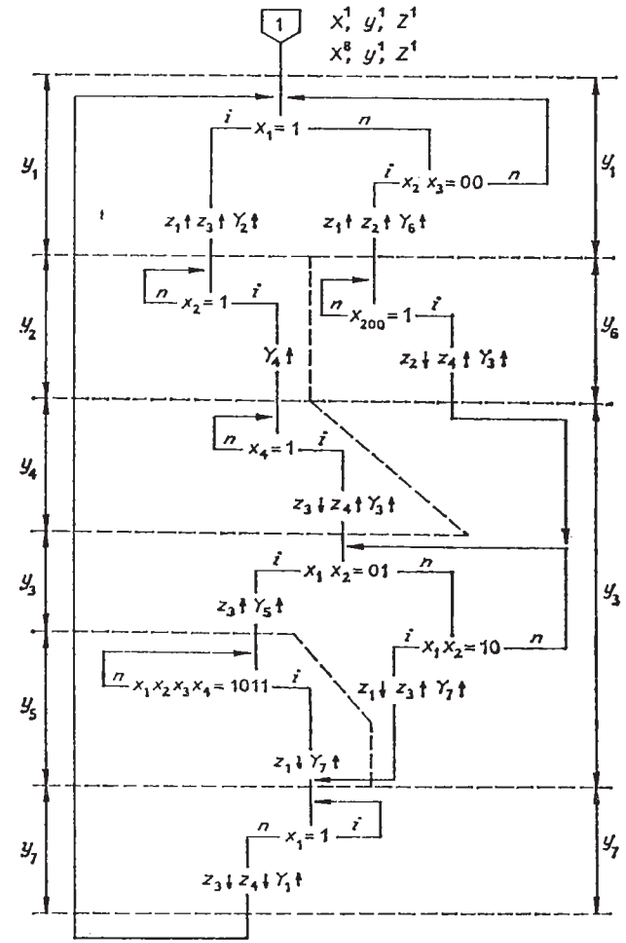


4.12. ábra. Előírt időtartamú kimeneti impulzus előállításához a folyamatábrán szükséges módosítások szemléltetése

$y_6=1$ fennállásának időtartamát nem adjuk meg egyértelműen, hiszen $y_6=1$ fellépése azonnal $y_3=1$-et okoz, ami rögtön $y_6=0$ beállításához vezet. Ily módon nem biztosítottuk, hogy a z_2 kimeneti jel a folyamatábrán előírt 200 ns ideig 1 értékű legyen. Ennek megoldásához fel kell tételeznünk, hogy a vezérlőegység számára egy funkcionális egység egy pótlólagos bemeneti jel értékének megváltoztatásával jelzi az előírt 200 ns idő leteltét. Ezt az időmérő funkcionális egységet a z_2 kimeneti jel megjelenése indíthatja el. Így a vezérlőegység bemeneti jelei közé a 4.12. ábra szerint pótlólag fel kell vennünk az x_{200} jelűt, és a működés folyamatábráját is ki kell egészítenünk azzal. Az ábrán feltételeztük, hogy az időmérőegység az előírt idő elteltével $x_{200}=1$-et hoz létre, és $x_{200}=0$-t állít be, ha $z_2=0$-t észlel. Megfigyelhetjük, hogy ilyen feltételezéssel a folyamatábra-részletet tulajdonképpen párbeszéd jellegűvé alakítottuk át. Ehhez tehát szükség volt az időmérőegységnek, mint pótlólagos funkcionális egységnek a felvételére, amit természetesen a logikai rendszer és ezen belül a vezérlőegység definíciójakor is megtehettünk volna. A pótlólagosan felvett x_{200} bemeneti jel természetesen módosítja a specifikációs bemeneti változásokat, így a 4.3. ábra szerinti specifikációs gráfot is. A fenti módosítások alapján Y_3 kifejezése az alábbi módon írható fel:

$$Y_3 = y_6 x_{200} + y_4 x_4 .$$

Ezáltal az $y_6=1$ által kijelölt fázis fennállásának idejét az $x_{200}=1$ érték megjelenésétől tettük függővé, ami megszabja $z_2=1$ fennállásának idejét.



4.13. ábra. A 4.11. ábrán szereplő folyamatábra módosítása az adott szélességű kimeneti impulzus előállítása céljából

A 4.13. ábrán felrajzoltuk a 4.11. ábrán szereplő folyamatábrának az időmérő egység figyelembevételével módosított változatát. Figyeljük meg, hogy az x_{200} jelre történő kiegészítéskor a 4.12. ábrán vázolt párbeszéd jellegű kapcsolatnak elegendő volt csak az első felét figyelembe venni. Az időmérőegység alaphelyzetbe állását ugyanis a folyamatábra szerinti továbblépéshez nem szükséges megvárnia a vezérlőegységnek. Ha azonban ugyanazt az időmérőegységet több kimeneti impulzus előállításához használjuk fel, akkor — mint azt később látni fogjuk — fontos a teljes párbeszéd jellegű folyamatot, vagyis az alaphelyzetbe állást is megvárni.

Az Y_i függvények a módosítások figyelembevételével az alábbiak:

$$
\begin{aligned}
Y_1 &= y_7\bar{x}_1,\\
Y_2 &= y_1x_1,\\
Y_3 &= y_6x_{200} + y_1x_1,\\
Y_4 &= y_2x_2,\\
Y_5 &= y_3\bar{x}_1x_2,\\
Y_6 &= y_1\bar{x}_2\bar{x}_3,\\
Y_7 &= y_3x_1\bar{x}_2 + y_5x_1\bar{x}_2x_3x_4.
\end{aligned}
\qquad (4.2.)
$$

Az x_{200} jel pótlólagos felvétele csak $z_2{:}0$ és $z_4{:}1$ kifejezését módosítja a (4.1.) függvények közül:

$$z_2{:}0 = y_6x_{200},$$

$$z_4{:}1 = y_4x_4 + y_6x_{200}.$$

A továbbiakban a vezérlőegységet megvalósító sorrendi hálózat elvi logikai rajzának meghatározásához a fenti értelmezésű folyamatábrát tekintjük kiindulási alapnak.

A (4.1.) és (4.2.) függvények felírásával tulajdonképpen a $\varphi_y(\Delta X, y) \Rightarrow \Delta Y$ és a $\varphi_z(\Delta X, y) \Rightarrow \Delta Z$ leképezéseket olvastuk ki a folyamatábrából, a feltételezett n-ből 1 kódú állapotok következtében igen egyszerűen. A φ_y és φ_z leképezések definíciójából következik, hogy ismeretükben az $f_z(X, y) \Rightarrow Z$ és $f_y(x, y) \Rightarrow Y$ leképezéseket is ismertnek tételezhetjük fel.

Figyeljük meg, hogy a példánkban szereplő vezérlőegység Mealy-modell szerint működik. A (4.1.) függvények alapján is látszik, hogy a kimeneti változásokat a bemeneti változások közvetlenül is befolyásolják. Moore-modellé úgy alakíthatjuk át a vezérlőegységet, hogy a folyamatábrán a kimeneti változásokat először rendre áthelyezzük a mindenkori soron következő fázisba, és ott első műveleteknek tekintjük azokat. Ezt szemlélteti a 4.14. ábra, amelyből az alábbi kimeneti függvényeket olvashatjuk ki:

$$z_1{:}1 = y_2 + y_6,$$

$$z_1{:}0 = y_7,$$

$$z_2{:}1 = y_6,$$

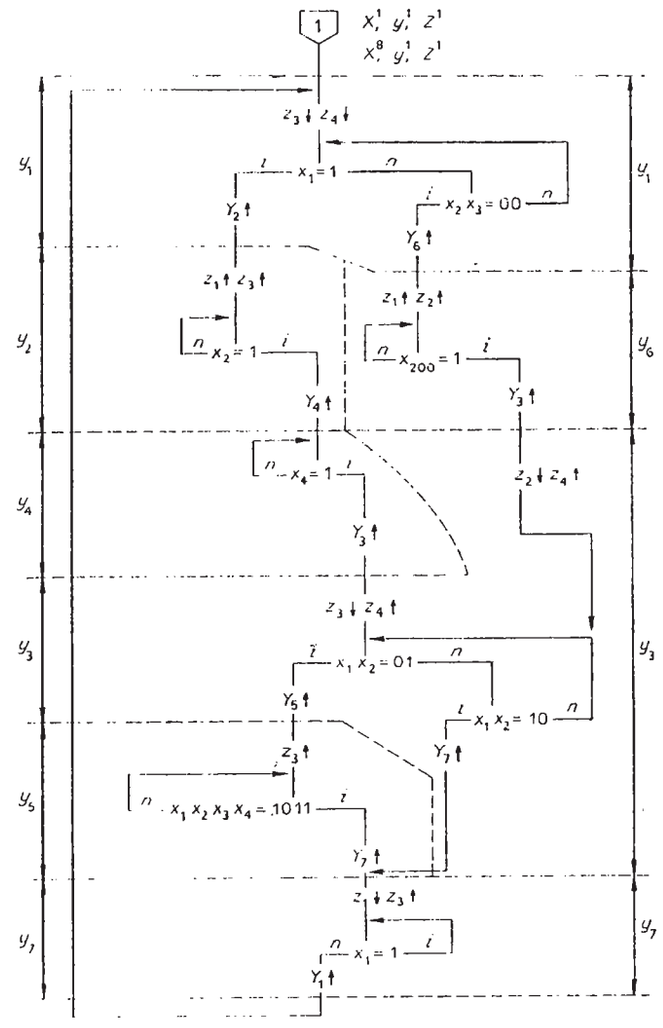
$$z_2{:}0 = y_3x_{200},$$

$$z_3{:}1 = y_2 + y_5,$$

$$z_3{:}0 = y_1 + y_3x_4,$$

$$z_4{:}1 = y_3$$

$$z_4{:}0 = y_1$$



4.14. ábra. A 4.13. ábrán szereplő folyamatábra átalakítása Moore-modellé

Láthatjuk, hogy csak a $z_2{:}0$ és $z_3{:}0$ függvények kifejezésében maradtak meg a bemeneti változók. Ahhoz, hogy ezeknek a függvényeknek is megszüntessük a közvetlen függőségét a bemeneti változóktól, az y_3 fázist kell két fázisra bontani. Ehhez természetesen pótlólagos szekunder változóra van szükség, és így az eredetileg y_3 fázison belül levő ágakat is kizárólag szekunder változókkal különböztethetjük meg. Ennek módja az eddigiek alapján könnyen elképzelhető, így további részletezését mellőzzük.

4.2.1. Szinkron fázisregiszteres vezérlőegység tervezése

A szinkron sorrendi hálózatok ismert működési modelljének megfelelően a fenti értelmezésben a 4.15. ábrán felrajzoltuk a 4.13. ábrán leírt működést megvalósító sorrendi hálózat vázlatos elvi logikai rajzát. A 3.6. pontban láttuk, hogy csak a reteszelt működésű (data-lock-out) flip-flop lehet a szinkron sorrendi hálózatok általános építőeleme. Ezért használtunk a 4.15. ábrán a visszacsatoló ágakban két-két élvezérelt D flip-flopból felépített reteszelt működésű D flip-flopokat.

A KH jelű több kimenetű kombinációs hálózat kimeneti függvényeit a (4.1.) és (4.2.) logikai függvények adják meg, amelyeket a folyamatábrából közvetlenül olvashattunk ki. A kimeneti jelek előállítását aszinkron S—R flip-flopok végzik a z_i:1 és z_i:0 logikai függvényeknek megfelelően.

A vezérlőegységet megvalósító szinkron sorrendi hálózat elvi logikai rajzát tehát a 4.15. ábra szerinti felépítésben a folyamatábra alapján közvetlenül felrajzolhatjuk, mivel a KH kombinációs hálózat kimeneti függvényei a folyamatábra és az y_i változók tulajdonságaiból következően a folyamatábrából közvetlenül kiolvashatók.

A 4.15. ábra szerinti felépítésben az egyes működési fázisokat kijelölő szekunder kombinációkat a működés során a visszacsatolóágbeli reteszelt flip-flopok, mint ún. *regiszterelemek* tárolják. Az ilyen felépítésű vezérlőegységet ezért *szinkron fázisregiszteres vezérlőegységnek* nevezik.

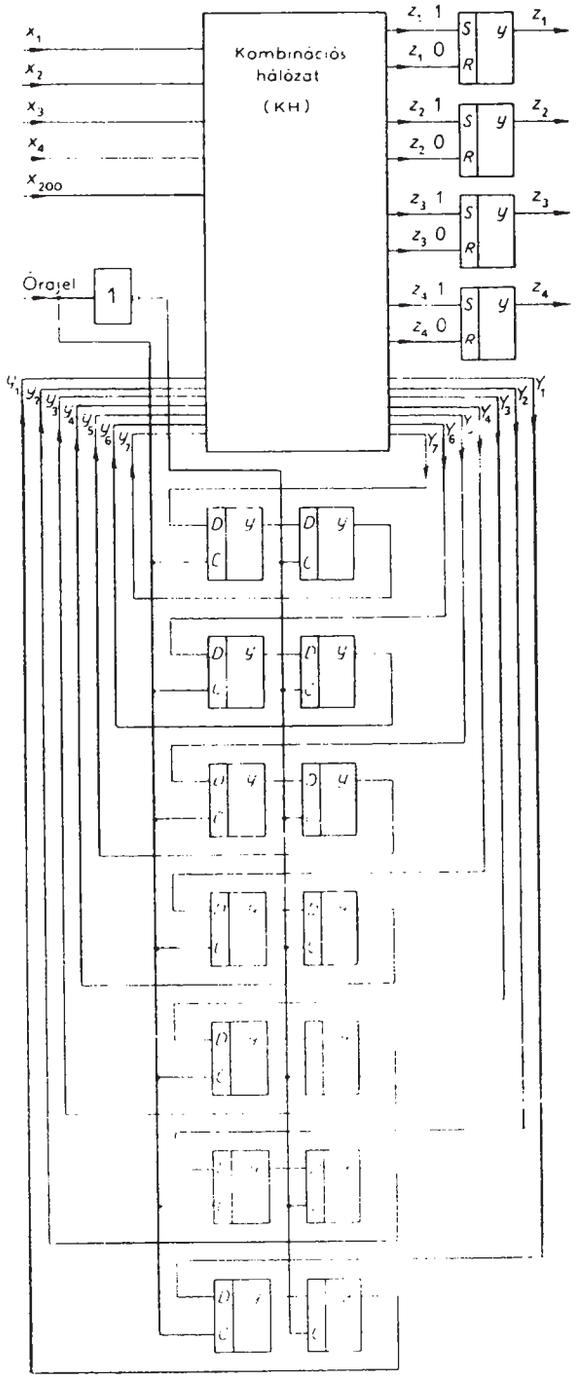
4.2.1.1. A szinkronizációs feltételek biztosítása a szinkron fázisregiszteres vezérlőegységben

Vizsgáljuk meg ezek után, hogy a 4.15. ábrán vázolt felépítésben megvalósul-e az y_i változók között feltételezett kényszerkapcsolat. Csak ekkor érvényes ugyanis a KH kombinációs hálózat kimeneti függvényeire alkalmazott egyszerű felírási mód. Láttuk, hogy a definiált kényszerkapcsolat n-ből 1 jellegű állapotkódolást jelent. Ennek teljesüléséhez az szükséges, hogy a működés során a visszacsatolóágakban levő flip-flopok mintavételi időpontjában ne léphessen fel olyan Y kombináció, amely eltér az n-ből 1 kódtól. Azt kell tehát biztosítani, hogy az óraimpulzus felfutásakor soha ne álljon fenn olyan Y kombináció, amelyben egynél több 1 érték van, vagy olyan, amelyben egyáltalán nincs 1 érték, azaz a csupa 0 kód.

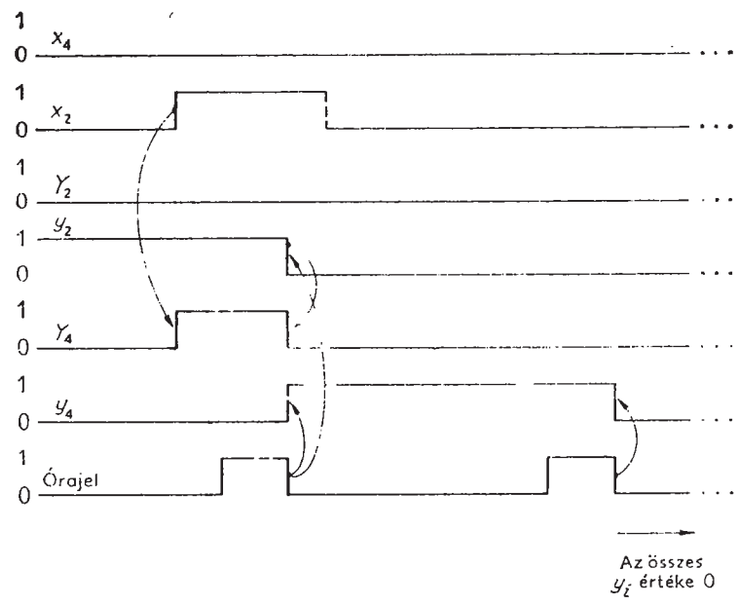
A szinkron sorrendi hálózatok vizsgálatakor (3.1. fejezet) láttuk, hogy a működés során létrejöhetnek olyan Y_i értékek, amelyeket már az új szekunder kombináció, de még a régi bemeneti kombináció állít elő. Tételezzük fel, hogy példánkban a

$$Y_4 = y_2 x_2$$

alakú függvénnyel leírt állapotváltozás időben a 4.16. ábra szerint játszódik le. Az állapotváltozás során $x_2 = 1$ bekövetkezésének hatására y_2 értéke 1-ről 0-ra, y_4 értéke pedig 0-ról 1-re változik a legközelebbi óraimpulzus lefutó élének érzékelése-



4.15. ábra. A 4.13. ábrán szereplő folyamatábra alapján felrajzolt vázlatos elvi logikai rajz



4.16. ábra. A csupa 0 állapotkód kialakulási lehetőségének szemléltetése a 4.15. ábrán vázolt hálózatban az $Y_4 = y_2 x_2$ függvénnyel leírt állapotváltozás esetén

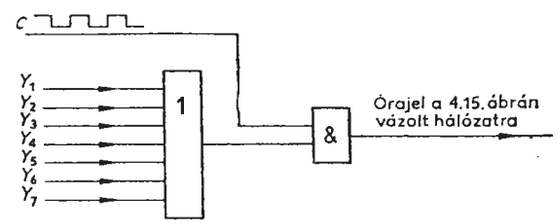
kor. Az ábrán megfigyelhetjük, hogy $x_2 = 1$ megjelenésekor $Y_4 = 1$ lép fel, ha $y_2 = 1$. Az ábrán a jelváltozások közötti ok—okozati kapcsolatokat nyilakkal jelöltük.

Az ábrázolt első óraimpulzus lefutó élének hatására így $y_4 = 1$ és $y_2 = 0$ jön létre. Ez utóbbi azért vesz fel 0 értéket, mert a vizsgált helyzetben $Y_2 = 0$ (l. 4.2. függvények). Az $y_2 = 0$ érték megjelenésének hatására Y_4 értéke is 0 lesz. Ha a bemeneti jelek nem változnak az ábrázolt második óraimpulzus felfutó éléig, akkor a lefutó él hatására az összes y_i érték 0 lesz. A 4.2. függvények alapján ugyanis $y_4 = 1$ mellett $Y_3 = 1$ léphetne fel $x_4 = 1$ esetén. Az ábrázolt esetben azonban feltételezzük, hogy x_4 értéke mindvégig 0. Így az ábrán feltüntetett első óraimpulzus után az összes Y_i érték 0-vá válik és a második óraimpulzus lefutó éle beállítja a csupa 0 állapotkódot, ami nem biztosítja az y_i változók közötti feltételezett kényszerkapcsolatot. Nem is folytatódhat ekkor a folyamatábrával leírt működés, bármilyen bemeneti kombináció lép is fel ezután, mert a 4.2. függvények mindegyike csak valamelyik y_i változó 1 értéke mellett szolgáltathat 1 függvényértéket. Így, ha a működés során az óraimpulzus felfutásakor egyszer fennáll a csupa 0 állapotkód, akkor ebből az állapotból a 4.15. ábra szerinti hálózat a bemeneti jelek hatására nem tud kimozdulni. Ennek következtében tehát a folyamatábrával leírt működés nem valósul meg, mivel megszűnik az y_i változók között feltételezett kényszerkapcsolat, ami a (4.1.) és (4.2.) függvények felírhatóságának is alapfeltétele volt.

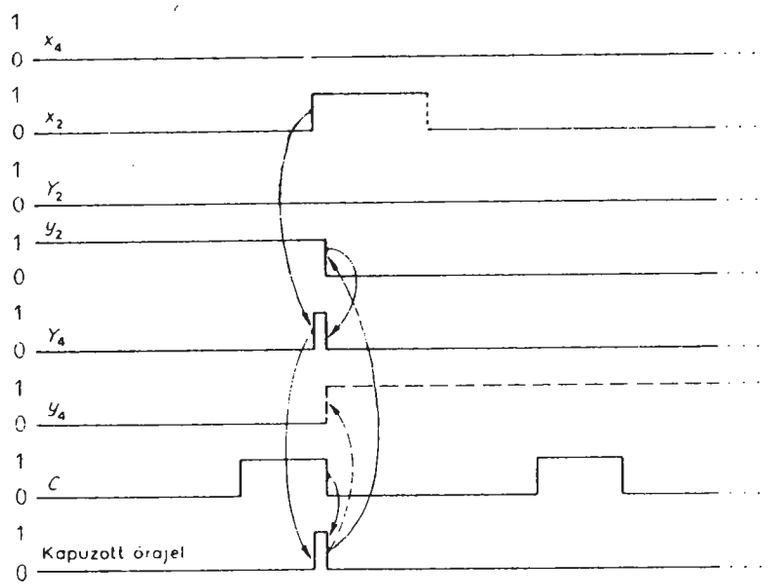
A fentiekben vázolt hibát kiküszöböljük, ha nem engedünk óraimpulzust a 4.15. ábra szerint felépített hálózatra olyankor, amikor az összes Y_i érték 0. A vizsgált esetben ez azt jelenti, hogy a 4.16. ábra második óraimpulzusát nem engedjük hatni. A megoldás egy lehetséges módját követhetjük a 4.17. ábrán. Eszerint a 4.15. ábrán vázolt hálózatra csak akkor jut óraimpulzus a C órajelből, ha legalább egy Y_i értéke 1. Így például a 4.16. ábra második óraimpulzusa nem jut a hálózatra, tehát az összes Y_i értéke most már nem hozhatja létre a csupa 0 állapotkódot. Az $y_4 = 1$ értékkel jellemezhető n-ből 1 kódú állapot ezáltal mindaddig fennmarad, amíg a bemeneti kombináció megváltozása újabb $Y_i = 1$ értéket nem eredményez, vagyis esetünkben amíg $x_4 = 1$ megjelenése az $Y_3 = 1$ értéket létre nem hozza.

Figyeljük meg, hogy a bemutatott megoldásnak az a lényeges következménye, hogy a hálózatra nem állandó frekvenciájú órajel jut, hanem a bemeneti változások az Y_i értékeken keresztül engedélyezhetik vagy tilthatják az egyes óraimpulzusok hatását. A bemeneti változásokra és a visszacsatolóágak nyitására-zárására vonatkozó szinkronizációs feltételt tehát úgy teljesítettük, hogy a visszacsatolóágak nyitását-zárását a bemeneti változások ütemezik.

A 4.17. ábra szeinti órajelkapuzás mellett is előfordulhat azonban hibás működés, ha a bemeneti jelek tetszőleges időpontban változhatnak. Ennek szemléltetésére a 4.18. ábrán felrajzoltuk a 4.16. ábrán szereplő állapotváltozás idődiagramját azzal a feltételezéssel, hogy az x_2 bemeneti jel a C állandó órajel lefutó éléhez igen közel változik 0-ról 1-re. Ilyenkor a lefutó élhez igen közel lép fel $Y_4 = 1$, miáltal a C állandó órajel impulzusából csak nagyon rövid időtartamú kapuzott óraimpulzust állít elő a 4.17. ábra szerinti órajelkapuzó hálózat. Előfordulhat, hogy a túl keskeny impulzus csak az y_2 változónak megfelelő flip-flopot képes billenteni, az y_4-nek megfelelőt nem. Az ábrán szaggatottan jelöltük az esetleg elmaradó jelváltozást. Így a kapuzott óraimpulzus lefutó élének hatására létrejöhet a csupa 0 állapotkód, ami hibás működést jelent. Hasonlóképpen hibát okoz az is, ha a keskeny kapuzott óraimpulzus csak az $y_4 = 1$ értéket tudja beállítani és az $y_2 = 0$-t nem. Ilyenkor a lefutó él után y_4 és y_2 egyaránt 1 értékű lesz, ami szintén az n-ből 1 kódtól eltérő állapotkódot és így hibás működést okoz. Hasonló gondolatmenettel láthatjuk be azt is, hogy hibás működést okozhat a C állandó órajel felfutó éléhez túl közeli bemeneti jelváltozás. Ilyenkor ugyanis a felfutó élhez igen közeli időtartományban változnak



4.17. ábra A csupa 0 állapotkód kialakulásának megakadályozása az órajel kapuzásával

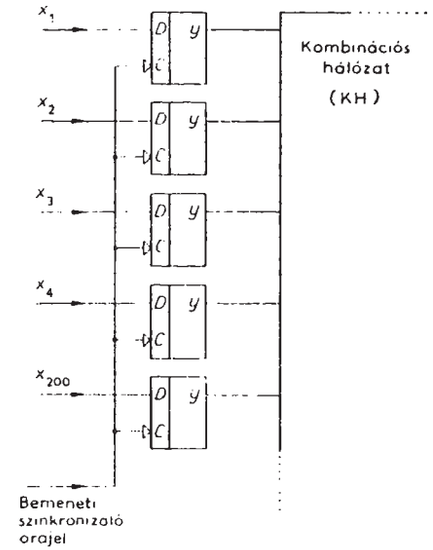


4.18. ábra. A 4.16. ábrán szereplő idődiagram módosulása, ha az x_2 bemeneti jel a C állandó órajel lefutó éléhez igen közel változik 0-ról 1-re

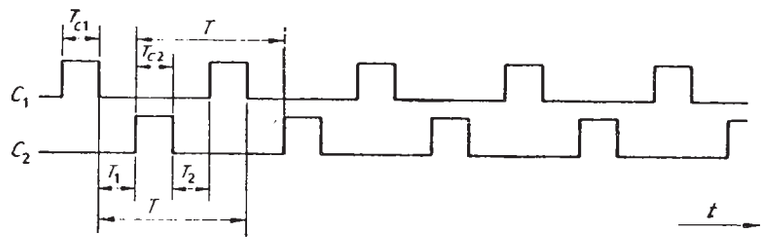
az Y_i értékek is. A (4.2.) függvényektől függően előfordulhat, hogy a bemeneti változás okozta átmeneti jelenségek következtében több Y_i is átmeneti értékeket vesz fel (lásd funkcionális hazárd). Ha a visszacsatolóágakban levő flip-flopok az óraimpulzus közeli felfutó éle miatt ezekből az átmeneti Y_i értékekből vesznek mintát, akkor a lefutó él hatására ezúttal is hibás y kombináció alakulhat ki.

A fenti hibalehetőségeket nyilvánvalóan kiküszöbölhetjük, ha a bemeneti jelek változásait csak az óraimpulzus éleitől jól meghatározott időbeli távolságban engedjük hatni a KH hálózatra. Ennek megoldásaként a 4.19. ábrán a bemeneti jeleket egy-egy élvezérlésű D flip-flopon keresztül vezetjük a KH hálózatra. Ezeket az ún. *bemeneti szinkronizló flip-flopokat* olyan órajellel kell vezérelnünk, amelynek élei a visszacsatoló ágbeli flip-flopok órajeléleihez képest az említett okokból jól meghatározott helyzetben vannak. Ezt a követelményt kielégíthetjük a 4.20. ábrán vázolt ún. *kétfázisú órajellel*, amelyet a C_1 és C_2 órajelek alkotnak. A működés akkor lesz hibamentes, ha a T_1 és T_2 idők elég nagyok ahhoz, hogy az élek hatását elkülönítsék a hálózatban.

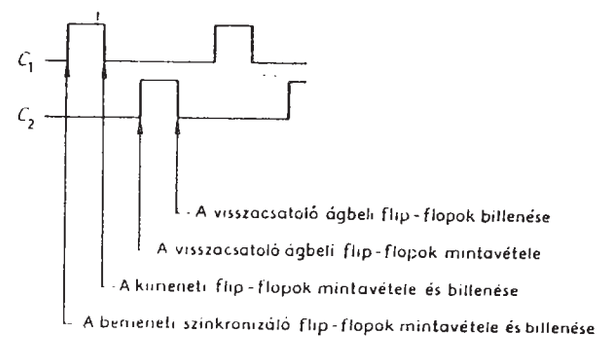
Az egyes élek szerepét például úgy jelölhetjük ki, hogy a C_1 órajel felfutó élei végzik a bemeneti szinkronizálást, a C_2 órajel impulzusai pedig a visszacsatolóágban levő reteszelt működésű flip-flopokat vezérlik. Ebből viszont az következik, hogy Mealy-modell esetén a kimeneti S—R flip-flopok bemeneteit a C_1 óraimpulzus hatására lezajló szinkronizált bemeneti változások is befolyásolják. Mivel ezek a bemeneti



4.19. ábra. A 4.15. ábrán vázolt hálózat kiegészítése a bemeneti szinkronizáló flip-flopokkal

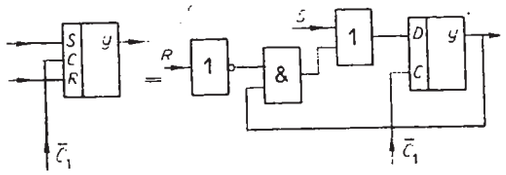


4.20. ábra. Kétfázisú órajel szemléltetése



4.21. ábra. A kétfázisú órajel éleinek hatása

változások nem szükségszerűen szomszédosak, ezért a funkcionális hazárd miatt a kimeneti S—R flip-flopok helytelen állapotba billenhetnek. Ezt a hibalehetőséget úgy küszöbölhetjük ki, hogy órajelvezérlésű kimeneti flip-flopot alkalmazunk. A 4.21. ábrán bemutattuk az alkalmazott kétfázisú órajel egyes éleinek felhasználását a



4.22. ábra. A kimeneti S—R flip-flop órajelvezérlésű változatának felépítése élvezérlésű D flip-flopból

kombinációváltozások ütemezésére, a 4.22. ábrán pedig a kimeneti S—R flip-flop órajelvezérlésű változatát élvezérlésű D flip-flopból felépítve. Így a bemeneti változást C_1 felfutó éle szinkronizálja, a kimeneti változást pedig C_1 lefutó éle. Ezáltal nem léphet fel hibás kimeneti érték funkcionális hazárd következtében, ha a T_{C1} idő alatt biztosan lejátszódnak az átmeneti jelenségek. A C_1 órajel impulzusszélességének megválasztásakor tehát erre feltétlenül ügyelnünk kell. A kétfázisú órajel szükségszerű alkalmazásával természetesen lassítjuk a hálózat működését az egyetlen órajeles működtetéshez képest, hiszen az egyes jelváltozástípusoknak a fenti hibák kiküszöbölése céljából történő biztonságos elkülönítése csak időben sorosan képzelhető el.

Az órajelkapuzással, bemeneti szinkronizálással és kétfázisú órajellel módosított 4.15. ábra szerinti felépítés esetén is észlelhet a KH hálózat az y kombináció változásakor átmenetileg n-ből 1 kódtól eltérő állapotkódot. Minden állapotváltozás esetén ugyanis az egyik y érték 1-ről 0-ra, a másik pedig 0-ról 1-re változik.

A jelterjedési késleltetések következtében a KH hálózat átmenetileg mindkét értéket 0-nak vagy 1 észlelheti a nem szomszédos változások miatt (l. funkcionális hazárd). Ez azonban csak a C_2 óraimpulzus lefutó élének közelében következhet be, a T_2 időt pedig nyilvánvalóan úgy kell megválasztanunk, hogy a C_1 óraimpulzus felfutó élének megjelenéséig az átmeneti jelenségek már biztosan lejátszódjanak. Így nem történhet mintavétel az átmenetileg fennálló Y kombinációkból.

4.2.1.2. A szekunder változók számának csökkentése a folyamatábra alapján

A 4.15. ábra és a bemutatott kiegészítések szerinti szinkron sorrendi hálózattal megvalósított vezérlőegység működési sebességét adott órajel-frekvencia mellett a következő gondolatmenettel becsülhetjük meg. Ahhoz, hogy a folyamatábrával leírt műveleteket végrehajtsa a vezérlőegység, az szükséges, hogy a folyamatábrán a mindenkori elágazási feltételeknek megfelelően végighaladva az „útbaeső" összes y_i változó rendre 1 értéket vegyen fel és rendre megtörténjenek az útbaeső kimeneti változások. Tudjuk, hogy minden egyes y_i változáshoz, ill. kimeneti változáshoz egy teljes C_2, ill. C_1 impulzus szükséges. Ez azt jelenti, hogy az y_i változások leggyakrabban C_2 periódusidőnként, a kimeneti változások pedig C_1 periódusidőnként következhetnek egymás után, amiből kiszámíthatjuk az elérhető működési sebesség

felső korlátját egy adott órajel-frekvencia mellett. Az így kiadódó működési sebességet természetesen csak akkor éri el a vezérlőegység, ha minden egyes C_1 impulzus felfutó élének pillanatában már jelen van az a bemeneti kombináció, amely a soron következő fázis-, ill. kimeneti változáshoz szükséges. Ha ugyanis pl. egy y_i változás után következő C_1 impulzus felfutó éléig nem alakult ki a következő y_i változást előidéző bemeneti kombináció, akkor a 4.17. ábrán bemutatott órajelkapuzás miatt a soron következő C_2 impulzusok mindaddig hatástalanok lesznek az állapotváltozás szempontjából, amíg egy C_1 impulzus felfutó éle a további y_i változáshoz szükséges bemeneti kombinációval nem találkozik. Így tehát a bemeneti változások időpontjától függően előfordulhat, hogy nem minden C_2 periódusban történik y_i változás, emiatt a működési sebesség az elérhető felső korlátnál kisebb lesz. A továbbiakban látni fogjuk, hogy a működés során szükséges y_i változások számának csökkentése sok esetben növeli a működési sebesség felső korlátját.

A folyamatábra bevezetésekor az ismertnek feltételezett φ_z és φ_y leképezések, valamint a specifikációs bemeneti, kimeneti és szekunder változások alapján megszerkesztett specifikációs gráfból indultunk ki. A folyamatábrához úgy jutottunk el, hogy a specifikációs változásokat a lehető legkevesebb jel megváltozásával jellemeztük, szem előtt tartva a szükséges megkülönböztetéseket. Láttuk, hogy a formálisan értelmezett folyamatábrából már nem tudunk egyértelműen visszakövetkeztetni a specifikációs bemeneti változásokra. A folyamatábrát viszont csak akkor tudjuk ellentmondásmentesen értelmezni, ha a specifikációs bemeneti változásokat ismerjük. A gyakorlati tervezés során sokszor nem tudunk a specifikációs gráfból kiindulni, mert nem ismerjük a φ_z és φ_y leképezéseket. A tervezés célja ugyanis éppen az, hogy ezeket a leképzéseket meghatározzuk és megvalósítsuk a megoldandó logikai feladatnak megfelelően. Ezenkívül a specifikációs bemeneti és kimeneti változásokat sem mindig célszerű felsorolni, mert általában ezek nem kombinációváltozások, hanem jelváltozások formájában adottak. Így a megoldandó logikai feladat alapján általában könnyebb közvetlenül megszerkeszteni a folyamatábrát, mint a specifikációs gráfból kiindulni. A megoldandó logikai feladat alapján közvetlenül megszerkesztett folyamatábra kezdetben általában csak a bemeneti és a kimeneti változások közötti kapcsolatokat tartalmazhatja, mert a φ_y leképezést éppen ezeknek megfelelően kell majd meghatározni a tervezés során. A specifikációs bemeneti változásokról feltétlenül tudnunk kell annyit, hogy a megoldandó feladatban őket jellemző jelváltozások során rendre melyek azok a jelek, amelyek az egyes változások idején állandó értékűeknek tekinthetők. Ennek ismerete nélkül ugyanis előfordulhat, hogy az egyes bemeneti jelváltozások által okozott kimeneti jelváltozások logikai függvényei nem olvashatók ki a folyamatábráról ellentmondásmentesen (l. 4.5., 4.6., 4.7. ábrák és pl. a 4.4. ábra alapján felírt z_4:0 kifejezés).

A szekunder változók szerepe is lényegében az, hogy a kimeneti változások logikai függvényeinek algebrai alakját ellentmondásmentesen tudjuk képezni. Az n-ből 1 jellegű állapotkódok következtében a szekunder változók megkülönböztető szerepét igen egyszerűen tudjuk figyelembe venni az algebrai alakok képzésekor.

A megoldandó logikai feladat alapján közvetlenül megszerkesztett folyamatábrához tehát úgy kell hozzárendelnünk a szekunder változásokat, azaz a fázishatárokat, hogy a kimeneti jelek megjelenésének és eltűnésének, valamint a fázisváltozásoknak a logikai függvényei egyértelműen legyenek felírhatók. Vizsgáljuk meg, hogy miként lehetne a lehető legkevesebb fázist, ill. állapotot kijelölni, vagyis a lehető legkevesebb szekunder változót felvenni.

Könnyen beláthatjuk, hogy minél kevesebbet tudunk az egyes specifikációs bemeneti változások során állandó értékű jelekről, annál több megkülönböztető feladat hárul a szekunder változókra, vagyis várhatóan növekszik a folyamatábrán kijelölendő fázisok száma. A továbbiakban feltételezzük, hogy adott a bemeneti és kimeneti változások közötti kapcsolatokat leíró folyamatábra a fázisok feltüntetése nélkül.

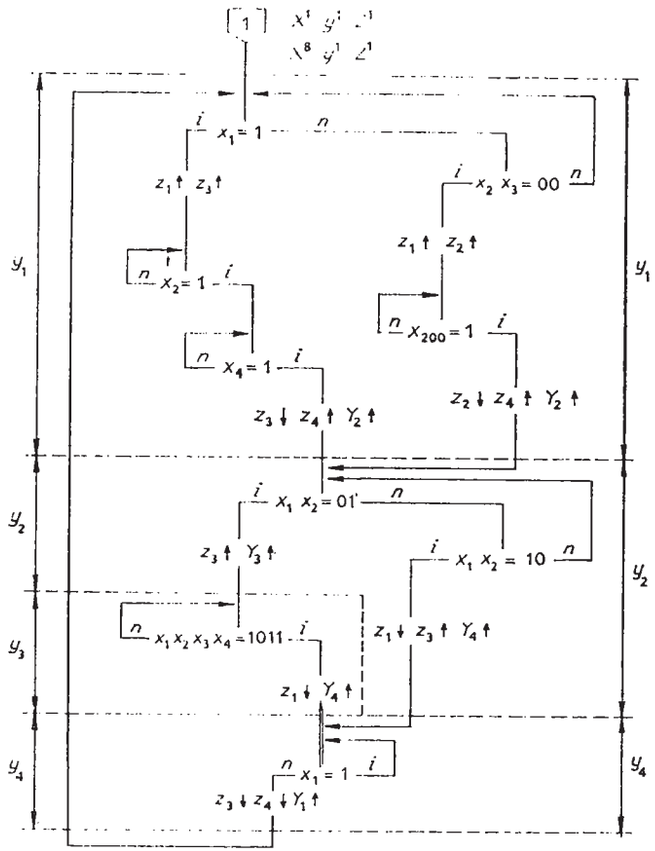
Bemutatjuk, hogy a specifikációs bemeneti változások ismeretének egy adott szintjén milyen gondolatmenettel célszerű a fázishatárokat kijelölni.

A folyamatábra kiindulási pontjától kezdődően vegyük fel az $y_1=1$ értéket, és újabb $y_i=1$ értéket (pl. $y_2=1$-et) csak akkor rendeljünk a folyamatábrához, ha azon végighaladva valamelyik z_i:1 vagy z_i:0 függvényt már nem tudjuk ellentmondásmentesen felírni az addig felvett y_i változók segítségével, ill. a specifikációs bemeneti változások vagy szakaszváltozók ismeretében.

A 4.23. ábrán bemutatjuk, hogy ezzel a gondolatmenettel miként lehet a szekunder változókkal takarékoskodni a 4.14. ábrán szereplő folyamatábra esetén. A 4.23. ábra bal oldali ágán haladva az y_1 fázis azért tarthat az $x_1x_2=01$ kombinációtól függő elágazó műveletig, mert a ΔX^{12} (ΔX^{82}), ΔX^{25}, ΔX^{56} specifikációs bemeneti változások során az $x_1=1$, $x_2=1$ és $x_4=1$ jelértékek rendre megjelennek és a további bemeneti változásig egyidejűleg fenn is maradnak. Így a kimeneti függvényeket a 4.7. ábra szerinti értelmezés alapján egyértelműen felírhatjuk.

Megfigyelhetjük, hogy a folyamatábra szerint ΔX^{25} lejátszódására érzéketlen a vezérlőegység, hiszen sem Y, sem Z változás nem követi közvetlenül ΔX^{25}-öt. Ezért ezen a folyamatábra-szakaszon specifikációs szakaszváltozást is értelmezhetnénk. Mivel azonban ebben az esetben az érzéketlenségi tartomány csak egyetlen bemeneti változásból áll, ezért egyszerűbb függvényeket kapunk szakaszváltozás definiálása nélkül.

A jobb oldali ágon haladva az $y_1=1$ érték által kijelölt fázis azért tarthat $x_{200}=1$ fellépéséig, mert az időmérő egységgel a 4.12. ábrán vázolt „párbeszéd" jellegű kapcsolatban működik együtt a vezérlőegység. Így az x_{200} bemeneti jel pótlólagos felvétele révén kialakuló specifikációs bemeneti változásokat az $x_2x_3=00$-ra és az $x_{200}=1$-re történő várakozások egyértelműen megkülönböztetik. Az x_{200} bemeneti jel értéke ugyanis a 4.12. ábra alapján csak akkor változhat 0-ról 1-re, ha $z_2=1$ megjelent, vagyis az $x_2x_3=00$ kombináció már fennáll. Az x_{200} bemeneti jel 0-ról 1-re történő változása pedig nem befolyásolja az $x_2x_3=00$ kombináció fennállását. Így a két művelet egyetlen várakozóműveletként értelmezhető a 4.7. ábrán vázolt módon.



4.23. ábra. A működési fázisok kijelölése a szekunder változók számának csökkentése céljából a 4.14. ábrán szereplő folyamatábrából kiindulva

Hasonló okokból nem szükséges fázishatárt felvenni a bal oldali ágon az $x_2=1$-re és az $x_4=1$-re várakozó műveletek közé. Ezek ugyanis a ΔX^{25} és ΔX^{56} specifikációs bemeneti változásokat különböztetik meg és feltételezésünk szerint ΔX^{56} során $x_2=1$ állandóan fennáll.

Az y_4 fázis kezdetét a bal oldali ágon azért kellett felvennünk, mert az $x_1x_2=01$-re és az $x_1x_2x_3x_4=1011$-re vonatkozó műveletek nyilvánvalóan nem vonhatók össze a 4.7. ábra szerint. Az ellentétes x_1 értékekre várakozó műveletekre ugyanis a 4.5.*a)* ábra esete érvényes. Így új fázis felvétele nélkül nem tudnánk ellentmondásmentesen felírni z_3 és z_4 eltűnési feltételeinek algebrai kifejezését. A jobb oldali ágon az y_4 fázis kezdetét azért célszerű az $x_1x_2=10$-ra vonatkozó művelet után elhelyezni, mert így a bal oldali ágba történő becsatlakozási pont egyértelműen jellemezhető

mindkét ágon $x_1x_2=10$ fennállásával. Az y_4 fázis felvételét az indokolja, hogy az $x_1x_2x_3x_4=1011$-re, ill. az $x_1x_2=10$-ra és az $x_1=0$-ra várakozó műveletek nem teszik lehetővé a 4.7. ábra szerinti értelmezést, mert csak a 4.5*a)* ábra szerint képzelhetjük el a működést. Hasonló gondolatmenettel magyarázható a bal oldali ágon y_3 felvételének szükségessége $z_1{:}0$ szempontjából. A fázishatárok kijelölésére később még további szempontokat is megismerünk. A 4.23. ábrán kijelölt fázisok alapján az alábbi logikai függvények adódnak a φ_z és φ_y leképezések, vagyis a KH hálózat megvalósításaként:

$$
\begin{aligned}
z_1{:}1 &= y_1(x_1+\bar{x}_2\bar{x}_3),\\
z_1{:}0 &= y_2x_1\bar{x}_2+y_3x_1\bar{x}_2x_3x_4,\\
z_2{:}1 &= y_1\bar{x}_2\bar{x}_3\bar{x}_{200},\\
z_2{:}0 &= y_1\bar{x}_2\bar{x}_3x_{200} \rightarrow y_1x_{200},\\
z_3{:}1 &= y_1x_1\bar{x}_4+y_2(\bar{x}_1x_2+x_1\bar{x}_2),\\
z_3{:}0 &= y_1x_1x_4+y_4\bar{x}_1 \rightarrow y_1x_4+y_4\bar{x}_1,\\
z_4{:}1 &= y_1x_1x_4+y_1\bar{x}_2\bar{x}_3x_{200} \rightarrow y_1(x_4+x_{200}),\\
z_4{:}0 &= y_4\bar{x}_1,\\
Y_1 &= y_4\bar{x}_1,\\
Y_2 &= y_1(x_1x_2x_4+\bar{x}_2\bar{x}_3x_{200}) \rightarrow y_1(x_4+x_{200}),\\
Y_3 &= y_2\bar{x}_1x_2,\\
Y_4 &= y_2x_1\bar{x}_2+y_3x_1\bar{x}_2x_3x_4.
\end{aligned}
\tag{4.3.}
$$

A kifejezésekben természetesen mindig a szinkronizált bemeneti jeleknek megfelelő változókat kell értelmeznünk, annak ellenére, hogy erre nem alkalmaztunk külön jelöléseket. A továbbiakban csak akkor használunk megkülönböztető indexet, ha ezt a változások időbeli vizsgálata igényli.

Figyeljük meg, hogy $z_2{:}0$, $z_4{:}1$ és Y_2 kifejezésében az $\bar{x}_2\bar{x}_3$ szorzatot elhagyhattuk, mert $x_{200}=1$ fennállása önmagában elegendő az y_1 fázis jobb oldali ágának egyértelmű kijelöléséhez. Az x_{200} bemeneti jel ugyanis csak akkor vehet fel 1 értéket a 4.12. ábra alapján, ha $z_2=1$ megjelent, ami csak az y_1 fázis jobb oldali ágán haladva történhet meg $x_2x_3=00$ fellépése esetén. Az x_{200} bemeneti jel pótlólagos felvételével keletkező specifikációs bemeneti változás sorrendben követi az $x_1=1$-re és az $x_2x_3=00$-ra vonatkozó bemeneti műveletekkel jellemzett, egymással VAGY kapcsolatban levő specifikációs bemeneti változásokat. Ezért a vezérlési folyamatot csak az ábrázolt sorrendben vizsgálva, az $\bar{x}_2\bar{x}_3$ szorzatot elhagyhattuk az érintett függvények felírásakor.

A $z_2{:}1$ függvény kifejezésében az $\bar{x}_{200}$ tényezőt azért kellett felvennünk, mert enélkül $z_2{:}1$ és $z_2{:}0$ értéke egyidejűleg 1 lehetne $x_1x_2x_3=000$ fennállása esetén $x_{200}=1$ fellépésének hatására mindaddig, amíg a vezérlőegység át nem lép az y_2 fázisba. Így

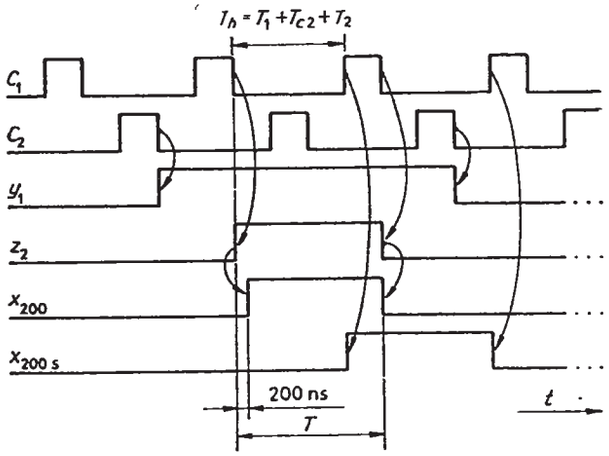
$x_{200}=1$ megjelenése után a z_2 kimeneti jelet előállító flip-flop S és R bemeneteire egyidejűleg 1 értékű jelek jutnának még az y_1 fázisban. Az $x_{200}=1$ megjelenésének hatására a C_2 impulzus lefutó éle létrehozná az $y_2=1$ értéket, ami viszont $z_2{:}1$ és $z_2{:}0$ értékét egyaránt 0-ra állítaná. Így a z_2 kimeneti jelet előállító flip-flop az $SR=11$ vezérlés után az $x_{200}=1$ megjelenését szinkronizáló C_1 impluzus lefutó élének pillanatában $SR=00$ vezérlést észlelne, ami a flip-flop belső áramköri késleltetéseitől függő kimeneti értéket hozna létre. Ezt a hibalehetőséget küszöbölhetjük ki azáltal, hogy az $\bar{x}_{200}$ tényezőt hozzáírjuk $z_2{:}1$ kifejezéséhez. Megfigyelhetjük, hogy a $z_3{:}0$, $z_4{:}1$ és Y_2 kifejezésekben elhagyhatjuk az x_2 tényezőt. Ennek az az oka, hogy az $x_2=1$-re és az $x_4=1$-re vonatkozó műveletek rendre a ΔX^{25} és ΔX^{56} egymást követő specifikációs bemeneti változásokat jellemzik. Mivel ezek sorrendje előírt, ezért elegendő csak az ábrázolt műveleti sorrendben képezni a feltételeket. A feltételezett X^2X^5 és X^6 kombinációk esetén ΔX^{25} során $x_2=1$ megjelenik, az ezt követő ΔX^{56} lejátszódásakor pedig $x_4=1$ lép fel, de $x_2=1$ is fennmarad. Így $x_4=1$ megjelenése önmagában képes jelezni a ΔX^{25} és a ΔX^{56} változások lejátszódását.

A $z_3{:}1$ kifejezés első tagjában azért van szükség az $\bar{x}_4$ tényezőre, mert enélkül az y_1 fázisban nem tudnánk megkülönböztetni z_3 megjelenésének és eltűnésének feltételeit.

A $z_3{:}0$, $z_4{:}1$ és Y_2 kifejezések első tagjából azért hagyhattuk el az x_1 tényezőt, mert a feltételezett specifikációs bemeneti változások mellett az y_1 fázisban csak a bal oldali ágon léphet fel az $x_4=1$ érték, tehát az $x_1=1$-gyel történő megkülönböztetés felesleges.

Láthatjuk, hogy a fázishatárok kijelölése az állapottáblával leírt sorrendi hálózatok esetében az állapot-összevonási eljárásnak felel meg. Tudjuk, hogy a nem teljesen határozott sorrendi hálózatok legegyszerűbb összevont állapottábláinak meghatározását nem tudtuk teljes egészében szisztematikus módszerrel végezni. A specifikációs bemeneti változások alapján bevezetett folyamatábra által leírt működés nyilvánvalóan nem teljesen határozott. Ezért a fázishatárok legkedvezőbb kijelöléséhez is szükség van próbálgatásra és találékonyságra.

Sokszor az egy fázisba kerülő ágak könnyű megkülönböztethetősége is döntő lehet a fázishatárok kijelölésében. Esetenként egyes kimeneti jelek fennálló értékét is felhasználhatjuk az egyes ágak megkülönböztetésére. Ha ezt a megoldást választjuk, akkor az egyes függvények felírásakor a kimeneti jeleknek megfelelő változókat is lényegében független változóknak tekintünk. Emiatt ügyelnünk kell arra, hogy az ilyen kimeneti jelek által befolyásolt $z_i{:}1$ és $z_i{:}0$ függvények szempontjából a C_1 óraimpulzus lefutó élének hatására funkcionális hazárd alakulhat ki. Ez az él végzi ugyanis a kimeneti flip-flopok mintavételét és billentését, így egyúttal a lényegében bemeneti jeleknek tekintett visszavezetett kimeneti jeleket is változtatja. Emiatt nem érvényesül a 4.22. ábra szerinti ütemezés hazárdmentesítő hatása. A hibát kiküszöbölhetjük, ha a visszavezetett kimeneti jeleket a bemeneti jelekhez hasonlóan C_1 felfutó élére működő bemeneti szinkronizáló flip-flopokon vezetjük keresztül a $z_i{:}1$ és $z{:}0$ függvények előállításához. Az Y_i függvények megvalósításakor ilyen hiba nem lép fel,



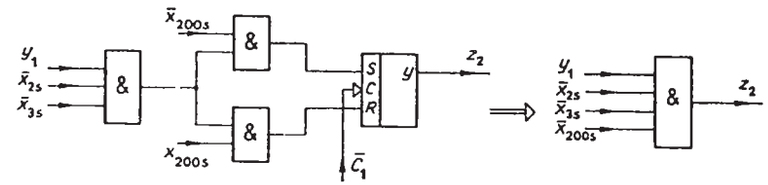
4.24. ábra. A kétfázisú órajel, valamint a bemeneti és kimeneti szinkronizálás hatásának szemléltetése a kimeneti impulzus előállítására. Az időmérőegységtől függetlenül a kimeneti impulzus időtartama nem lehet rövidebb T-nél

mert C_1 lefutó éle és C_2 felfutó éle között feltételezésünk szerint befejeződnek a tranziens jelenségek.

Ha az egy fázison belüli ágak megkülönböztetésére nem találunk megoldást, akkor áganként külön fázisok kijelölésére van szükség. Az áganként külön fázisok esetén az elágazást okozó bemeneti jelnek csak addig kell állandónak lennie, amíg értékétől függően az adott ág fázisa létre nem jön. A további működés során az ágak megkülönböztetését a létrejött fázis, azaz y kombináció alapján végezhetjük, amely az elágazást okozó bemeneti jel időközbeni megváltozásától független.

Vizsgáljuk meg ezek után, hogy példánkban az előírt szélességű kimeneti impulzus milyen időzítés szerint játszódik le. A z_2 kimeneti jellel kapcsolatos függvények az előírt szélességű impulzus megjelenésének és eltűnésének feltételeit egy fázison (y_1) belül fogalmazzák meg. A felírt függvények alapján a bemeneti és kimeneti szinkronizálást, valamint a 4.12. ábrán vázolt jelváltozásokat feltételezve a 4.24. ábra idődiagramján szemléltettük a z_2 kimeneti impulzus lejátszódását. A jelváltozásokat $z_2=1$ megjelenésétől kezdődően ábrázoltuk. Az ábrán x_{200s}-sel jelöltük az x_{200} jelhez tartozó bemeneti szinkronizáló flip-flop kimeneti jelét, amely a KH hálózatra jut.

Megfigyelhetjük, hogy a 4.12. ábrán bemutatott párbeszéd jellegű jelváltozás ezúttal az órajel ütemezésével játszódik le. A z_2 kimeneti jel ugyanis csak az x_{200s} jel közvetítésével vesz tudomást az x_{200} jel megváltozásáról. Láthatjuk, hogy ennek az a következménye, hogy az időmérőegységtől függetlenül a kimeneti impulzus időtartama nem lehet rövidebb az órajel periódusidejénél (T). Könnyen beláthatjuk, hogy a kimeneti impulzusok időtartama jó közelítéssel csak az órajelek periódusidejének egész számú többszöröse lehet. Elegendő ehhez a 4.24. ábrán vázolt idődiagramot elképzelnünk arra az esetre, ha $T_h<200$ ns. Ekkor ugyanis a $z_2=1$ megjelenését követő első C_1 impluzus felfutó éle még $x_{200}=0$-ból vesz mintát, vagyis



4.25. ábra. A Z_2 kimeneti jelet előállító S—R flip-flop feleslegességének szemléltetése a 4.23. ábrából kiolvasott függvények alapján

$x_{200s}=0$ áll fenn mindaddig, amíg valamelyik C_1 impulzus felfutó éle $x_{200}=1$-et nem észlel.

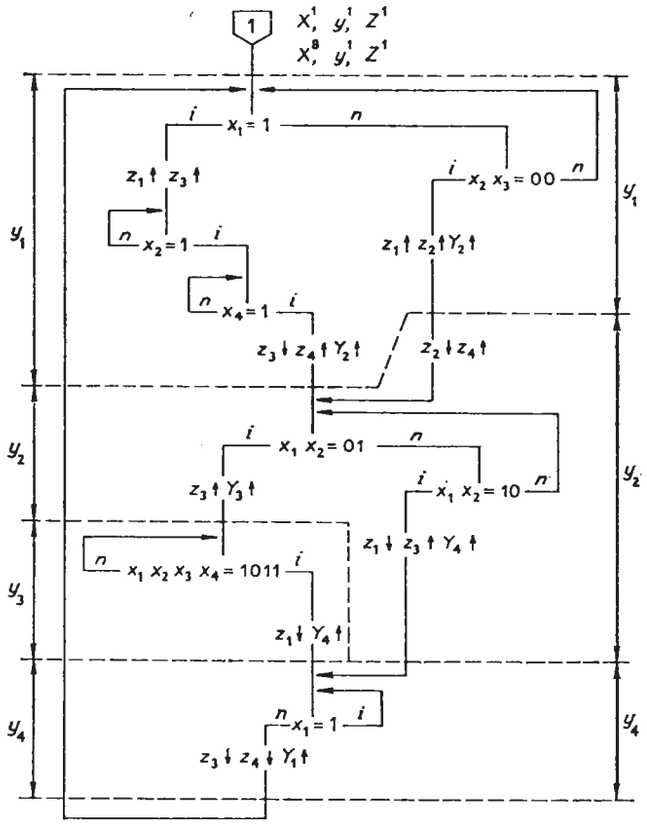
A z_2 kimeneti jelre felírt függvények alapján azt is megállapíthatjuk, hogy ezt a jelet kimeneti flip-flop nélkül is előállíthatjuk. Ezt szemlélteti a 4.25. ábra vázlata, amelyen a felírt függvények alapján rajzoltuk fel a z_2-t előállító kimeneti flip-flop vezérlését. Láthatjuk, hogy $y_1=1$ és $x_2x_3=00$ esetén a flip-flop állapota x_{200} mindenkori értékének negáltja. Ezért az S—R flip-flopot elhagyhatjuk és z_2 értékét közvetlenül előállíthatjuk a KH hálózat kimenetén:

$$z_2 = y_1\bar{x}_{2s}\bar{x}_{3s}\bar{x}_{200s}.$$

A szinkronizációra vonatkozó s indexet az eddigiekhez hasonlóan elhagyhatjuk, hiszen a bemeneti szinkronizáció ténye a szinkron fázisregiszteres vezérlőegység bemutatott felépítéséből következik. Eddig az egyszerűség érdekében a felírt kifejezésekben nem jelöltük, hogy a változóknak megfelelő jelek minden esetben szinkronizáltak. Nem szabad azonban megfeledkeznünk arról, hogy a kimeneti flip-flop elhagyása miatt nem érvényesül a C_1 órajel lefutó élének (4.22. ábra) hazárdmentesítő hatása. Az érintett kimeneteken a bemeneti és a szekunder változások idején így funkcionális hazárd léphet fel, ami miatt sok esetben nem éri meg megtakarítani a kimeneti flip-flopot. A 4.24. ábra alapján könnyen elképzelhetjük a kimeneti flip-flop nélkül előállított z_2 impulzus szélességének kialakulását is. Nyilvánvaló, hogy így előállíthatók az órajel periódusidejénél rövidebb impulzusok is.

Az eddigiek alapján megállapíthatjuk, hogy a kimeneti impulzus időtartamát a bemeneti és kimeneti szinkronizálás miatt nem csak az időmérőegység határozza meg. Ha azonban az x_{200} jelet szinkronizálás nélkül juttatjuk a KH hálózatra, a z_2 kimeneti jelet pedig aszinkron S—R flip-floppal állítjuk elő, akkor a kialakuló z_2 impulzus időtartama nyilvánvalóan kizárólag az időmérőegység kimeneti jelének megjelenésétől függ. A bemeneti és kimeneti szinkronizáció elmaradása az időmérőegységgel kapcsolatos jelek esetében a 4.12. ábrán bemutatott párbeszéd jellegű működés következtében nem okoz hibás működést az n-ből 1 kódú állapotok szempontjából. A kimeneti szinkronizáció elmaradása miatt azonban ügyelnünk kell a z_2 kimeneten lehetséges funkcionális hazárdra.

Ha az előállítandó kimeneti impulzus időtartamára nincs különösebb megkötés és megfelelő a periódusidő (T), ill. annak egész számú többszöröse impulzus-időtartam-

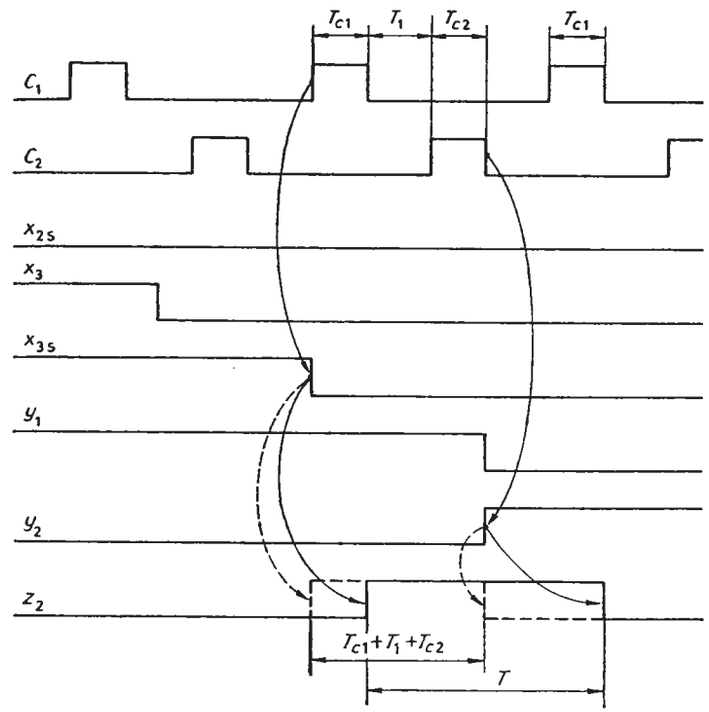


4.26. ábra. A 4.23. ábra módosítása az időmérő bemenet (x_{200}) elhagyásával és a fázishatár megfelelő felvételével a kimeneti impulzus előállítása céljából

ként, akkor az időmérőegységre nincs is szükség, vagyis felesleges a 4.12. ábrán bemutatott megoldás alapján módosítani a folyamatábrát. Sokkal egyszerűbb ugyanis, ha a 4.26. ábrán vázolt módon a kimeneti jel megjelenését és eltüntetését okozó műveletek közé fázishatárt veszünk fel.

Az ábra alapján az alábbi függvényeket írhatjuk fel:

$$z_1{:}1 = y_1(x_1+\bar{x}_2\bar{x}_3),$$
$$z_1{:}0 = y_2x_1\bar{x}_2+y_3x_1\bar{x}_2x_3x_4,$$
$$z_2{:}1 = y_1\bar{x}_2\bar{x}_3,$$
$$z_2{:}0 = y_2,$$
$$z_3{:}1 = y_1x_1\bar{x}_4+y_2(\bar{x}_1x_2+x_1\bar{x}_2),$$
$$z_3{:}0 = y_1x_4+y_4\bar{x}_1,$$
$$z_4{:}1 = y_1x_4+y_2\bar{x}_2\bar{x}_3,$$
$$z_4{:}0 = y_4\bar{x}_1,$$



4.27. ábra. A Z_2 kimeneti impulzus kialakulásának szemléltetése a 4.26. ábra szerinti fáziskijelölés esetén. A kimeneti impulzus időtartama nem lehet rövidebb az órajel periódusidejénél

$$Y_1 = y_4\bar{x}_1,$$
$$Y_2 = y_1(x_4+\bar{x}_2\bar{x}_3),$$
$$Y_3 = y_2\bar{x}_1x_2,$$
$$Y_4 = y_2x_1\bar{x}_2+y_3x_1\bar{x}_2x_3x_4.$$

Figyeljük meg, hogy $z_2{:}0$ kifejezésében azért elegendő csupán y_2 figyelembevétele, mert nem okoz hibás működést az, hogy z_2 értékének „0"-ra állítása nem csak az y_2 fázis kezdetének a jobb oldali, hanem a bal oldali ágán is megtörténik.

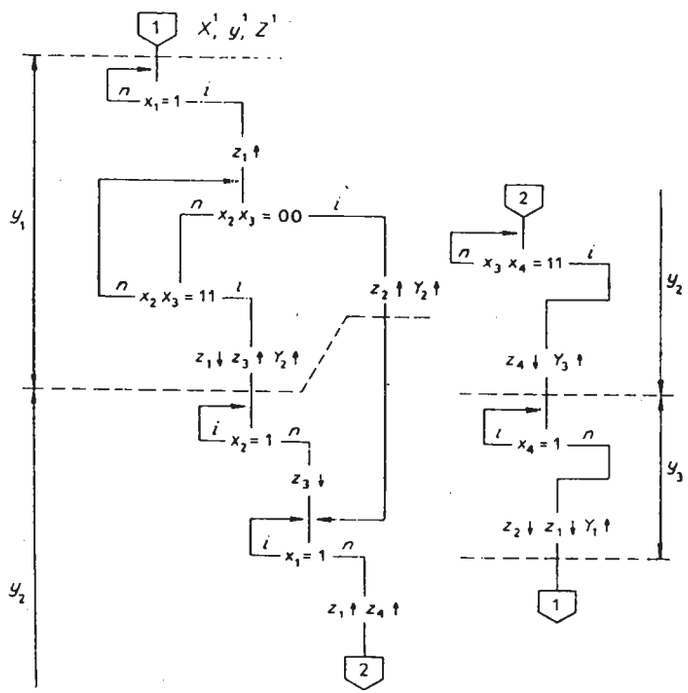
A bal oldali ágon ugyanis $z_2=1$ nem léphet fel, tehát 0 értékre állítása nem változtatja meg a folyamatábrával leírt működést. A z_2 kimeneti impulzus hosszának kialakulását szemlélteti a 4.27. ábra idődiagramja. Láthatjuk, hogy a kimeneti impulzus időtartama nem lehet rövidebb az órajel periódusidejénél, ha a jelkésleltető hatásoktól eltekintünk. Az impulzusszélességet T_{C1}-re vagy T_{C2}-re csökkenthetjük, ha a 4.27. ábra szerint előállított z_2 jelet ÉS kapcsolatba hozzuk a C_1 vagy a C_2 órajellel. Így egyetlen C_1 vagy C_2 impulzust kapuzhatunk a kimenetre.

Hosszabb impulzust úgy állíthatunk elő, hogy a $z_2=1$ és a $z_2=0$ műveletek közé több egymás utáni fázishatárt veszünk fel. Minden újabb felvett fázishatár egy-egy órajel-periódusnyi idővel hosszabbítja meg a kimeneti impulzus időtartamát.

A 4.27. ábra és a felírt $z_2{:}1$, $z_2{:}0$ függvények alapján azt is megállapíthatjuk, hogy a z_2 impluzust ezúttal is előállíthatjuk kimeneti S—R flip-flop nélkül. A z_2 jel ugyanis a megjelenését követően feltétel nélkül bekövetkező fázisváltozás (y_1-ről y_2-re) hatására feltétel nélkül eltűnik. Így felírhatjuk, hogy z_2 csak $y_1=1$ tartama alatt áll fenn $x_2x_3=00$ esetén:

$$z_2 = y_1\bar{x}_2\bar{x}_3.$$

Ilyenkor azonban z_2 változásai a kimeneti flip-flop elmaradása miatt nincsenek szinkronizálva a C_1 óraimpulzus lefutó éleivel, hanem a bemeneti és a szekunder változásokat szinkronizáló élek hatására közvetlenül jönnek létre. A 4.27. ábrán szaggatottan jelöltük ezeket a változásokat. Megállapíthatjuk, hogy az így kialakuló z_2 impulzus szélessége T helyett $T_{C1}+T_1+T_{C2}$.



4.28. ábra. A 4.10. ábrán szereplő párbeszéd jellegű folyamatábra fázishatárainak kijelölése azzal a feltételezéssel, hogy az egymás utáni várakozóműveletekben a továbbhaladáshoz szükséges jelértékek a mintavételi időpontban (C_1 felfutó éle) még átfedik egymást

A kimeneti impulzus szélességét tág határok között állíthatjuk be, ha ún. monostabil elemet alkalmazunk, amelynek működését a z_2 jel megváltozása indíthatja el.

Térjünk vissza ezek után a fázishatárok kijelölésének szempontjaira. A szükséges fázisok száma általában csökken, ha az egymás utáni bemeneti műveletekben a továbbhaladást előidéző bemeneti jelértékekről, ill. kombinációkról feltételezhetjük, hogy a mintavételi időpontban (C_1 felfutó éle) még átfedik egymást. Láttuk, hogy ilyen esetben a függvények felírásakor feltételezhetjük a 4.7. ábra szerinti átrajzolási lehetőséget, miáltal az érintett bemeneti műveletek közé nem szükséges fázishatárt felvenni. Ez alól természetesen kivétel az olyan eset, amelyben az egymás utáni bemeneti műveletek azonos bemeneti jelek ellentétes értékeire vonatkoznak.

A 4.28. ábrán a fenti feltételezéssel bemutattuk a 4.10. ábrán szereplő párbeszéd jellegű folyamatábra fázishatárainak kijelölését. Ha a működés párbeszéd jellegű, akkor még azt is feltételezhetjük, hogy a bemeneti változások mindig a megelőző kimeneti változás következményei. Így a kimeneti jelet állító műveletek végrehajtásakor mindig érvényesek azok a bemeneti jelértékek, amelyek a szóban forgó kimeneti műveletet követő bemeneti műveletekben várakozást okoznak. A párbeszéd jellegből ugyanis következik, hogy a továbbhaladáshoz szükséges bemeneti jelértékek csak a kimeneti változások után alakulnak ki. Így például a 4.28. ábra y_1 fázisában a $z_1=1$ értéket beállító művelet végrehajtásakor feltételezhetjük, hogy $x_2x_3=11$ nem áll fenn. A 4.28. ábra alapján az alábbi függvényeket írhatjuk fel:

$$\begin{aligned}
z_1{:}1 &= y_1x_1(\overline{x_2x_3})+y_2\bar{x}_1 = y_1x_1(\bar{x}_2+\bar{x}_3)+y_2\bar{x}_1,\\
z_1{:}0 &= y_1x_2x_3+y_3\bar{x}_4,\\
z_2{:}1 &= y_1\bar{x}_2\bar{x}_3,\\
z_2{:}0 &= y_3\bar{x}_4,\\
z_3{:}1 &= y_1x_2x_3,\\
z_3{:}0 &= y_2\bar{x}_2,\\
z_4{:}1 &= y_2\bar{x}_1(\overline{x_3x_4}) = y_2\bar{x}_1(\bar{x}_3+\bar{x}_4),\\
z_4{:}0 &= y_2x_3x_4,\\
Y_1 &= y_3\bar{x}_4,\\
Y_2 &= y_1(\bar{x}_2\bar{x}_3+x_2x_3),\\
Y_3 &= y_2x_3x_4.
\end{aligned} \tag{4.4.}$$

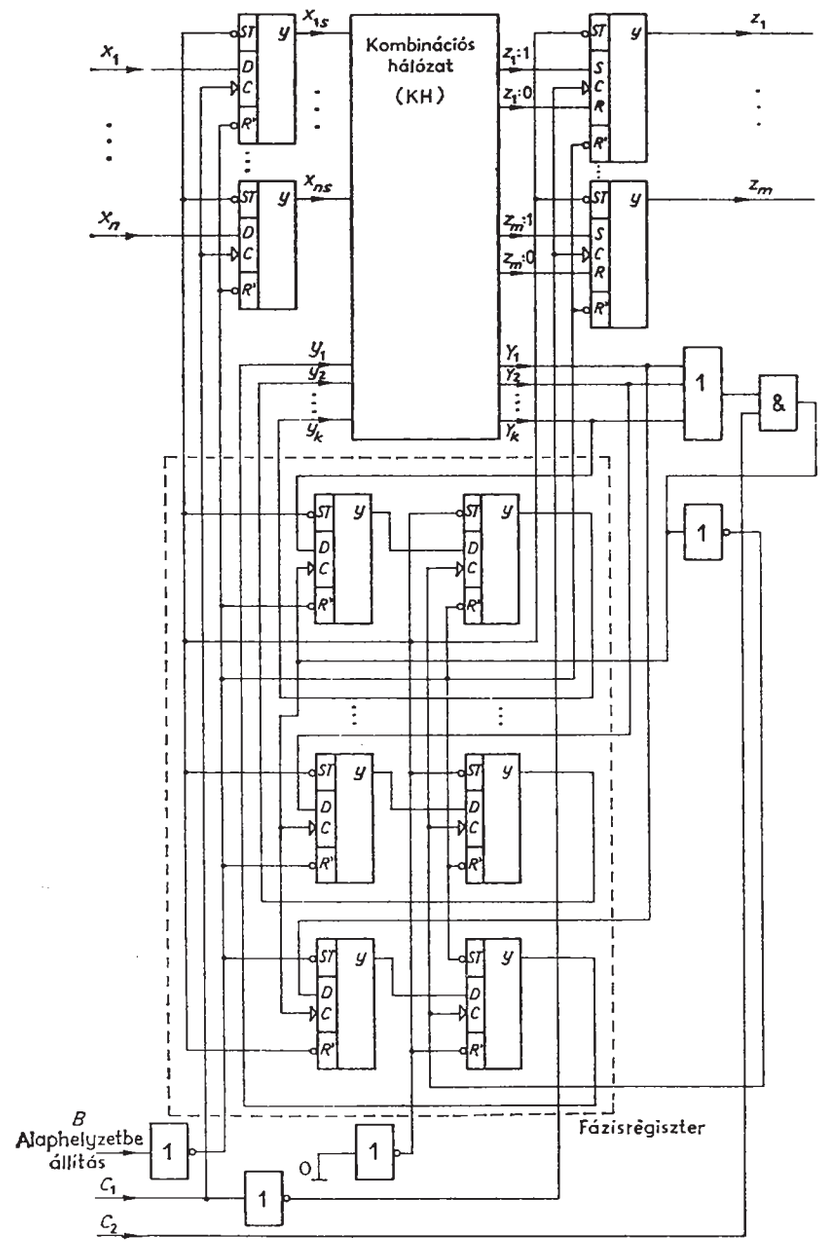
Figyeljük meg, hogy például $z_1{:}1$ kifejezésének első tagjában figyelembe kellett vennünk az $\overline{x_2x_3}$ tényezőt, mert enélkül az y_1 fázis bal oldali ágán haladva z_1 megjelenésének és eltűnésének feltételei egyidejűleg $z_1{:}1=1$ és $z_1{:}0=1$ értékeket szolgáltatnának. Ugyanilyen okból kellett $z_4{:}1$ kifejezésében az $\overline{x_3x_4}$ tényezőt felírni.

Látható, hogy ezeknek a pótlólagos tényezőknek a figyelembevételével lényegében fázisokat takarítunk meg, kihasználva a párbeszéd jellegből következő kapcsolatot a bemeneti és kimeneti változások között.

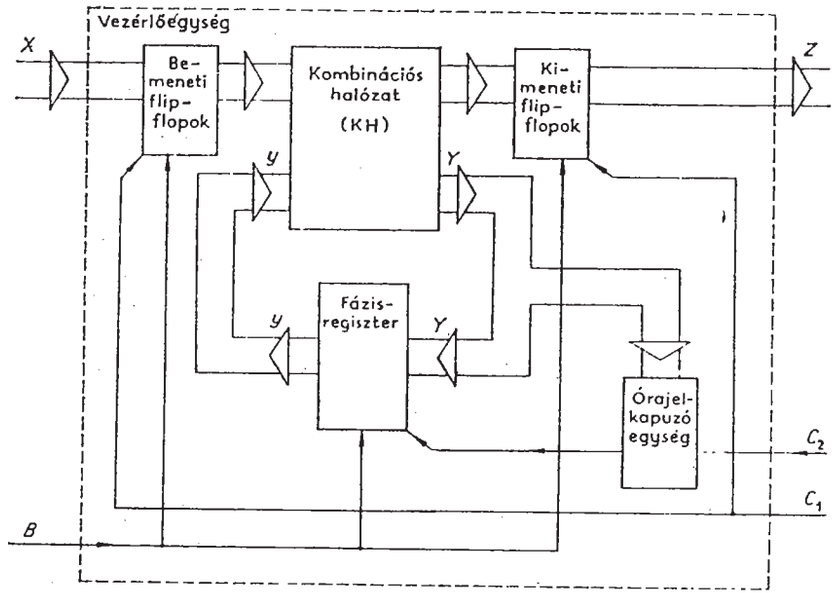
Beláthatjuk, hogy ha a működés nem párbeszéd jellegű, akkor a pótlólagos tényе-

zők alkalmazásának bemutatott módja a 4.28. ábra szerinti fáziskijelölés esetén nem ad egyértelmű megoldást. A várakozóműveletek előtti bemeneti jelértékeket ugyanis nem mindig ismerjük a specifikációs gráf nélkül megszerkesztett folyamatábrák esetén. Ilyenkor csak pótlólagos fázisok felvételével kaphatunk helyes megoldást. A z_3:0 kifejezés felírásakor nem különböztettük meg az y_2 fázisban levő két ágat. Így a $z_3=0$ értéket beállító műveletet mindkét ág mentén haladva végrehajtja a vezérlőegység. Ez hibás működést nem okoz, mert a jobb oldali ág mentén z_3 nem jelenik meg, tehát a $z_3=0$-t beállító művelet felesleges végrehajtása megengedhető. Megfigyelhetjük, hogy a két ág megkülönböztetése új fázis felvételét igényelné, mert a két ágat létrehozó y_1-beli várakozóműveletek és a $z_3=0$ műveletet y_2-ben megelőző várakozóművelet egyaránt kapcsolatos az x_2 bemeneti jellel. Emiatt ezeket az egymás után következő várakozóműveleteket nem értelmezhetjük a 4.7. ábrán vázolt módon, vagyis nem írhatnánk fel ellentmondásmentesen olyan logikai függvényt, amely kizárólag a bemeneti változókonértelmezve alkalmas volna az y_2-beli két ág megkülönböztetésére. A kapott függvényeket mindig célszerű megvizsgálni a kimeneti flip-flopok elhagyhatósága szempontjából. Nem szabad azonban megfeledkeznünk a kimeneti flip-flop nélkül előállított kimeneteken jelentkező funkcionális hazárdról. Ha a 4.15. ábrán bemutatott felépítési vázlaton elvégezzük az órajelkapuzás, valamint a kétfázisú órajellel történő bemeneti és kimeneti szinkronizáció miatti módosításokat, akkor a szinkron fázisregiszteres vezérlőegység általános felépítési vázlatához jutunk, amelyet a 4.29. ábrán szemléltetünk. Az ábrán az alaphelyzetbe állítás módja is követhető, amelynek célja, hogy a működés kezdetekor a vezérlőegység a folyamatábra kiindulási pontjáról induljon.

A 3.12. fejezetben foglalkoztunk az alaphelyzetbe állító impulzus előállítási módjaival és szerepével. A 4.29. ábrán vázolt szinkron fázisregiszteres vezérlőegységben az alaphelyzetbe állításhoz feltétlenül szükséges, hogy a működés a folyamatábra kiindulási pontját tartalmazó ún. *kiindulási fázisból* induljon. Így a visszacsatolóágban levő reteszelt működésű flip-flopok közül azt kell 1 állapotba hozni, amely a kiindulási fázisnak megfelelő szekunder változót valósítja meg. (Az eddigi folyamatábrákon y_1-et.) Ezt a legegyszerűbben az ismert funkciójú Preset (ST) és Clear (R′) bemenetek megfelelő vezérlésével oldhatjuk meg. A 4.29. ábrán megfigyelhetjük, hogy az alaphelyzetbe állandó impulzus (B) a kiindulási fázis flip-flopját 1 állapotba, a többiét pedig 0 állapotba állítja. Az ábrán figyelembe vettük, hogy a Preset (ST) és a Clear (R′) bemenetek a legtöbb flip-flop esetén tagadott jelekre hatásosak. Az ábrán a nem használt Preset, illetve Clear bemenetekre egy 0-ra kötött bemenetű INVERTER kimenetéről juttattunk 1 értéket. A kiindulási fázis beállításán túlmenően a kiindulási pontból való induláshoz adott bemeneti kombináció beállítása is szükséges. A 4.29. ábrán feltételeztük, hogy az összes bemeneti és kimeneti jelet kiinduláskor 0 értékre kell állítani. Ezért az alaphelyzetbe állító impulzussal a bemeneti és a kimeneti szinkronizáló flip-flopok Clear bemeneteit vezéreltük, de nyilvánvalóan bármilyen kiindulási bemeneti és kimeneti kombinációt beállíthatunk a B impulzussal a flip-flopok Preset és Clear bemeneteit megfelelően vezérelve.



4.29. ábra. A szinkron fázisregiszteres vezérlőegység általános felépítése



4.30. ábra. A szinkron fázisregiszteres vezérlőegység összefoglaló rendszertechnikai vázlata

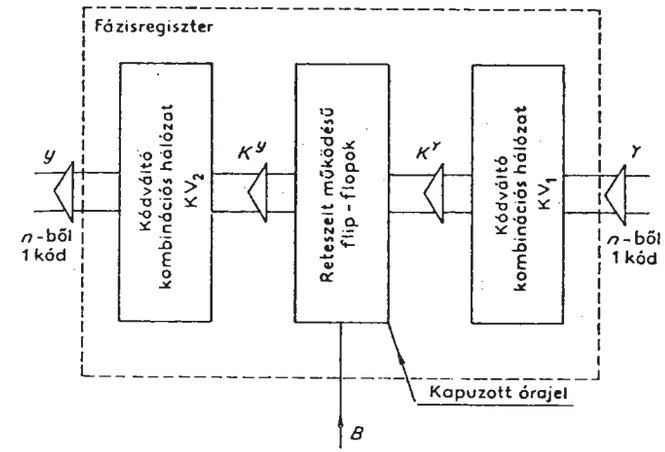
A működés megfelelő elindításához általában az is szükséges lehet, hogy a kétfázisú órajel C_1 és C_2 impulzusai a 4.21. ábra szerinti sorrendben induljanak. Így a B impulzusnak valamilyen módon hatnia kell az órajel-generátorra is.

A visszacsatolóágban levő reteszelt működésű flip-flopok összességét fázisregiszternek neveztük, amelyet a 4.29. ábrán szaggatottan körülhatároltunk. Ennek megfelelően a 4.30. ábrán látható rendszertechnikai felépítés foglalja össze a szinkron fázisregiszteres vezérlőegység belső részegységeit.

Összefoglalva megállapíthatjuk, hogy a szinkron fázisregiszteres vezérlőegység tervezése lényegében három lépésből áll:

— a folyamatábra megszerkesztése;
— a fázishatárok kijelölése a folyamatábrán;
— a KH hálózat kimeneti függvényeinek felírása a folyamatábra alapján.

Láttuk, hogy a 4.29., ill. 4.30. ábra szerinti felépítés biztosítja az n-ből 1 jellegű állapotkódokat, ami lehetővé teszi a KH hálózat kimeneti függvényeinek közvetlen kiolvasását a folyamatábrából. A fázisregisztert alkotó flip-flopok számát csökkenthetjük, ha azt a 4.31. ábra szerint építjük fel. Az így létrejövő fázisregiszter bemenetén és kimenetén továbbra is n-ből 1 kód jelenik meg. A bemeneti kódváltó kombinációs hálózat a mindenkori n-ből 1 kódú Y kombinációhoz olyan kódú (pl. bináris) kombinációt rendel hozzá, amely a lehető legkevesebb helyértékkel különbözteti meg az összes Y kombinációt. Így a lehető legkevesebb reteszelt működésű flip-flopra



4.31. ábra. Kódolt fázisregiszter rendszertechnikai vázlata

van szükségünk. A flip-flopok kimenetére kapcsolódó KV_2 kódváltó kombinációs hálózat visszaállítja az n-ből 1 kódot az y kombinációban. Így az ún. *kódolt fázisregiszter* környezete nem is vesz tudomást a 4.29. ábrához képesti eltérő felépítésről. A kétfázisú órajellel történő szinkronizáció miatt a kódváltáskor keletkező funkcionális hazárdnak nincs káros hatása. A kimeneti flip-flopok esetleges elhagyásakor azonban ügyelnünk kell arra, hogy a kódolt fázisregiszter esetén az y kombinációkban jelentkező funkcionális hazárd közvetlenül éreztetheti hatását a flip-flop nélküli kimeneti jelek változásaiban.

A kódolt fázisregiszterben az egymás mellett függetlenül működő flip-flopok helyett alkalmazhatunk flip-flopokból felépített szinkron sorrendi hálózatokat. Ilyenek az ún. *számlálók* és az ún. *léptetőregiszterek*. A számlálók a bemenetükre érkezett impulzusok számát ábrázolják valamilyen kódban a kimenetükön. Működésük úgy is vezérelhető, hogy számlálás helyett adott kimeneti kombinációt hozzanak létre az ún. *párhuzamos bemeneti jelek* értékétől függően (párhuzamos beírás). Az impulzusszámláló bemenetre a kapuzott órajelet, a párhuzamos bemenetekre pedig a 4.31. ábrán szereplő K^Y-jeleket vezethetjük. Így a folyamatábra egymás utáni, elágazás nélküli fázisváltozásait a kapuzott órajel egyedül vezérelheti a számláló jellegű működés miatt. A KV_1 hálózatnak csak az elágazások esetében szükséges fázisváltozások megvalósításában van szerepe a párhuzamos beírás lebonyolítása révén. Ha tehát a folyamatábrán kevés az elágazás, akkor a KV_1 hálózatnak kevés Y kombinációt kell a K^Y kombinációkkal megkülönböztetnie, vagyis a KV_1 hálózat egyszerű felépítésű.

A léptetőregiszterek a kódolt fázisregiszterben történő felhasználás szempontjából csak annyiban különböznek a számlálóktól, hogy számlálóbemenet helyett léptetőbemenetük van. Az ide érkező impulzus a léptetőregiszter kimenetén jelenlevő mindenkori bináris kombinációt egy helyértékkel lépteti jobbra vagy balra. Így a

léptetőbemenetre kapcsolt kapuzott órajel ezúttal is létrehozhatja a folyamatábra egymás utáni, eiágazás nélküli fázisváltozásait. A párhuzamos bemenetekre érkező K^Y kombinációknak tehát szintén csak az elágazások során kialakuló fázisokat kell megkülönböztetniük. Ha a léptetőregiszterben az egyes fázisoknak n-ből 1 kódú kombinációkat feleltetünk meg, akkor KV_2 természetesen elhagyható, de ilyenkor a regiszter szükséges flip-flopjainak száma sem csökken a kódolás nélküli esethez képest.

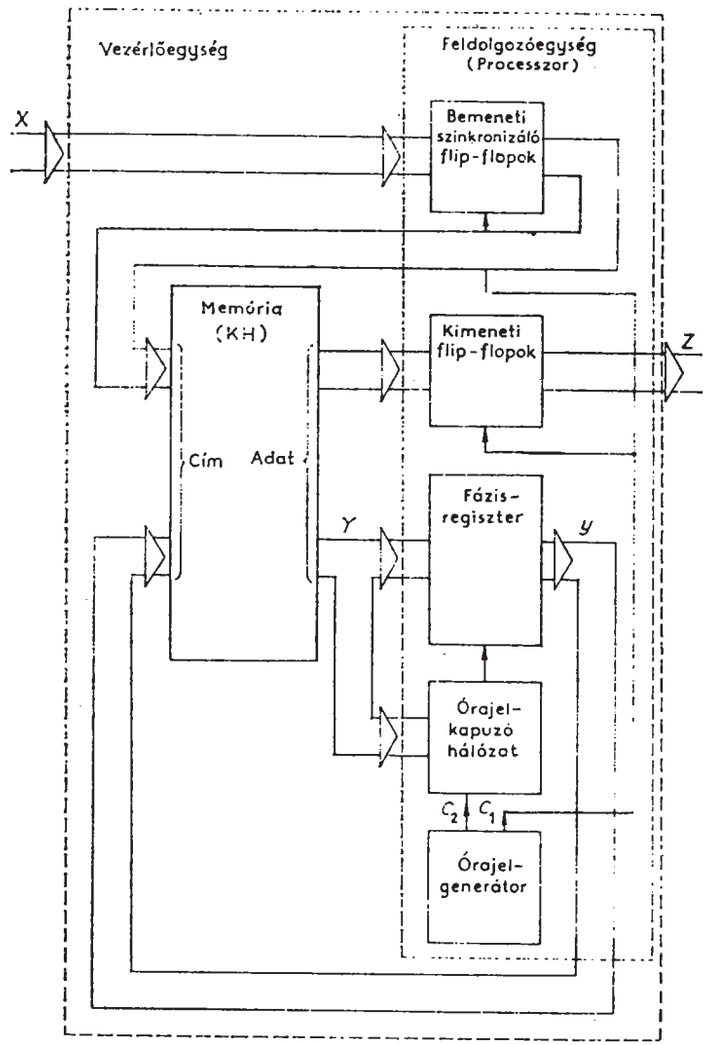
Mind számláló, mind léptetőregiszter alkalmazása esetén gondoskodni kell a működés vezérlésének megfelelő átállításáról a számláló, ill. léptető üzemmódból párhuzamos beíró üzemmódba és viszont. Ezt a feladatot ilyenkor a KV_1 hálózat kiegészítésével oldhatjuk meg, ami csökkenti a KV_1 hálózat említett egyszerűsödéséből származó előnyöket.

Ha a kódolt fázisregiszterben alkalmazott sorrendi hálózat működése nem követi a reteszelt működés előírásait (pl. nem reteszelt flip-flopokból épül fel), akkor a 4.29. ábrán vázolt megoldás nyilvánvalóan nem szavatolja a teljes vezérlőegység hazárdmentességét. Ilyen sorrendi hálózatokat csak akkor alkalmazhatunk, ha megvizsgáljuk és kiküszöböljük mindazokat a hibalehetőségeket, amelyek alapján reteszelt flip-flopok esetén a 4.29. ábrán vázolt felépítéshez jutottunk.

Befejezésül példaképpen a 4.32. ábrán felrajzoltuk a 4.28. ábrán adott folyamatábra szerinti működést megvalósító szinkron fázisregiszteres vezérlőegység elvi logikai rajzát a (4.4.) egyenletek alapján. Az ábrán feltételeztük, hogy a bemeneti szinkronizálást végző élvezérelt D flip-flopoknak tagadott kimenetük is van.

4.3. Mikroprogramozott vezérlőegységek tervezése

Láttuk, hogy a szinkron és aszinkron fázisregiszteres vezérlőegységek kombinációs hálózatait leíró logikai függvényeket közvetlenül kiolvashatjuk a folyamatábrából. Az így kapott logikai függvények alapján a kombinációs hálózatot az ismert módon megvalósíthatjuk memóriával vagy PLA elemekkel is. Például ha a szinkron fázis-

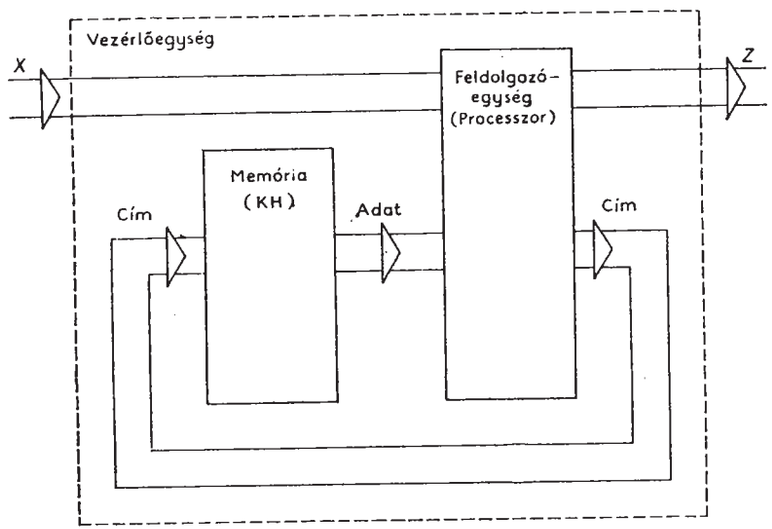


4.59. ábra. A szinkron fázisregiszteres vezérlőegység felépítésének értelmezése memóriával megvalósított kombinációs hálózatrész esetén

regiszteres vezérlőegység KH kombinációs hálózatát memóriaelemekből építjük fel, akkor az így kialakuló vezérlőegység a memórián kívül csak az általános, a megvalósítandó folyamatábrától függetlenül mindig szükséges fázisregisztert, az órajelkapuzó-hálózatot, valamint a bemeneti és kimeneti flip-flopokat tartalmazza. A felépítés ilyen értelmezését szemlélteti a 4.59. ábra. A kétfázisú órajelet (C_1, C_2) előállító ún. órajel-generátort is beleérthetjük az általánosan szükséges hálózatok közé. A vezérlőegységnek így kialakuló szaggatottan körülhatárolt, memórián kívüli része lényegében a memóriából jövő adatok feldolgozása révén állítja elő a kimeneti jeleket és az adatok eléréséhez szükséges memóriacímeket. Ezért nevezhetjük a 4.59. ábrán szaggatottan körülhatárolt hálózatok összességét *feldolgozóegységnek* vagy *processzornak*. A 4.59. ábrán bemutatott értelmezés szerinti egyszerűsített ábrázolással a 4.60. ábrán szereplő felépítéshez jutunk.

Megállapíthatjuk, hogy a processzor lényegében minden vezérlőegységben azonos lehet, mert a megvalósítandó folyamatábra csak a memóriatartalmat befolyásolja. Ez nyilvánvaló, hiszen a szinkron fázisregiszteres vezérlőegységben is csak a KH kombinációs hálózatot határozza meg a folyamatábra.

A folyamatábra fázisainak, valamint a bemeneti és kimeneti jeleknek száma természetesen megszabja a processzoron belül értelmezett fázisregiszter elemeinek, valamint a bemeneti és kimeneti flip-flopok számát. Ez azonban különböző folyamatábrák esetén nem jelent minőségi változást a processzorban, így vizsgálatunkban különböző folyamatábrák megvalósítására azonos felépítésű, ún. *uniformizált* processzort tételezhetünk fel. Ezek szerint a memóriatartalom módosítása révén változatlan processzorral különböző vezérlési folyamatábrákat valósíthatunk meg.



4.60. ábra. A 4.59. ábra értelmezése szerinti felépítés egyszerűsített ábrázolása

Könnyen beláthatjuk, hogy az aszinkron fázisregiszteres felépítés esetén is hasonló gondolatmenettel különíthetjük el az uniformizált processzort.

A továbbiakban bemutatjuk, hogy a 4.60. ábra szerinti felépítéshez a fázisregiszteres tervezési lépések nélkül, közvetlenül a folyamatábra alapján is eljuthatunk és az így kiadódó processzor is uniformizáltnak tekinthető.

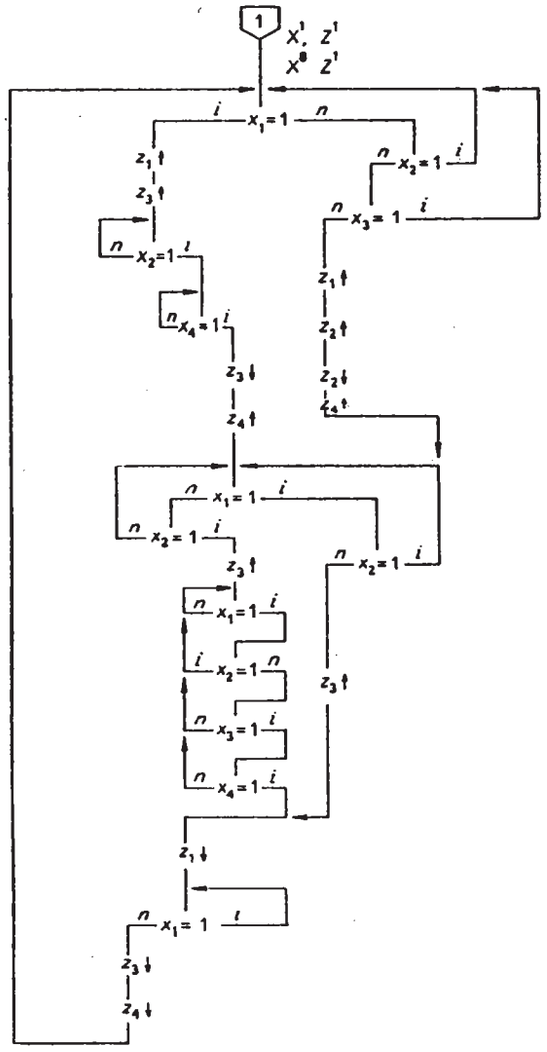
4.3.1. A folyamatábra elemi műveleteinek ábrázolása a memóriatartalomban

Bonstuk fel a folyamatábra bemeneti és kimeneti műveleteit ún. *elemi műveletekre*, amelyek mindegyike csak egyetlen bemeneti jel értékének vizsgálatát, ill. csak egyetlen kimeneti jel értékének beállítását végzi. A 4.61. ábrán ennek megfelelően rajzoltuk fel a 4.26. ábrán szereplő folyamatábrát. A szekunder változásokat nem tüntettük fel a 4.61. ábrán, mert a bemutatandó eljárás szempontjából lényegtelenek. A kimeneti műveletek jelenkénti egymás utáni végrehajtása néha összevonásokat tesz lehetővé a folyamatábra csomópontjainak formai áthelyezése révén. Így a 4.26. ábrán az y_4 fázis kezdetén levő csomópontot a 4.61. ábrán áthelyeztük az azt megelőző $z_1=0$-t beállító művelet elé. Ezáltal ezt a műveletet a jobb oldali becsatlakozó ágról elhagyhattuk. Megfigyelhetjük, hogy a több jellel kapcsolatos műveleteket egymás utáni elemi műveletekből állítottuk össze. Az egyes felbontott műveleteket alkotó elemi műveletek sorrendje nyilvánvalóan tetszőleges lehet. Eredőjük ugyanis mindig egy-egy nem szomszédos specifikációs változást vagy bemeneti szakaszváltozást képvisel, amelynek lejátszódásakor a változó jelek változási sorrendjét úgyis a jelterjedési késleltetésektől függően észleli, ill. állítja elő a vezérlőegység.

Feleltessünk meg a folyamatábra minden egyes elemi műveletének egy-egy adatot, azaz címezhető információegységet a memóriában. A processzornak ekkor csupán az a feladata, hogy a folyamatábra által megszabott sorrendben rendre előállítsa az egyes végrehajtandó elemi műveleteknek megfelelő memóriabeli információegységek címeit és a memóriából kapott mindenkori adatok, azaz információegységek alapján elvégezze az azok által előírt elemi műveleteket.

Az elemi műveleteknek megfelelő memóriabeli információegységeket a továbbiakban *mikroutasításoknak* nevezzük, amelyek lényegében a processzort egy-egy elemi művelet végrehajtására utasítják. Ilyen értelmezésben a folyamatábrát mikroutasítások sorozataként ábrázoljuk a memóriatartalomban, vagyis a folyamatábrát a memóriában egy ún. *mikroprogram* írja le, amelyet a működés során a processzor végrehajt. Az ilyen elven kialakított vezérlőegységet *mikroprogramozott vezérlőegységnek* nevezzük.

Megjegyezzük, hogy a szakirodalomban [9], [17] általában mikroprogramozottnak nevezik a felépítést már akkor is, ha lényegében a 3.11. pontban vázolt módon visszacsatolt memóriáról van szó. Mivel azonban eddigi gondolatmenetünkben nem tettünk elvi különbséget a között, hogy egy sorrendi hálózat kombinációs részét memória-



4.61. ábra. A 4.26. ábrán szereplő folyamatábra elemi bemeneti és kimeneti műveletekből felépítve

elemmel vagy anélkül valósítjuk meg, ezért célszerűnek látszik a mikroprogramozott és a nem mikroprogramozott felépítés közötti határt a klasszikus szakirodalomtól eltérő módon értelmezni. A további ismeretek szempontjából ez nem okozhat semmiféle félreértést. Célunk ugyanis az ún. mikroprocesszorok lényegének egyfajta megközelítése, a hagyományos szakirodalmi értelmezés szerint pedig a folyamatábra elemi műveletein alapuló felépítést már gyakran mikroprocesszornak nevezik.

Nem szabad azonban megfeledkeznünk arról, hogy a bemutatandó gondolatmenettel csak az egyik lehetséges megoldást kapjuk meg a mikroprogramozott, ill.

mikroprocesszoros felépítésre. A választott mikroutasítások jellemzői azonban igen erősen befolyásolják a rendszertechnikai felépítést.

A folyamatábra elemi műveleteinek megfelelő mikroutasításoknak természetesen meg kell adniuk a processzor számára, hogy

— milyen fajtájú műveletet kell végrehajtani,
— melyik bemeneti, ill. kimeneti jellel kapcsolatos a művelet,
— a folyamatábra elágazása esetén melyik a soron következő művelet.

A memória egyes adatbitjei jelöljék például a 4.62. ábra szerinti elrendezésben a processzor által igényelt fenti végrehajtási előírásokat. Eszerint a D_1, D_2, D_3 adatbitek n-ből 1 kódban jelölik ki végrehajtandó művelet fajtáját, vagyis ezen a három biten jelenik meg az ún. *műveleti kód:*

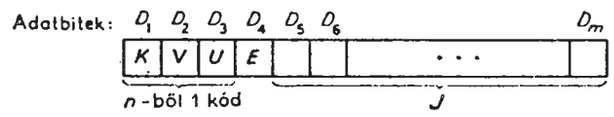
$K=1$, ha kimeneti jellel kapcsolatos a művelet.
$V=1$, ha bemeneti jellel kapcsolatos a művelet.
$U=1$, ha elágazást kell végrehajtani.

A D_4 adatbit értéke (E) jelölje azt, hogy kimeneti jellel kapcsolatos művelet esetén az illető kimeneti jelet milyen értékűre kell beállítani, bemeneti jellel kapcsolatos művelet esetén pedig az illető bemeneti jelet milyen értékkel kell összehasonlítani. A D_5-től D_m-ig terjedő bitek (J) értékkombinációi azt jelölik ki, hogy

— $K=1$ esetén melyik kimeneti jelre vonatkozik a művelet,
— $V=1$ esetén melyik bemeneti jelre vonatkozik a művelet,
— $U=1$ esetén melyik memóriacímre kell elágazni.

Az összes lehetséges kijelölő értékkombináció száma: 2^{m-4}. A memória adatkimeneteinek szükséges számát (m) tehát az határozza meg, hogy hány kimeneti jelet, bemeneti jelet, vagy memóriacímet kell megkülönböztetni a vezérlőegység működése során. Az adatkimenetek szükséges számát így mindig a legnagyobb megkülönböztetési igényű műveletfajta alapján kell meghatároznunk.

A 4.62. ábrán így a fenti értelmezésben megadtuk a mikroutasítás ún. *szószervezésének* egy lehetséges módját. A processzor működését az egymás utáni mikroutasítások határozzák meg. Az alaphelyzetbe állítás az ilyen felépítésben azt jelenti, hogy a processzor számára valamilyen módon megadjuk a folyamatábra kiindulási pontja utáni első elemi műveletnek megfelelő mikroutasítás címét. A processzor így ezt a címet közli a memóriával és erre válaszképpen az elsőként végrehajtandó elemi műveletnek megfelelő mikroutasítást kapja meg az adatkimeneteken. Egy mikroutasítás végrehajtása után a processzor általában a soron következő címet küldi a memóriának feltételezve, hogy ott helyezkedik el a soron következő mikroutasítás. Ezt a rögzített



4.62. ábra. A folyamatábra elemi műveleteit kijelölő memória-adatbitek egy lehetséges értelmezése a mikroutasításokhoz

sorrendet az elágazást okozó mikroutasítással ($U=1$) lehet megbontani, amelynek végrehajtásakor a processzor az elágazást okozó mikroutasítás D_5-től D_m-ig terjedő bitjein adott értékkombináció (J) alapján határozza meg a következő mikroutasítás címét. A bemeneti jelekkel kapcsolatos mikroutasítások végrehajtásakor szintén elágazást kell végeznie a processzornak. A kijelölt bemeneti jel értékétől függően ugyanis más-más helyen folytatódik a folyamatábra. Ezt például úgy oldhatja meg a processzor, hogy ha a kijelölt bemeneti jel értéke megegyezik az E jelű bit értékével, akkor a soron következő mikroutasítást kihagyja és az ez utáni címet küldi a memóriának. Ha pedig a kijelölt bemeneti jel értéke nem egyezik meg az E jelű bit értékével, akkor a soron következő címet juttatja a memóriára. A kihagyott címre elhelyezett elágazást okozó mikroutasítással így tetszőleges helyre történő elágazást tudunk megvalósítani a mikroprogramban.

A 4.62. ábrán bemutatott mikroutasítás szószervezést és az elágazások fentiekben vázolt megoldását feltételezve a folyamatábrát formálisan átalakíthatjuk ún. *szimbolikus mikroprogrammá.* Ebben az egyes mikroutasítások műveleti kódját a K, V és U betűk valamelyikével jelöljük, a műveleti kód után álló 1 vagy 0 az E jelű bit értékét jelenti. A kijelölőkombináció (J) értékét annak a jelnek vagy címnek a betűszimbólumával jelöljük, amelyre az adott mikroutasítás vonatkozik. Példaként a következőképpen írhatjuk fel a 4.61. ábrán szereplő folyamatábrának megfelelő mikroprogramot:

CIM1:	V 1 x_1
	U CIM6
	K 1 z_1
	K 1 z_3
CIM2:	V 1 x_2
	U CIM2
CIM3:	V 1 x_4
	U CIM3
	K 0 z_3
	K 1 z_4
CIM7:	V 1 x_1
	U CIM8
CIM4:	V 0 x_2
	U CIM4
	K 1 z_3
CIM10:	K 0 z_1

CIM5:	V 0 x_1
	U CIM5
	K 0 z_3
	K 0 z_4
	U CIM1
CIM6:	V 0 x_2
	U CIM1
	V 0 x_3
	U CIM1
	K 1 z_1
	K 1 z_2
	K 0 z_2
	K 1 z_4
	U CIM7
CIM8:	V 1 x_2
	U CIM7
	K 1 z_3
CIM9:	V 1 x_1
	U CIM9
	V 0 x_2
	U CIM9
	V 1 x_3
	U CIM9
	V 1 x_4
	U CIM9
	U CIM10.

Az elágazásokkal kapcsolatos címeket szintén szimbolikusan jelöltük. A szimbolikus mikroprogram alapján egyszerűen meghatározhatjuk az annak megfelelő memóriatartalmat. Ezt követhetjük a 4.63. ábrán. A memória adatkimeneteinek szükséges számát az elágazási címek határozzák meg, mert a mikroutasítások száma 42, a bemeneti és kimeneti jelek száma pedig egyaránt csupán 4.

A szükséges adatkimenetek száma így 10, mert a 42 mikroutasítás kijelöléséhez legalább 6 címbemenetet kell alkalmaznunk (C_1—C_6). A bemeneti és kimeneti jele-

	CÍM						ADAT									
							K	V	U	E	J					
	C_1	C_2	C_3	C_4	C_5	C_6	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
CÍM 1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1
	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	1
	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1
	0	0	0	0	1	1	1	0	0	1	0	0	0	0	1	1
CÍM 2	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0
	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0
CÍM 3	0	0	0	1	1	0	0	1	0	1	0	0	0	1	0	0
	0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0
	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1
	0	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0
CÍM 7	0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	1
	0	0	1	0	1	1	0	0	1	0	0	1	1	1	1	0
CÍM 4	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	0
	0	0	1	1	0	1	0	0	1	0	0	0	1	1	0	0
	0	0	1	1	1	0	1	0	0	1	0	0	0	0	1	1
CÍM 10	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1
CÍM 5	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1
	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0
	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	1
	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0
	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0
CÍM 6	0	1	0	1	0	1	0	1	0	0	0	0	0	0	1	0
	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0
	0	1	0	1	1	1	0	1	0	0	0	0	0	0	1	1
	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
	0	1	1	0	0	1	1	0	0	1	0	0	0	0	0	1
	0	1	1	0	1	0	1	0	0	1	0	0	0	0	1	0
	0	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0
	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	0
	0	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0
CÍM 8	0	1	1	1	1	0	0	1	0	1	0	0	0	0	1	0
	0	1	1	1	1	1	0	0	1	0	0	0	1	0	1	0
	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1
CÍM 9	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	1
	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1
	1	0	0	0	1	1	0	1	0	0	0	0	0	0	1	0
	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
	1	0	0	1	0	1	0	1	0	1	0	0	0	0	1	1
	1	0	0	1	1	0	0	0	1	0	1	0	0	0	0	1
	1	0	0	1	1	1	0	1	0	1	0	0	0	1	0	0
	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1
	1	0	1	0	0	1	0	0	1	0	0	0	1	1	1	1

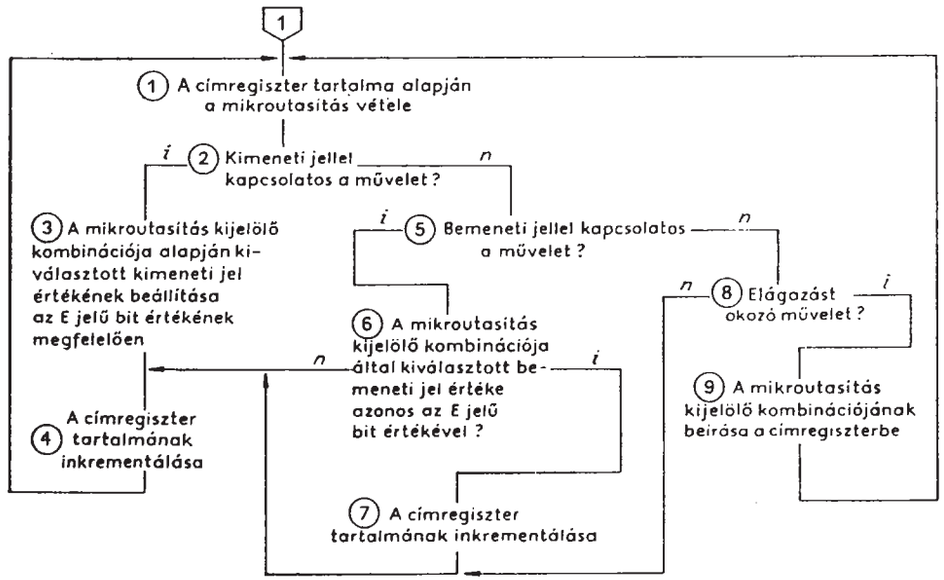
4.63. ábra. A 4.26. ábra alapján felírt szimbolikus mikroprogramnak megfelelő memóriatartalom

ket kijelölő értékkombinációkat az egyszerűség kedvéért úgy választottuk meg, hogy megegyezzenek a jeleket képviselő változók indexeinek megfelelő bináris számmal.

Mivel esetünkben a vezérlőegységnek csak 4 bemeneti, ill. kimeneti jele van és a J kijelölő értékkombináció szükséges helyértékeinek száma a mikroutasítások száma miatt 6, ezért a bemeneti és kimeneti jelek kijelöléséhez n-ból 1 kódot is használhattunk volna. Így a processzornak a bemeneti és kimeneti műveletek esetén nem kellene átkódolást végeznie, ami a működési sebesség szempontjából is kedvező hatású.

4.3.2. A processzor működésének leírása

A processzor eddig megfogalmazott feladata a mikroutasítások végrehajtása és a következő mikroutasítás címének előállítása. Az utóbbi részfeladat megoldása céljából célszerű, ha a processzor tartalmaz a mindenkori soron következő mikroutasítás címének tárolására egy olyan ún. *címregisztert,* amelyben a memóriára jutó cím minden egyes bitjének értékét egy-egy flip-flop tárolja. A címregiszternek ezen túlmenően célszerű számláló tulajdonsággal is rendelkeznie. Ez úgy értendő, hogy a címregiszternek van egy olyan bemenete is, amelyre adott impulzus hatására a címregisztert alkotó flip-flopok által tárolt bináris értékhez 1 hozzáadódik, azaz a címregiszter tartalma inkrementálódik. Ezért nevezhetjük az ilyen tulajdonságú címregisztert *utasításszámlálónak* is. Ez a számláló jelleg egyszerűvé teszi a soron következő cím előállítását. A 4.64. ábrán az eddigiek alapján szöveges folyamatábrával összefoglal-



4.64. ábra. A processzor működésének összefoglaló leírása

tuk a processzor működésének főbb mozzanatait. Az egyes részfeladatokat a későbbi hivatkozások érdekében számoztuk meg. Megfigyelhetjük, hogy a bemeneti jellel kapcsolatos művelet esetén az E értékkel való egyezés azt eredményezi, hogy a címregiszter inkrementálása kétszer kerül sorra és így valósul meg a soron következő címen elhelyezkedő mikroutasítás kihagyása. Az elágazást okozó műveletre vonatkozó kérdés „nem" ága lényegében azt az esetet jelenti, amikor a mikroutasítás műveleti kódja hibás, mert egyik műveletet sem jelöli ki. Ilyenkor kérdéses, hogy a processzor hogyan működjön tovább. A 4.64. ábrán azt a megoldást választottuk, hogy a processzor hagyja ki a hibás műveleti kódú mikroutasítást és vegye a következő mikroutasítást a soron következő címről. Egyetlen hibás műveleti kódú mikroutasítás sorrakerülésekor tehát annyival hosszabb idő telik el a megelőző és a követő helyes kódú utasítások végrehajtása között, amennyi időt igényel a processzor a soron következő utasítás vételéhez. Így hibás műveleti kódú mikroutasítások szándékos közbeiktatásával látszólagos várakozásra lehet késztetni a processzort két helyes kódú mikroutasítás között. Ezt a lehetőséget kihasználhatjuk adott szélességű impulzusok előállításakor. Az így előállítható impulzus időtartama természetesen nem lehet kisebb, mint a hibás kódú mikroutasítás végrehajtási ideje és ennek egész számú többszörösei szerint növekedhet attól függően, hogy hány hibás kódú mikroutasítást iktatunk közbe a mikroprogram adott helyén. Az adott szélességű impulzusok előállítására természetesen alkalmazhatjuk a fázisregiszteres vezérlőegységek esetében bemutatott megoldásokat is.

4.3.3. A processzor tervezése

A mikroutasítás szószervezésére a 4.62. ábrán választott megoldás és a 4.64. ábrán szereplő működésleírás alapján megtervezhetjük a processzort. Tekintsük a processzort logikai rendszernek a 1.4. pont szerint. A tervezés első lépése ilyenkor a részfeladatok és a funkcionális egységek meghatározása. A processzor kívánt működésének ismeretében például az alábbi részfeladatok fogalmazhatók meg:

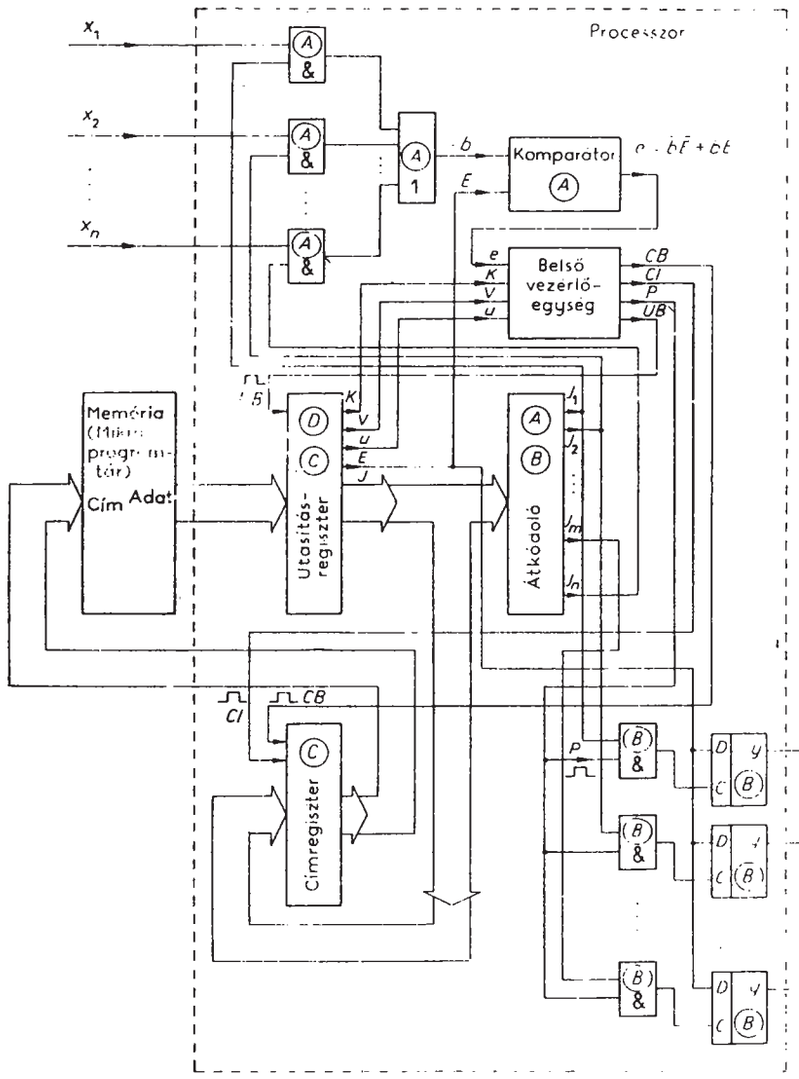
A) A mikroutasításban kijelölt bemeneti jel összehasonlítása az E jelű bit értékével.

B) A mikroutasításban kijelölt kimeneti jel beállítása az E jelű értékére.

C) A mikroutasítások címének előállítása inkrementálással, vagy az elágazást okozó mikroutasítás kijelölő kombinációja (J) alapján.

D) A mikroutasítások vétele a memória adatkimeneteiről és tárolásuk végrehajtás időtartamára.

A *D)* pontban említett részfeladatra azért van szükség, mert ezáltal a mikroutasítás végrehajtásának idején már előállítható a következő mikroutasítás címe. Ha a processzor nem tárolná a mikorutasítást, akkor csak a végrehajtás után volna ajánlatos a következő mikroutasítás címét a memóriára juttatni, mert ellenkező esetben a végrehajtás alatt megváltozhatna a memória adatkimenetein levő mikroutasítás, ami hibás



4.65. ábra. A processzor funkcionális egységeinek vázlata

végrehajtást okozna. Ennek a hibának a fellépése nyilvánvalóan annál valószínűbb volna, minél gyorsabb memóriát alkalmaznánk.

A 4.65. ábrán vázlatosan ábrázoltuk a fenti részfeladatokat megoldó funkcionális egységek egy lehetséges változatát. Az egyes kapukon és egységeken bekarikázva feltüntettük, hogy az A, B, C és D pontokban csoportosított részfeladatok közül, melyeknek a megoldásában vesznek részt.

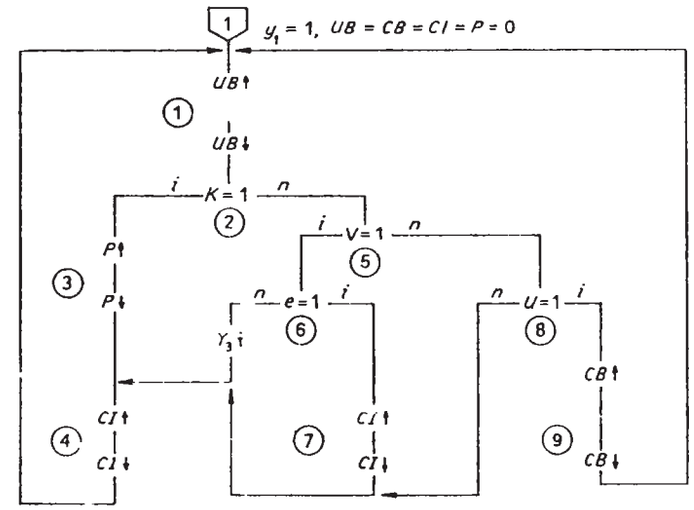
Az átkódolóegység a mikroutasítás kijelölő kombinációját (J) n-ből 1 kóddá ala-

kítja át. Így a $J_1 J_2 \ldots J_m \ldots J_n$ kimenetek közül mindig csak annak az értéke 1, amelynek megfelelő bináris kombináció az átkódolóegység bemenetén éppen fennáll. Ezeket a kimeneteket az *A* és *B* jelű ÉS kapcsolatok révén arra használtuk fel, hogy kiválasszuk azt a bemeneti, ill. kimeneti jelet, amelyet az utasításregiszterben levő mikroutasítás kijelölő kombinációja jelöl ki. Egy *A* jelű ÉS kapunak tehát akkor, és csak akkor lehet 1 értékű a kimenete, ha a bemenetére jutó bemeneti jel éppen ki van jelölve és értéke 1. Így az *A* jelű VAGY kapu *b* jelű kimenete akkor, és csak akkor 1, ha az éppen kijelölt bemeneti jel értéke 1. A komparátoregység, ennek a *b* jelnek az értékét hasonlítja össze az utasításregiszterben levő mikroutasítás *E* jelű bitjének értékével. A komparátor *e* jelű kimenetén akkor, és csak akkor jelenik meg 1 érték, ha $b=E$. Bemeneti jellel kapcsolatos művelet esetén tehát az összehasonlítás eredményét a komparátor kimenetének értéke szolgáltatja. A processzor belső vezérlőegysége így az *e* jel értékétől függően végezheti a következő mikroutasítás címének előállítását.

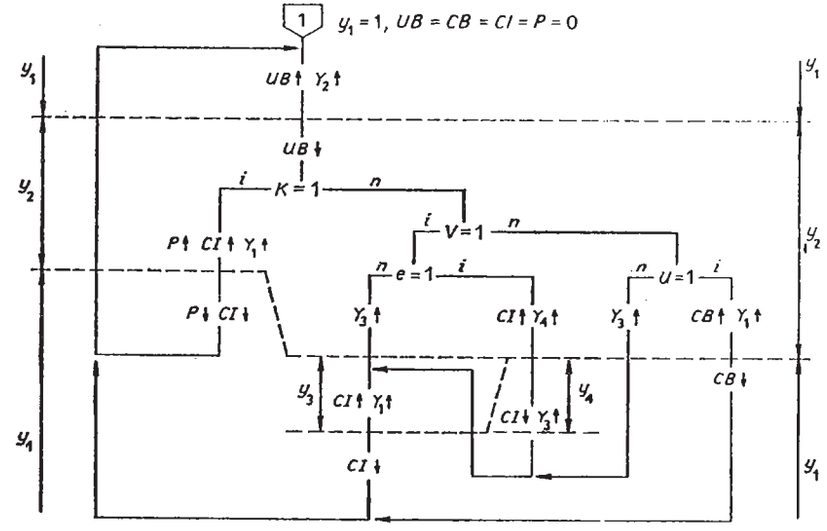
A kimeneti jelekkel kapcsolatos műveletek egyszerű végrehajthatóságát szolgálják a *B* jelű élvezérelt D flip-flopok. Ezek *D* bemenete az utasításregiszter *E* jelű kimenetére kapcsolódik. Az órajelbemeneteket a *B* jelű ÉS kapuk vezérlik, amelyek közül csak annak a kimenetén jelenhet meg a *P* jelű impulzus, amely az átkodoló kimenetei révén ki van választva, vagyis csak a mikroutasításban kijelölt kimeneti jel flip-flopja kaphatja meg az óraimpulzust, amely elvégzi az *E* értékére történő beállítást. A processzor belső vezérlőegységének tehát csak a *P* impulzust kell előállítania, ha kimeneti jelekkel kapcsolatos mikorutasítások kerülnek sorra. A belső vezérlőegységnek ezenkívül vezérelnie kell a memóriából érkező mikroutasítás betöltését az utasításregiszterbe, a címregiszter tartalmának inkrementálását, ill. a mikroutasítás kijelölő kombinációjának betöltését a címregiszterbe. A 4.65. ábrán feltételeztük, hogy a belső vezérlőegység ezeket a műveleteket egy-egy impulzussal (rendre: *UB*, *CI* és *CB*) vezérli. A vezérlőjelek megfelelő kiadásához elegendő a mikroutasítás műveleti kódjainak ismeretén kívül az *e* jelű komparátorkimenet értékének ismerete. Ezzel lényegében megadtuk a belső vezérlőegység kívánt működését a 4.65. ábrán vázolt funkcionális egységek esetén.

4.3.3.1. Szinkron működésű belső vezérlőegység tervezése

A 4.66*a)* ábrán a processzor belső vezérlőegységének folyamatábráját követhetjük a fenti gondolatmenettel megfogalmazott működés esetén. A belső vezérlőegység által megoldandó logikai feladat annyira egyszerű, hogy a specifikációs bemeneti és kimeneti változások ennek alapján igen könnyen jellemezhetők az egyes jelváltozásokkal. Így a 4.66*a)* ábrán bemutatott folyamatábrát a megoldandó feladat alapján közvetlenül célszerű megszerkeszteni a specifikációs gráf előzetes összeállítása nélkül. Az egyes műveletek melletti bekarikázott számok jelzik, hogy a 4.64. ábrán látható összefoglaló leírás melyik részfeladatának megoldását végzik az illető műveletek.



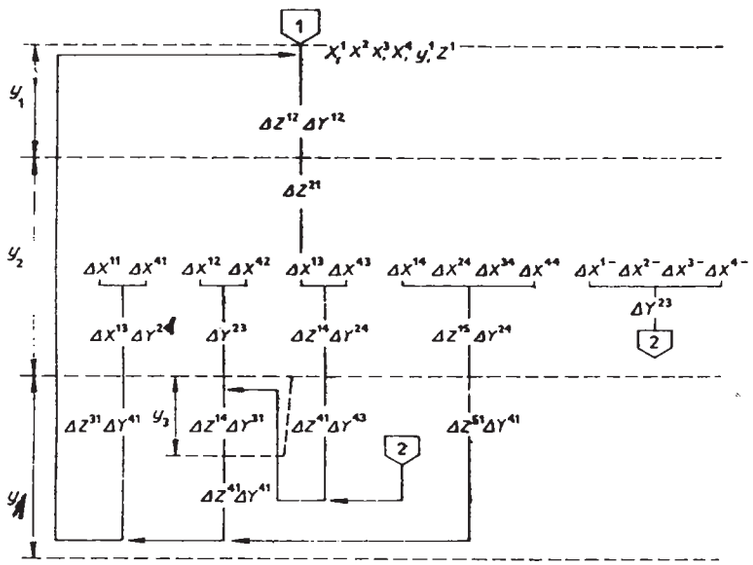
4.66. *a)* ábra. A processzor belső vezérlőegységének folyamatábrája a 4.65. ábrán szereplő funkcionális vázlat alapján megfogalmazott működés esetén



4.66. *b)* ábra. A processzor belső vezérlőegységének folyamatábrája a kimeneti műveletekhez tartozó címinkrementálás és a kimeneti flip-flopok beírásának párhuzamos végrehajtása esetén (A *P* és *CI* impulzusok egyidejűek)

	K	V	U	e
X^1	1	0	0	–
X^2	0	1	0	0
X^3	0	1	0	1
X^4	0	0	1	0

	UB	P	CI	CB
Z^1	0	0	0	0
Z^2	1	0	0	0
Z^3	0	1	1	0
Z^4	0	0	1	0
Z^5	0	0	0	1



4.66. *c)* ábra. A processzor belső vezérlőegységének specifikációs gráfja a 4.66*b)* ábra alapján

Figyeljük meg a 4.65. ábra alapján, hogy az utasításregiszter lehetővé teszi $K=1$ esetén a P és a CI impulzusok egyidejű előállítását. A CI impulzus ugyanis kimeneti művelet esetén ($K=1$) inkrementálhatja a címregiszter tartalmát már a P impulzus hatásával egyidejűleg, vagyis már ekkor a memória címbemeneteire juttathatja a következő mikroutasítás címét, hiszen az utasításregiszter a soron következő UB impulzusig tárol minden mikroutasítást. Így a korábban előállított következő cím az éppen folyamatban levő utasítás-végrehajtást nem zavarja, a működés viszont ezáltal gyorsabb lehet a kimeneti műveletek esetén. Könnyen belátható ugyanis, hogy a P-vel egyidejű CI impulzus így a 4.66*a)* ábrán vázolt megoldáshoz képest korábban indítja a memória működését a következő mikroutasítás vétele céljából.

A 4.66*b)* ábrán követhetjük a fenti módosítás révén keletkező folyamatábrát.

A 4.66*c)* ábrán felrajzoltuk a 4.64. és 4.66*b)* ábrából visszakövetkeztethető specifikációs gráfot. Az X^1 bemeneti kombinációban az e jelnek megfelelő helyértéken közömbös bejegyzést tettünk, hiszen a kimeneti művelet végrehajtását a belső vezérlőegység nem különbözteti meg az e jel értéke szerint. Természetesen az alkalmazott X^1 helyett két bemeneti kombinációt is rögzíthettünk volna $e=0$, ill. $e=1$ értékekkel. Így azonban még jobban megnövekedne a specifikációs változások száma. A 4.62. ábrán vázolt szószervezés alapján értelmezett mikroutasításokból felépülő mikroprogramról feltételezhetjük, hogy bemeneti műveletet ($V=1$) csak elágazást végrehajtó művelet ($U=1$) követhet közvetlenül. Így a specifikációs bemeneti változások közül kihagyhattuk a ΔX^{21}, ΔX^{23}, ΔX^{31}, ΔX^{32}, ΔX^{22}, ΔX^{33} jelűeket.

Mivel a belső vezérlőegység működését leíró folyamatábrák (4.66*a)* és 4.66*b)* ábra) kiindulási pontja utáni első művelet az utasításregiszter betöltése az UB impulzussal, ezért a kiindulási bemeneti kombináció megadásának csak formális jelentősége van. A specifikációs gráfon azért tüntettük fel az összes specifikált bemeneti kombinációt a kiindulási pontban, mert a működés során a kiindulási ponthoz való visszatéréskor nyilvánvalóan bármelyikük fennállhat. A ΔX^{1-}, ΔX^{2-} stb. jelöléseket a tömörebb leírás céljából alkalmaztuk mindazokra a specifikációs bemeneti változókra, amelyek során az X^1, X^2 stb. specifikált bemeneti kombinációkat valamelyik nem specifikált bemeneti kombináció (pl.: $KVUe=1101$) követi.

A 4.64. ábra szerint ezek a bemeneti változások hibás műveleti kódú mikroutasítás hatására játszódhatnak le és a címregiszter tartalmának egyéb művelet nélküli inkrementálását okozzák. Határozzuk meg a belső vezérlőegység szinkron fázisregiszteres, elvi logikai rajzát. A 4.66*b)* ábrán bejelölt fázishatárok alapján a kimeneti jelekre vonatkozó függvények:

$$UB{:}1 = y_1, \qquad UB{:}0 = y_2,$$
$$P{:}1 = y_2 K, \qquad P{:}0 = y_1 K,$$
$$CI{:}1 = y_2(K+Ve)+y_3, \qquad CI{:}0 = y_1(K+V+\bar{U})+y_4 = y_1\bar{U}+y_4,$$
$$CB{:}1 = y_2 U, \qquad CB{:}0 = y_1 U.$$

Például a $CI{:}1$ kifejezésben a formálisan adódó $\bar{K}Ve$ szorzatból a $\bar{K}$ tényező azért is elhagyható, mert a mikroutasítások műveleti kódjára n-ből 1 kódot tételezünk fel, így $V=1$ esetén $\bar{K}$ biztosan 1 értékű. Ugyanilyen okból egyszerűsíthettük a $CB{:}1$ kifejezésre formálisan adódó $y_2\bar{K}\bar{V}U$ szorzatot. A $CB{:}0$ kifejezésben szintén az n-ből 1 kód miatt helyettesíthetjük a $K+V$ tényezőt $\bar{U}$-tal.

A kimeneti flip-flopok lehetséges elhagyásával az alábbi kimeneti függvényekhez jutunk.

$$UB = y_1,$$
$$P = y_2 K,$$
$$CI = y_2(K+Ve)+y_3,$$
$$CB = y_2 U.$$

$CI=0$ értékét így $U=1$ esetén is beállítja a belső vezérlőegység, de ez hibát nem okoz, mert $U=1$ mellett $CI=1$ nem lép fel.

Az állapotváltozásokra (fázisátmenetekre) vonatkozó függvények a 4.66*b)* ábra alapján:

$$Y_1 = y_2(K+U)+y_3,$$

$$Y_2 = y_1,$$

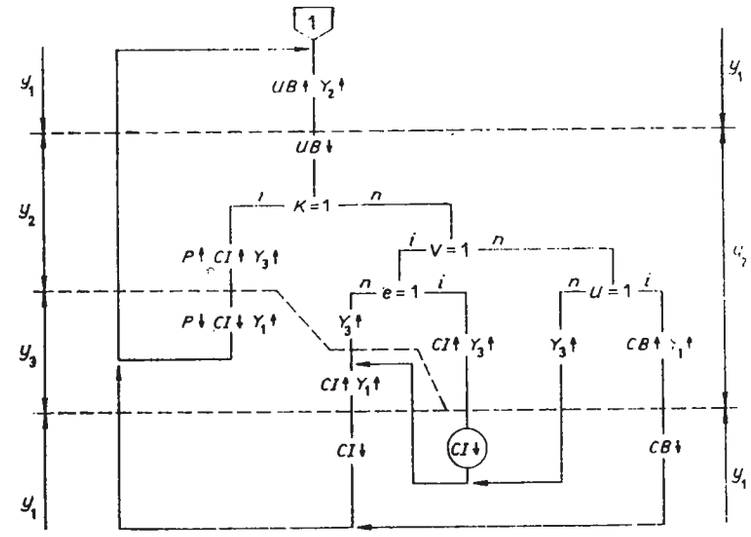
$$Y_3 = y_2(V\bar{e}+\bar{K}\,\overline{VU})+y_4,$$

$$Y_4 = y_2 Ve.$$

Megfigyelhetjük, hogy a specifikációs bemeneti változásoknak csupán egy-egy jelértékkel történt jellemzése következtében az így kiadódó folyamatábra szerint működő belső vezérlőegység nem minden hibás műveleti kód esetén inkrementálja a címregiszter tartalmát. A 4.66*b)* ábra alapján megállapíthatjuk, hogy ezt az összes lehetséges hibás műveleti kódok közül csupán a $KVU=000$ idézi elő. A többi hibás műveleti kódot azzal jellemezhetjük, hogy a K, V és U bemeneti jelek közül legalább kettőnek az értéke 1. A folyamatábrából a fenti módon kiolvasott függvények alapján megvalósított fázisregiszteres egységben az ilyen hibás műveleti kódok hibás működést okoznak. A függvények felírásakor ugyanis a K, V és U értékektől függő elágazások értelmezésében kihasználtuk, hogy a KVU kombinációk n-ből 1 kódúak, és a K, V, U jelekre vonatkozó bemeneti műveletek VAGY kapcsolatban levő specifikációs bemeneti változásokat jellemeznek, tehát sorrendjük felcserélhető. Emiatt a hibás műveleti kódok előidézhetik például azt, hogy $K=1$ és $U=1$ mellett a CI, P és CB impulzusok egyidejűleg fellépnek, ami nyilvánvalóan hibás működésnek tekinthető.

Így csak egyetlen hibás műveleti kód ($KVU=000$) eredményezi a címregiszter tartalmának inkrementálását egyéb művelet-végrehajtás nélkül. Ezáltal csak ez a műveleti kód alkalmas a 4.64. ábrával kapcsolatban említett impulzus-előállításra. A $KVU=000$ műveleti kódú mikroutasítást *művelet nélküli* utasításnak nevezhetjük, amelyhez hasonló ún. *NOP (No Operation)* utasításként megtalálható a legtöbb mikroprogramozott berendezés mikroutasítás-készletében.

Láthatjuk, hogy a processzor vezérlőegységének a bemutatott megoldásban impulzusokat kell előállítania. Lényegében az egymás utáni kimeneti impulzusok határozták meg a szükséges fázisok számát. Általában csökkenthetjük a szükséges fázisok számát, ha a kimeneti impulzusok előállításához felhasználjuk a szinkron fázisregiszteres vezérlőegységben alkalmazott órajelet. A 4.66*d)* ábrán követhetjük, hogy miként csökken a fázisok száma, ha példaként a CI impulzust a C_1 órajellel történő kapuzással hozzuk létre. Az így keletkező CI kimeneti impulzusok időtartamát nyilvánvalóan a C_1 impulzus szélessége szabja meg, tehát rövidebb lesz, mint az órajelperiódus. Így az egymás utáni CI impulzusok közötti impulzusszünetet nem kell külön fázissal biztosítanunk. Az egymás utáni CI impulzusok tehát egymást közvetlenül követő fázisokban lehetnek, mert a C_1 órajellel történő kapuzás még CI megjelenésének fázisaiban létrehozza a szükséges impulzusszüneteket. Azt is könnyen beláthatjuk, hogy így a $V=1$ és $e=1$ esetén igényelt két CI impulzus közül az első-



4.66*d)* A processzor belső vezérlőegységének folyamatábrája a CI kimeneti impulzus órajelkapuzással történő előállításának feltételezése esetén

höz tartozó $CI=0$ értéket beállító műveletet nem is szükséges feltüntetni a folyamatábrán. A 4.66*d)* ábrán ezt az elhagyható műveletet bekarikázással jelöltük. Elhagyása esetén a $Ve=11$ ágon is y_2-ből az y_3 fázisba történik közvetlen átmenet. A 4.66*d)* ábrán felvett fázishatárok alapján az alábbi logikai függvényeket írhatjuk fel:

$$UB{:}1 = y_1,$$

$$UB{:}0 = y_2,$$

$$P{:}1 = y_2 K,$$

$$P{:}0 = y_3 K,$$

$$CI{:}1 = [y_2(K+Ve)+y_3\bar{K}]C_1$$

$$CI{:}0 = [y_1+y_3 K]\,C_1,$$

$$CB{:}1 = y_2 U,$$

$$CB{:}0 = y_1 U,$$

$$Y_1 = y_2 U+y_3,$$

$$Y_2 = y_1$$

$$Y_3 = y_2 \bar{U}.$$

A kimeneti flip-flopok elhagyásával a kimeneti függvények a következőképpen alakulnak:

$$UB = y_1,$$

$$P = y_2 K,$$

$$CI = [y_2(K+Ve)+y_3\overline{K}]\,C_1,$$

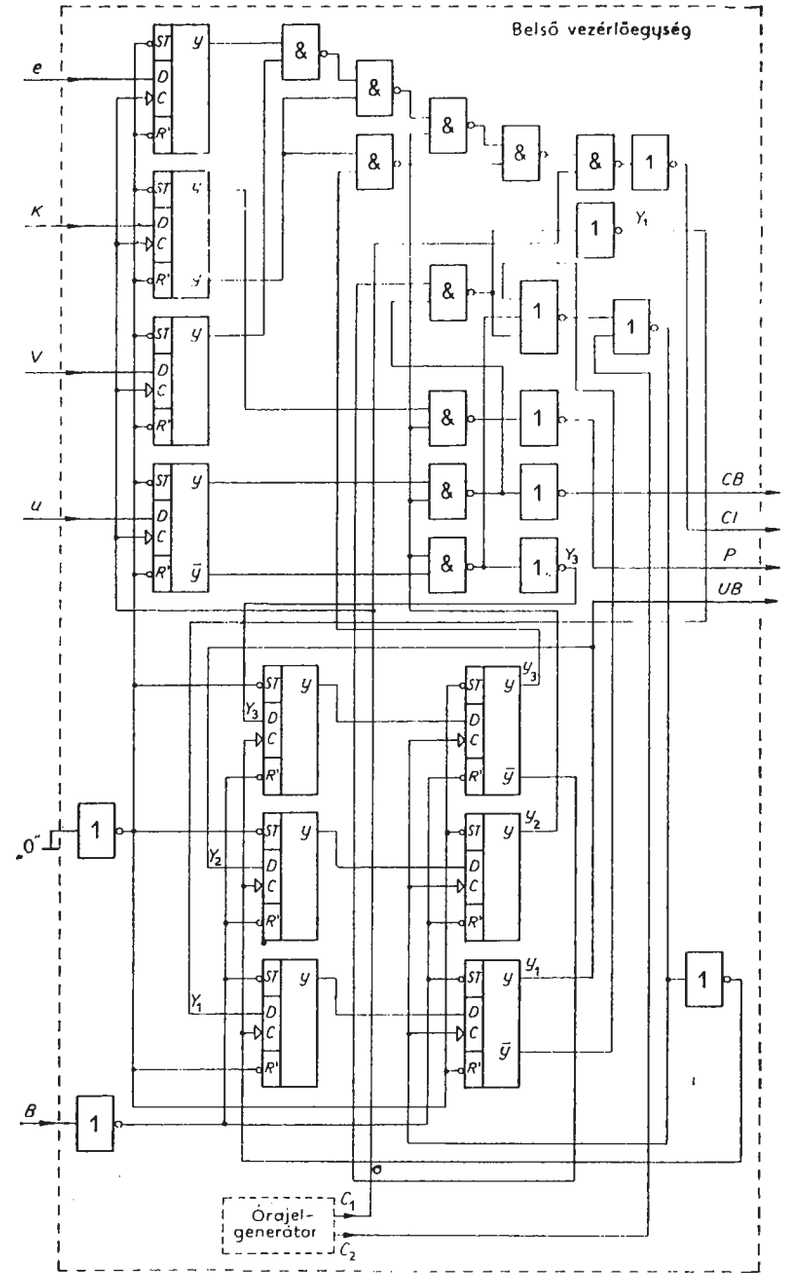
$$CB = y_2 U.$$

Figyeljük meg, hogy a P, CI és CB jelek 0-ra állításához nem vettünk fel új fázist, hanem az y_1 fázis kezdetét helyeztük át a folyamatábra kiindulási pontjáról. Ezzel lényegében azt idéztük elő, hogy az alaphelyzetbe állításkor a létrejövő y_1 fázisban rögtön megtörténik a P, CI és CB jelek 0-ra állítása, ami a működés indulásakor nyilvánvalóan megengedhető.

A fentiekhez hasonló megoldásokkal általában csökkenthetjük a szükséges fázisok számát.

A 4.67. ábrán a fenti függvények alapján felrajzoltuk a processzor belső vezérlőegységének szinkron fázisregiszteres elvi logikai rajzát. Tételezzük fel, hogy az utasításregiszter és a címregiszter UB és CB vezérlőbemenetei szintérzékenyek, a címregiszter CI vezérlőbemenete pedig élérzékeny a felfutóélre. A továbbiakban látni fogjuk, hogy az utasításregisztert célszerű D—G flip-flopok-ból (latch) felépíteni és az UB jelet a G bemenetekre vezetni. A címregiszternek a CI vezérlőbemenet előírt szerepe miatt számláló tulajdonsággal kell rendelkeznie. A CB vezérlőjel a címregisztert alkotó flip-flopok Preset és Clear bemenetei révén fejtheti ki a hatását legegyszerűbben. Ezért úgy tekinthetjük, hogy a J kombináció egyes bitjeit a CB jel a címregisztert alkotó flip-flopok Pr, ill. Cl bemeneteire kapuzza az egyes bitek értékétől függően. A felírt függvények alapján megfigyelhetjük, hogy ilyen feltételezés mellett a kimeneti flip-flopok elhagyása miatt nem lépnek fel hazárdjelenségek a kimeneteken. A belső vezérlőegységbe beleérthetjük a kétfázisú órajelet előállító órajel-generátort is.

Az alaphelyzetbe állító impulzus (B) hatását a szinkron fázisregiszteres vezérlőegységekben szokásos módon oldottuk meg. A bemeneti szinkronizáló flip-flopokra azonban nem vezettük rá a B impulzust. Ennek oka az, hogy a belső vezérlőegység alaphelyzetbe állítása nem elegendő a teljes processzor alaphelyzetének beállításához. Ehhez ugyanis mindenképpen az szükséges, hogy kiindulásként a címregiszterbe kerüljön a végrehajtandó mikroprogram első utasításának memóriabeli címe (pl. a csupa 0-t tartalmató cím). Ezért a B impulzusnak hatnia kell a címregiszterre is (pl. nulláznia kell annak tartalmát). A belső vezérlőegységet a B impulzus nyilvánvalóan csak akkor engedheti tovább az y_1 fázisból, ha az első mikroutasítás már bekerült az utasításregiszterbe. Ellenkező esetben ugyanis az y_2 fázisban hibás műveleti kód alapján folytatódhatna a működés. A B impulzus időtartama alatt tehát a belső vezérlőegység bemeneti szinkronizáló flip-flopjainak fogadniuk kell a K, V, U és e jeleket.



4.67. ábra. A processzor belső vezérlőegységének szinkron fázisregiszteres elvi logikai rajza

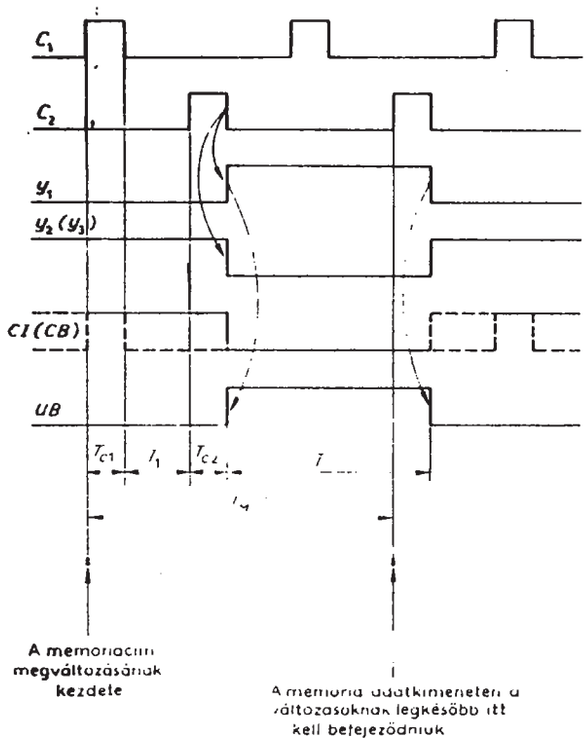
Ha a B impulzussal a bemeneti szinkronizáló flip-flopokat is alaphelyzetbe állítanánk a Pr és Cl bemeneteik felhasználásával, akkor ezáltal a belső vezérlőegységet a B impulzus időtartama alatt érzéketlenné tennénk a bemeneti változásokra, ami a processzor alaphelyzetbe állításakor az y_2 fázisban nem az első mikroutasítás műveleti kódja szerinti működést okozná.

A B impulzus időtartamát a fentiek értelmében megszabja a memória ciklusideje. Ha ugyanis az impulzusszélesség kisebb, mint a ciklusidő, akkor a belső vezérlőegység a B megszűnése utáni legközelebbi C_2 óraimpulzus hatására még azelőtt átléphetne az y_2 fázisba, mielőtt még a memória adatkimenetein megjelent volna az első mikroutasítás. Így szintén hibás műveleti kódot értelmezne a belső vezérlőegység az y_2 fázisban.

Hasonló okokból képez peremfeltételt a memória ciklusideje a C_1, C_2 kétfázisú órajel periódusidejére és impulzusszélességeire vonatkozóan. A címregiszter tartalmának módosítása (CI vagy CB hatására) ugyanis mindig az y_1-et megelőző fázisban történik (y_2-ben vagy y_3-ban). Ezt szemlélteti a 4.68. ábrán látható idődiagram. A felírt függvényekből következik, hogy a CB és CI jelek vagy C_2 lefutó vagy C_1 felfutó élének hatására válnak 1 értékűvé az y_2, ill. y_3 fázisban. Az ábrán erre szaggatott vonalak utalnak. A memóriacím megváltozása tehát az y_2, ill. y_3 fázisban nem mindig indul meg rögtön a fázisba való átlépéskor, hanem a bemeneti jelektől való függőség (pl. CI kifejezésének első tagja) miatt csak a C_1 óraimpulzus felfutóélének hatására. Az y_2, ill. y_3 fázist követő y_1 fázisban a belső vezérlőegység az UB jellel írja be a mikroutasítást az utasításregiszterbe. Mivel az utasításregiszter UB bemenetéről szintérzékenységet tételeztünk fel, ezért a helyes működéshez csupán azt kell biztosítanunk, hogy még UB lefutó éle előtt elegendő idővel befejeződjenek a memória adatkimenetein a változások, vagyis megjelenjen a végrehajtandó mikroutasítás. Így az UB jel lefutó éle után (az y_2 fázisban) a belső vezérlőegység már biztonságosan érzékelheti a műveleti kódot és az e jel értékét. Ez utóbbi állandósult értékének kialakulását természetesen a 4.65. ábra szerinti A jelű kapuk és a komparátor jelkésleltető hatása is befolyásolja. A 4.68. ábra alapján legkedvezőtlenebb esetként tételezzük fel, hogy a memóriacím megváltozása az y_2, ill. y_3 fázisban a C_1 óraimpulzus felfutó élének hatására indul meg. Tekintsük még elfogadhatónak, ha az adatkimenetek változása legkésőbb egy C_2 impulzusszélességnyi idővel (T_{c2}) UB lefutó éle előtt befejeződik. Az ábrán T_M-mel jelöltük azt az így kiadódó időtartamot, amelynek a memória ciklusidejénél feltétlenül nagyobbnak kell lennie:

$$
\begin{aligned}
T_M &= T_{c1}+T_1+T_{c2}+T-T_{c2} = \\
&= T_{c1}+T_1+T = \\
&= T_{c1}+T_1+T_{c2}+T_2+T_{c1}+T_1 = \\
&= 2(T_{c1}+T_1)+T_2+T_{c2} > t_c,
\end{aligned}
$$

ahol t_c-vel jelöltük a memória ciklusidejét.



4.68. ábra. A memória ciklusideje által a C_1, C_2 kétfázisú órajel periódusidejére és impulzusszélességére megszabott peremfeltétel szemléltetése

4.4. A fázisregiszteres és a mikroprogramozott megoldás összehasonlítása a vezérlőegység működési sebessége szempontjából

Az előző fejezetekben két alapvetően különböző úton jutottunk el a vezérlőegységnek a 4.60. ábrán vázolt felépítéséhez. A fázisregiszteres megoldások közül nyilvánvalóan az aszinkron változat a gyorsabb működésű. A szinkron és az aszinkron sorrendi hálózatok működési módjából ez egyértelműen következik. A mikroprogramozott megoldás még a szinkron fázisregiszteres változatnál is biztosan lassúbb, ha a processzor belső vezérlőegysége szinkron fázisregiszteres felépítésű. A bemutatott mikroprogramozott megoldásban ugyanis a folyamatábra minden elemi művelete egy-egy mikroutasítást igényel. Egy mikroutasítás végrehajtása során a processzor szinkron fázisregiszteres belső vezérlőegysége három fázisátmenetet, vagyis legalább három órajel-periódusnyi időt igényelhet. Ugyanakkor — azonos órajel-frekvenciát feltételezve — a szinkron fázisregiszteres vezérlőegység egyetlen órajel-periódusnyi idő alatt a folyamatábra több műveletét hajthatja végre.

Ha a processzor belső vezérlőegysége aszinkron fázisregiszteres felépítésű, akkor a mikroutasítás végrehajtási idejét lényegében csak az alkalmazott memória ciklusideje határozza meg. Ha tehát a szinkron fázisregiszteres megoldásban a kombinációs hálózatot ugyanilyen sebességű memóriával valósítjuk meg, akkor a mikroprogramozott változat kisebb működési sebessége csak abból adódik, hogy a folyamatábra minden egyes elemi művelete egy-egy memóriaciklust igényel. A szinkron fázisregiszteres felépítésben viszont egy memóriaciklusnyi és egy órajel-periódusnyi idő alatt általában több, nem feltétlenül elemi folyamatábra-művelet játszódik le. A két megoldás sebességarányát így az órajel periódusideje és a memória ciklusideje közötti arány befolyásolja.

Ha a fázisregiszteres megoldás aszinkron felépítésű, akkor természetesen az órajel-periódusidő nem adódik hozzá az egy fázisban levő műveletek végrehajtási idejéhez.

A bemutatott felépítésű mikroprogramozott vezérlőegységek esetén a bemeneti kombinációk megengedhető legnagyobb változási sebességét jelentősen korlátozhatja az a tény, hogy egy mikroutasítás hatására csak egyetlen jelet képes megvizsgálni, ill. beállítani a processzor. Így egy specifikációs változás, vagy szakaszváltozás észleléséhez, ill. létrehozásához általában több mikroutasítás végrehajtására van szükség. Nyilvánvaló, hogy a bemeneti kombináció mindaddig nem változhat, amíg egy adott specifikációs bemeneti változásra, ill. szakaszváltozásra történő várakozásnak megfelelő összes mikroutasítás végrehajtása be nem fejeződött. Ellenkező esetben az egymás utáni jelenkénti vizsgálatok egyik része a régi, másik része az új bemeneti kombinációra vonatkozna, ami hibás elágazást vagy elakadást okozhatna a vezérlőegység működésében. A jelenkénti kimeneti változást okozó mikroutasítások végrehajtásához szükséges idő szintén csökkenti a bemeneti változások megengedhető

legnagyobb sebességét, hiszen ezalatt a bemutatott felépítésű mikroprogramozott vezérlőegység nem tud bemeneti kombinációt észlelni.

Ha a fázisregiszteres megoldásban a kombinációs hálózatot nem memóriával valósítjuk meg, akkor a memória ciklusideje helyett a sebesség-összehasonlítás során a kombinációs hálózat jelterjedési késleltetését kell figyelembe venni, amely általában lényegesen kisebb a memória ciklusidejénél. Különösen így van ez, ha a kombinációs hálózatot PLA elemekből építjük fel. Tételezzük fel, hogy a memória ciklusideje ugyanolyan nagyságrendű, mint a szinkron megoldásokban alkalmazható legkisebb órajel-periódusidő. Ekkor a vezérlőegység különböző felépítési módjai csökkenő működési sebesség szerint a következőképpen rangsorolhatók:

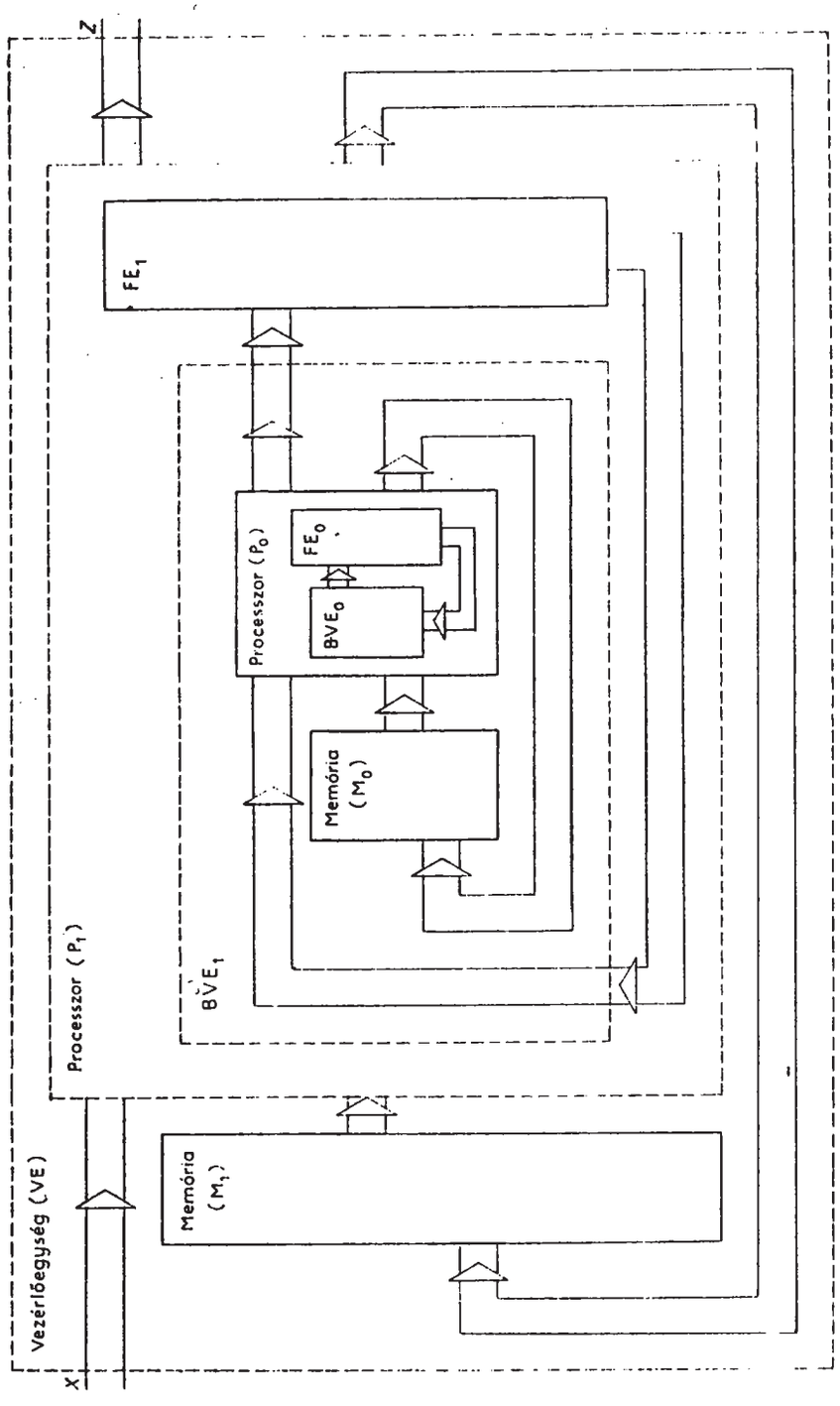
1. Aszinkron fázisregiszteres, a kombinációs hálózat memória nélkül megvalósítva.
2. Aszinkron fázisregiszteres, a kombinációs hálózat memóriával megvalósítva.
3. Szinkron fázisregiszteres, a kombinációs hálózat memória nélkül megvalósítva.
4. Szinkron fázisregiszteres, a kombinációs hálózat memóriával megvalósítva.
5. Mikroprogramozott, a processzor belső vezérlőegysége aszinkron fázisregiszteres.
6. Mikroprogramozott, a processzor belső vezérlőegysége szinkron fázisregiszteres.

A 2. pont szerinti megoldásnak kevés gyakorlati jelentősége van, mert a memóriakimenetek tranziens viselkedése az aszinkron működést zavarja.

A fázisregiszteres megoldásokban akkor érhetjük el a legnagyobb működési sebességet, ha a kombinációs hálózatot PLA-val valósítjuk meg. A mikroprogramozott megoldások vizsgálatakor nem szabad megfeledkeznünk a processzor átkódolóegységéről, amely mint kombinációs hálózat felépíthető, memóriaelemekből is. Ilyenkor természetesen további memóriaciklus-idővel kell számolnunk minden mikroutasítás végrehajtásakor. Sebesség szempontjából tehát mindenképpen kedvező, ha az átkódolóegység elhagyható. Láttuk, hogy a 4.26. ábra alapján felírt szimbolikus mikroprogram esetén erre lett volna lehetőség. Általában azonban az átkódolóegység elhagyhatóságát eredményező n-ből 1 kód alkalmazása lényegesen megnöveli a mikroprogramtárat alkotó memóriaegység szükséges adatkimeneteinek számát. Az átkódolóegység nélküli megoldást *horizontális mikroprogramozásnak* nevezik, ellentétben a *vertikális mikroprogramozással,* amely az n-ből 1 kódtól eltérő kijelölő értékkombinációkat alkalmazó felépítésre utal.

A mikroprogramozott vezérlőegység bemutatott változatában jelentős lassító tényező az, hogy a processzor egy mikroutasítással csak egyetlen elemi műveletet tud végrehajtani. Ha bonyolultabb folyamatábra-műveleteket feleltetünk meg a mikroutasításoknak, akkor a processzor folyamatábrájának bonyolultabb működést kell leírnia, következésképpen mind a belső vezérlőegység, mind a belső funkcionális egységek bonyolultabbak lesznek.

Ha nem törődnénk a működési sebesség csökkenésével, akkor a bonyolultabb processzort előállíthatnánk a 4.72. ábrán vázolt egymásba ágyazott felépítés szerint.



4.72. ábra. Egymásba ágyazott mikroprogramozott felépítés szemléltetése

Az M_0 memória a P_0 processzort alkotó BVE_0 belső vezérlőegységgel és az FE_0 funkcionális egységekkel együtt a 4.60. ábrán bemutatott módon létrehozza az elemi folyamatábra-műveleteket végrehajtó mikroprogramozott vezérlőegységet. Ezt tekintjük a P_1 processzor BVE_1 belső vezérlőegységének, amely az FE_1 funkcionális egységekkel és az M_1 memóriával együtt a VE jelű mikroprogramozott vezérlőegységet alkotja. A VE mikroprogramozott vezérlőegység működésének leírásához az elemi folyamatábra-műveleteknél bonyolultabb műveleteket használhatunk. Ezeket a bonyolultabb műveleteket az M_0 memóriában levő elemi folyamatábra-műveleteknek megfelelő mikroutasításokból összeállított mikroprogram határozza meg természetesen az FE_1 funkcionális egységekkel együtt. Az M_1 memóriában alkalmazható mikroutasítások így ezeknek a bonyolultabb műveleteknek felelnek meg. A VE vezérlőegység működését meghatározó M_1-beli mikroprogram egyes utasításai tehát az M_0-beli mikroprogram több elemi mikroutasításának végrehajtását indítják el. Ezért az M_0 és P_0 egységeket rendre *végrehajtó-(executiv) tárnak* és *végrehajtó-(executiv) processzornak* nevezik.

A fenti, jelentősen legyszerűsített gondolatmenetből az is következik, hogy a 4.72. ábrán vázolt egymásba ágyazott felépítés elvileg tovább folytatható. A VE vezérlőegységet tekinthetnénk értelemszerűen a BVE_2 belső vezérlőegységnek, amely a szükséges funkcionális egységekkel együtt további fölérendelt processzort és ezzel további ún. *magasabb mikroprogramozási szintet* hozna létre, és így tovább. Az egyre magasabb mikroprogramozási szintek felé haladva a processzorok egyre bonyolultabb mikroutasítások végrehajtására lesznek képesek. Az egyre bonyolultabb mikroutasítások végrehajtási ideje azonban az egyre több alsóbb szinten működő memóriaegység miatt egyre hosszabb. Az egymásba ágyazást tehát addig a szintig érdemes folytatni, amíg a legfelső szinten alkalmazható mikroutasítások végrehajtási ideje olyan hosszú nem lesz, hogy bonyolultságukat már éppen emiatt nem tudjuk hasznosítani az adott feladat megoldása során.

A magasabb mikroprogramozási szintek felé haladva a funkcionális egységek is általában egyre bonyolultabbak. Az 1.4. fejezetben bemutatott gondolatmenettel az alkalmazható építőelem-készlettől függően egy adott szinten előfordulhat, hogy funkcionális egységeken belül is belső vezérlőegységet kell megvalósítani. Az egymásba ágyazható szintek számát adott sebességi követelmények esetén nyilvánvalóan alapvetően befolyásolja:

— az egyes szintek mikroprogramjait tároló memóriák működési sebessége, vagyis ciklusideje,
— az építőelem-készlet fejlettsége.

Az integrált áramköri építőelem-készlet fejlődése leegyszerűsítve úgy fogalmazható meg, hogy növekszik az integráltsági fok és a működési sebesség, csökken a méret és a teljesítményfelvétel.

Az integráltsági fok növekedésének hatását a 4.72. ábra alapján úgy képzelhetjük el, hogy egy adott szintig egyre kevesebb építőelem felhasználásával juthatunk el.

Ha például BVE_0 és FE_0 rendelkezésünkre áll egy-egy integrált áramköri tokban, akkor BVE_1 előállításához csupán az M_0-beli mikroprogramot kell elkészíteni. Ugyanez a helyzet, ha a teljes P_0 processzor adott egyetlen építőelemként. Minőségi változásnak tekinthető, ha egy adott feladatra BVE_1 készen kapható. Ilyenkor ugyanis a már a P_1, FE_1, M_1 által meghatározott szint válik végrehajtóvá (executivvá) az esetleges magasabb szintek szempontjából. Ha a BVE_1 egységet megvalósító építőelem viszonylag gyors működésű (például PLA elem felhasználásával készült aszinkron fázisregiszteres belső felépítésű), akkor a P_1 processzor viszonylag bonyolult mikroutasításokból álló mikroprogramok viszonylag gyors végrehajtására lesz alkalmas. Leegyszerűsítve tehát azt mondhatjuk, hogy annál bonyolultabb mikroutasításokig folytatható a 4.72. ábrán vázolt egymásba ágyazás, minél fejlettebb építőelem-készlet áll rendelkezésünkre. Az integráltsági fok fejlődése révén egyre magasabb mikroprogramozási szintekre alkalmas belső vezérlőegységek, funkcionális egységek, processzorok és memóriaegységek jelennek meg készen kapható építőelemekként, amelyeket összefoglalóan *mikroprocesszor-elemeknek* neveznek. Ezek belső felépítése általában nem egyezik meg teljesen a bemutatott fázisregiszteres és mikroprogramozott megoldásokkal, mert az integrált áramköri technológia különleges peremfeltételeket szab a logikai tervezéshez is. A bemutatott gondolatmenettel kapott megoldások ismeretében azonban egyszerűen módosíthatnánk egyes tervezési lépéseket a technológiai peremfeltételek teljesítése céljából.

Függelék

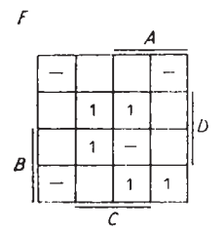
Gyakorló feladatok

F.1. Kombinációs hálózatok (2. fejezet)

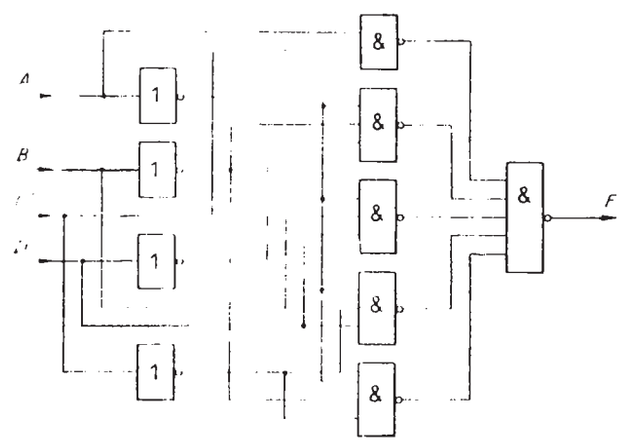
F.1.1. Az F.1. ábrán adott Karnaugh-tábla alapján írjuk fel a kombinációs hálózat legegyszerűbb hazárdmentes diszjunktív alakját!
Melyek a lényeges prímimplikánsok?
Írjuk fel a tagadott függvény legegyszerűbb hazárdmentes diszjunktív alakját!

F.1.2. Vizsgáljuk meg, hogy az F.2. ábrán megadott kétszintű kombinációs hálózat
— minimális-e?
— hazárdmentes-e? (ha nem, akkor végezzük el a hazárdmentesítést),
— szimmetrikus függvényt valósít-e meg?
(ha igen, melyek a szimmetriaszámok?)

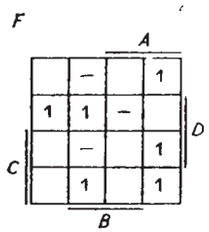
F.1.3. Írjuk fel a F.3. ábrán Karnaugh-táblával megadott logikai függvénynek és negáltjának a legegyszerűbb konjuktív alakját! Melyek a függvény lényeges



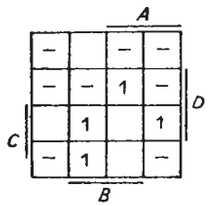
F.1. ábra. Az F.1.1. feladathoz megadott logikai függvény



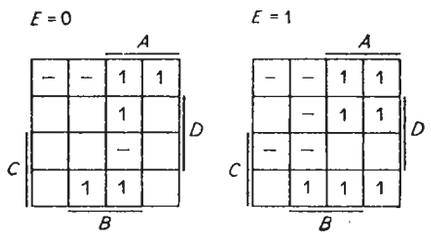
F.2. ábra. Az F.1.2. feladathoz megadott kombinációs hálózat elvi logikai rajza



F.3. ábra. Az F.1.3. feladathoz megadott logikai függvény



F.4. ábra. Az F.1.4. feladathoz megadott logikai függvény.



F.5. ábra. Az F.1.6. feladathoz megadott ötváltozós logikai függvény

prímimplikánsai? Vizsgáljuk meg, hogy létezik-e a függvénynek egyszerű diszjunkt dekompozíciója!

F.1.4. A közömbös értékek megfelelő rögzítésével szimmetrikussá tehető-e az F.4. ábrán megadott logikai függvény? Ha igen, akkor határozzuk meg a szimmetriaszámokat!

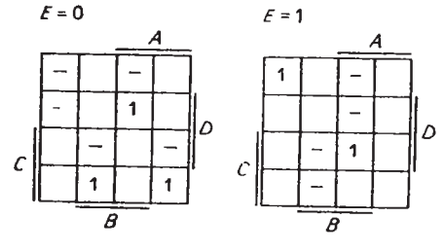
F.1.5. Rajzoljuk fel az F.4. ábrán megadott logikai függvényt megvalósító kombinációs hálózat legegyszerűbb kétszintű, NAND kapus, hazárdmentes elvi logikai rajzát!

F.1.6. Rajzoljuk fel az F.5. ábrán megadott logikai függvényt megvalósító kombinációs hálózat legegyszerűbb kétszintű, NAND kapus, hazárdmentes elvi logikai rajzát!

F.1.7. A közömbös értékek megfelelő rögzítésével szimmetrikussá tehető-e az F.6. ábrán megadott logikai függvény? Ha igen, akkor határozzuk meg a szimmetriaszámokat!

F.1.8. Rajzoljuk fel az alábbi logikai függvényt megvalósító kombinációs hálózat legegyszerűbb kétszintű, NAND kapus, hazárdmentes elvi logikai rajzát!

$$F = \sum^{6} [(0, 2, 14, 18, 21, 27, 32, 41, 49, 53, 62) + (6, 9, 23, 25, 34, 55, 57, 61)].$$



F.6. ábra. Az F.1.7. feladathoz megadott ötváltozós logikai függvény

A tengely szöghelyzete	Az érzékelőberendezés kimeneti kombinációja			
	E_3	E_2	E_1	E_0
0°– 30°	0	0	1	1
30°– 60°	0	0	1	0
60°– 90°	0	1	1	0
90°– 120°	0	1	1	1
120°– 150°	0	1	0	1
150°– 180°	0	1	0	0
180°– 210°	1	1	0	0
210°– 240°	1	1	0	1
240°– 270°	1	1	1	1
270°– 300°	1	1	1	0
300°– 330°	1	0	1	0
330°– 360°	1	0	1	1

F.7. ábra. Kódtáblázat az F.1.12. feladathoz

F.1.9. Rajzoljuk fel az alábbi logikai függvényt megvalósító kombinációs hálózat legegyszerűbb kétszintű, ÉS—VAGY kapus elvi logikai rajzát!

$$F = \sum^{5} [(3, 5, 11, 13, 23, 27, 31) + (7, 10, 17, 20)].$$

F.1.10. Rajzoljuk fel az alábbi logikai függvényt megvalósító kombinációs hálózat legegyszerűbb kétszintű, NOR kapus, hazárdmentes elvi logikai rajzát!

$$F = \sum^{6} (0, 2, 6, 14, 18, 21, 23, 27, 32, 41, 49, 53, 62).$$

F.1.11. Rajzoljuk fel az alábbi logikai függvényt megvalósító kombinációs hálózat legegyszerűbb kétszintű, NOR kapus, hazárdmentes elvi logikai rajzát!

$$F = \prod^{6} (0, 2, 6, 14, 18, 21, 23, 27, 32, 41, 49, 53, 62).$$

F.1.12. Egy tengely szöghelyzetét érzékelő berendezés kimenetén megjelenő kombináció az F.7. ábrán megadott módon 30°-os lépésekben jellemzi a szöghelyzetet. Tervezzük meg azt a kombinációs hálózatot, amely a kimenetén egy-

értelműen jelzi, ha a tengely 0° és 90° közötti szöghelyzetben van! Feltételezhetjük, hogy az F.7. ábrán nem szereplő kombinációk nem lépnek fel.

F.1.13. Egy logikai rendszer három funkcionális egysége egymástól függetlenül négy működési fázisban lehet. Tegyük fel, hogy a megfelelő vezérlés céljából jelezni kell, ha a három funkcionális egység közül bármely kettő egyidejűleg azonos működési fázisban van. E célból minden egyes funkcionális egységnek van két olyan kimenete, amelyen megjelenő kimeneti kombináció az illető funkcionális egység mindenkori működési fázisát jellemzi. Tervezzük meg azt a kombinációs hálózatot, amely elvégzi a megfelelő vezérléshez szükséges fenti kijelzést! Vizsgáljuk meg a kimeneti függvényt szimmetria és dekompozíció szempontjából! Kíséreljük meg a hálózat felépítését valamilyen több szintű alakzat szerint is!

F.1.14. Tervezzünk kombinációs hálózatot, amely egyértelmű kijelzést ad abban az esetben, ha a bemenetére jutó 10 bites kombinációban négy egymást közvetlenül követő 1-es érték van! Vizsgáljuk meg a kimeneti függvényt szimmetria és dekompozíció szempontjából! Kíséreljük meg a hálózat felépítését valamilyen több szintű alakzat szerint is!

F.1.15. Tervezzük meg azt a kiegészítő kombinációs hálózatot, amelynek segítségével az F.8. ábrán jellemzett építőelem felhasználásával előállítható a megvalósítandó függvény! Az eredményt hasonlítsuk össze a kétszintű megoldással!

F.1.16. Tervezzük meg az alábbi logikai függvényt megvalósító kombinációs hálózatot diszjunkt dekompozíciós alakzatban!

$$F = \sum^{5} [(2, 6, 8, 12, 16, 20, 26, 30)+(3, 4, 14)].$$

F.1.17. Vizsgáljuk meg az F.1.16. feladatban megadott logikai függvényt szimmetria szempontjából, és építsük fel az $S^5_{1,3}(x_1 \dots x_5)$ szimmetrikus függvénnyel jellemezhető építőelem felhasználásával!

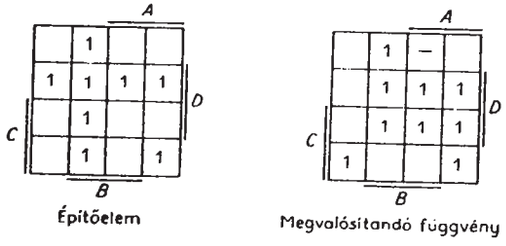
F.1.18. Vizsgáljuk meg, hogy az F.9. ábrán szereplő több szintű kombinációs hálózat hazárdmentes-e? Ha nem, akkor végezzük el a hazárdmentesítést!

F.1.19. Vizsgáljuk meg, hogy az F.10. ábrán megadott két kimenetű kombinációs hálózat

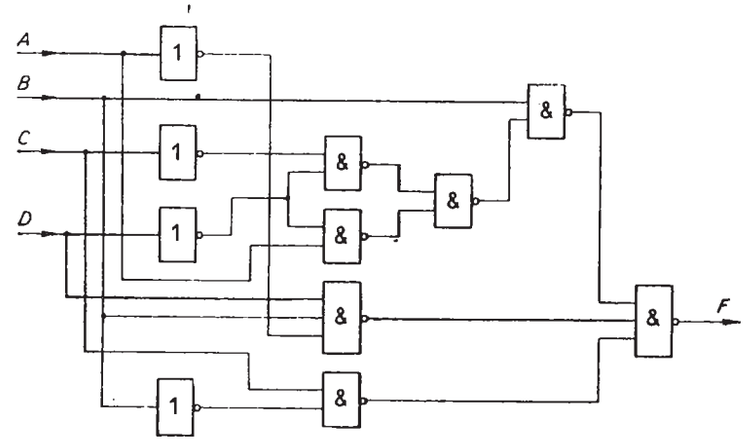
— minimális-e?

— kimenetenként hazárdmentes-e?

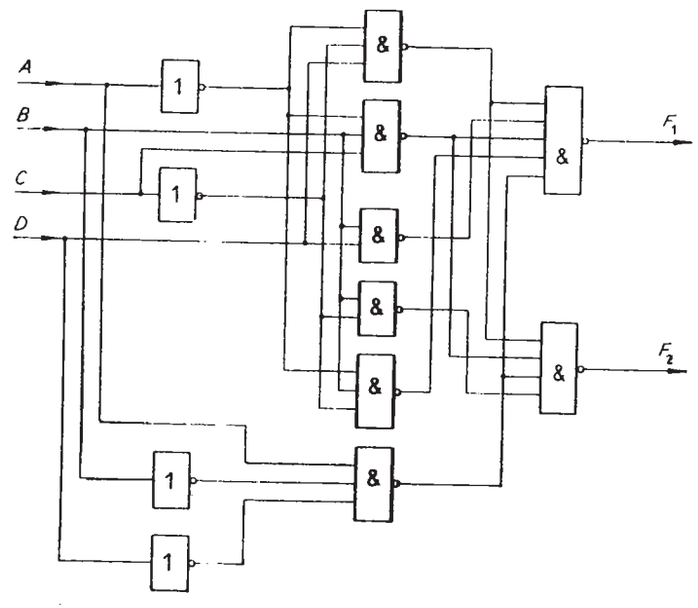
Ha nem, akkor végezzük el az egyszerűsítést, ill. a hazárdmentesítést!



F.8. ábra. Az építőelem és a megvalósítandó függvény Karnaugh-táblája az F.1.15. feladathoz

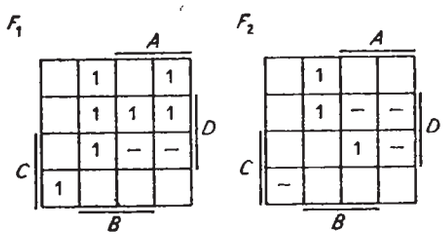


F.9. ábra. Több szintű kombinációs hálózat elvi logikai rajza az F.1.18. feladathoz



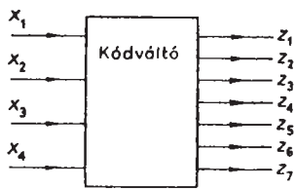
F.10. ábra. Az F.1.19. feladathoz megadott kombinációs hálózat elvi logikai rajza

F.1.20. Határozzuk meg az F.11. ábrán Karnaugh-táblával megadott két kimeneti függvényt megvalósító kombinációs hálózat legegyszerűbb kétszintű, NAND kapus elvi logikai rajzát!



F.11. ábra. Az F.1.20. feladathoz megadott logikai függvény

x_1	x_2	x_3	x_4	z_1	z_2	z_3	z_4	z_5	z_6	z_7
0	0	0	0	1	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	1	0
0	1	0	0	0	0	1	0	0	1	0
1	1	0	0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	0	1	1	0
1	0	1	0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	0	0	0	1
1	1	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0	1	0	1



F.12. ábra. Az F.1.23. feladatban szereplő kódváltó működésének leírása

F.1.21. Határozzuk meg az alábbi kimeneti függvényeket megvalósító kombinációs hálózat legegyszerűbb kétszintű, ÉS—VAGY kapus elvi logikai rajzát!

$$F_1 = \sum^{4} (1, 5, 7, 8, 12, 13, 14, 15),$$

$$F_2 = \sum^{4} (0, 1, 5, 8, 14),$$

$$F_3 = \sum^{4} (0, 2, 3, 6, 8, 9, 10, 11, 14).$$

F.1.22. Határozzuk meg az alábbi kimeneti függvényeket megvalósító kombinációs hálózat legegyszerűbb kétszintű, NAND kapus elvi logikai rajzát!

$$F_1 = \sum^{4} [(0, 2, 9, 10)+(1, 8, 13)],$$

$$F_2 = \sum^{4} [(1, 3, 5, 13)+(0, 7, 9)],$$

$$F_3 = \sum^{4} [(2, 8, 10, 11, 13)+(3, 9, 15)].$$

F.1.23. Tervezzük meg az F.12. ábrán megadott kódváltást megvalósító kombinációs hálózatot! Feltételezhetjük, hogy az F.12. ábrán nem szereplő bemeneti kombinációk nem lépnek fel.

F.1.24. Tervezzünk ún. komparátorhálózatot, melynek két bemenete (x_1, x_2) és három kimenete (F_1, F_2, F_3) van! A megoldandó logikai feladat az F.13. ábrán vázolt módon az, hogy a hálózat a kimenetein jelezze ki az x_1 és x_2 bemenetekre jutó logikai értékek közötti nagyságrendet, vagyis

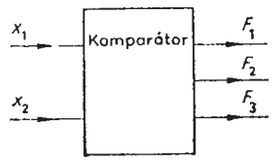
$F_1=1$, ha $x_1>x_2$;

$F_2=1$, ha $x_1=x_2$;

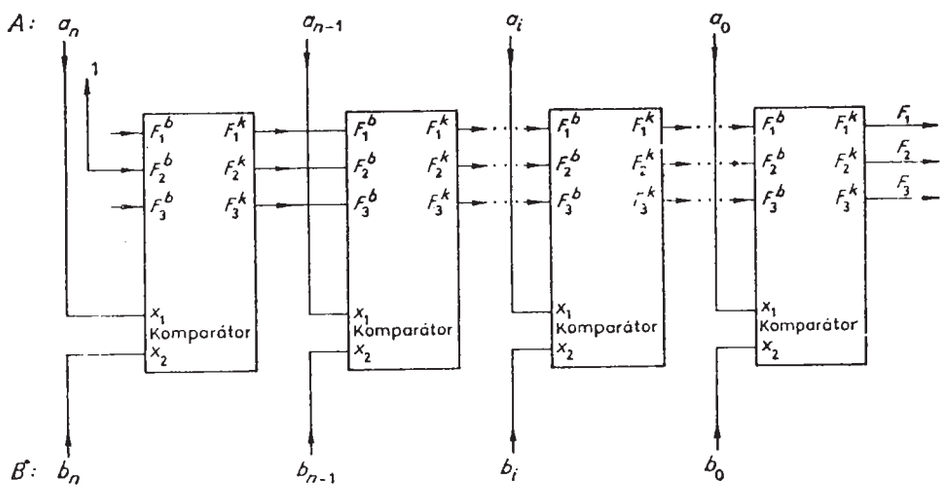
$F_3=1$, ha $x_1<x_2$.

F.1.25. Az F.1.24. feladat megoldásaként kapott komparátort tegyük alkalmassá ún. kaszkád kapcsolásban két, tetszőleges számú bitet tartalmazó bináris szám közötti nagyságrend kijelzésére! Az F.14. ábra szemlélteti a kaszkád kapcsolás lényegét. A két összehasonlítandó bináris szám (*A* és *B*) egyes bitjeit rendre a_n, ill. b_n, a_{n-1}, ill. b_{n-1}, a_i, ill. b_i, a_0, ill. b_0, jelöli. A legmagasabb helyértékű bit a_n, ill. b_n. Az ábra szerint a két szám összehasonlítása a legmagasabb helyértéken kezdődik. Ha ezek a bitek azonosak, akkor a nagyságrend eldöntéséhez a további bitek vizsgálata szükséges. A legmagasabb helyértéken működő komparátor ilyenkor az F.1.24. feladatnak megfelelően

x_1	x_2	F_1	F_2	F_3
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0



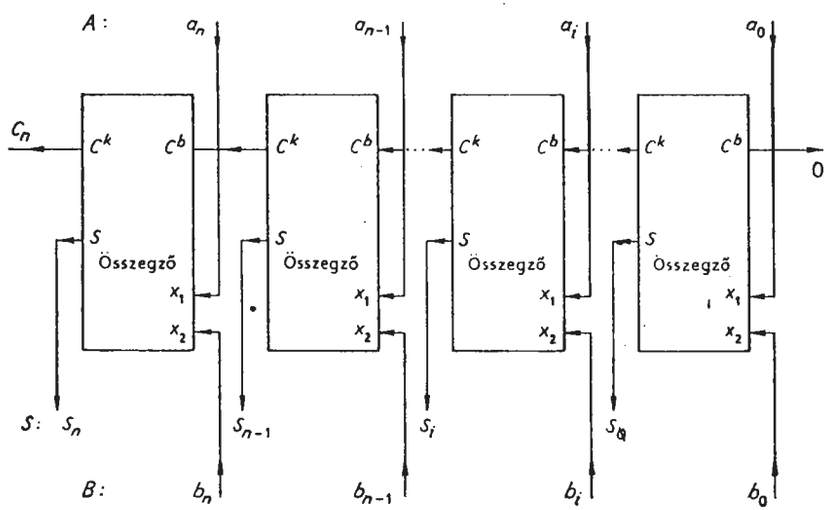
F.13. ábra. Az F.1.24. feladatban szereplő komparátor működésének leírása



F.14. ábra. A komparátorok kaszkád kapcsolásának vázlata az F.1.25. feladathoz

$F_1^k F_2^k F_3^k = 010$ kimeneti kombinációt állít elő. Az $n-1$-edik helyértéken levő komparátor tehát az $F_2^b = 1$ bemeneti jelérték alapján értesül arról, hogy a nagyságrend még nem dőlt el és így az $n-1$-edik helyértéken is el kell végezni az összehasonlítást. Az alacsonyabb helyértékek felé haladva mindaddig így folytatódik a működés, amíg valamelyik komparátor kimeneti kombinációjában $F_2^k = 0$ fel nem lép, amely az F.1.24. feladatnak megfelelően $F_1^k = 1$ vagy $F_3^k = 1$ értékkel párosul. Ez azt jelenti, hogy az adott helyértéken eldőlt a nagyságrend. A kaszkád kapcsolás kimenetén tehát az $F_1 F_2 F_3$ kombinációnak ezt kell jeleznie. Az egyes komparátoroknak e célból úgy kell működniük, hogy $F_2^b = 0$ esetén ne végezzenek összehasonlítást az x_1 és x_2 értékek között, hanem az $F_1^b F_2^b F_3^b$ kombinációt változtatás nélkül továbbítsák az $F_1^k F_2^k F_3^k$ kimenetekre. Így ugyanis a kaszkád kapcsolás révén a nagyságrendet eldöntő komparátor kimeneti kombinációja változtatás nélkül halad át az alacsonyabb helyértékekre kapcsolt komparátorokon keresztül a legkisebb helyértéken működő komparátor kimenetéig, amely így a kaszkád kapcsolással felépített teljes komparátorrendszer kimenetének tekinthető. A leírt működés alapján könnyen belátható, hogy a legmagasabb helyértéken levő (n-edik) komparátor F_2^b bemenetét állandóan 1 értéken kell tartani. Így ennek a komparátornak az F_1^b és F_3^b bemetére jutó jelek értékei nyilvánvalóan közömbösek. (Megjegyezzük, hogy a kaszkád kapcsolás elvét a legkisebb helyértéken kezdődő összehasonlítás esetén is alkalmazhatjuk a koparátoregység felépítésekor.) Tervezzük meg az F.14. ábrán vázolt komparátoregységet a fenti működési előírás alapján!

F.1.26. Tervezzünk ún. összegzőhálózatot, amely az F.15. ábrán vázolt módon elvégzi két n helyértékű bináris szám (A és B) összeadását. A két szám egyes



F.15. ábra. Összegző hálózat vázlata az F.1.26. feladathoz

bitjeit rendre a_n, ill. b_n, a_{n-1}, ill. b_{n-1}, ..., a_i, ill. b_i, ..., a_0, ill. b_0 jelöli. A legmagasabb helyértékű bit a_n, ill. b_n. Az összegként keletkező bináris szám (S) bitjei S_n, S_{n-1}, ..., S_i, ... S_0. Az ábra szerint összekapcsolt összegzőegységek mindegyike előállítja a megfelelő összegbitet az előző helyértéken keletkezett átvitel (C^b) figyelembevételével, és képzi az adott helyértéken keletkező átvitelt (C^k), amely a következő helyértéken veendő figyelembe az összeadáskor. A fenti működési előírásból következően a legkisebb helyértéken levő összegzőegység C^b bemenetére állandóan 0 értéknek kell jutnia.

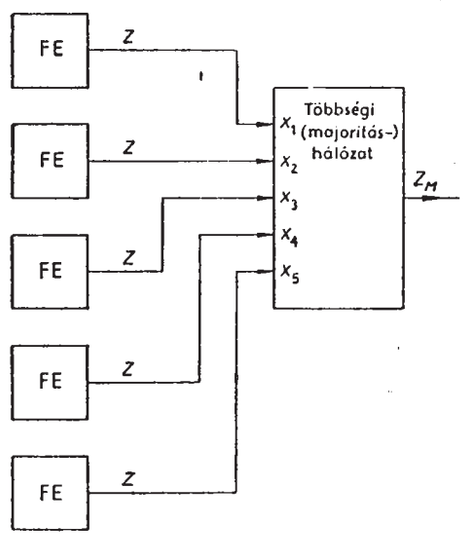
Tervezzük meg az F.15. ábrán alkalmazott összegzőegységet!

F.1.27. Egészítsük ki az F.1.26. feladat szerinti összegzőegységet egy ún. üzemmódvezérlő-bemenettel, amelyre adott logikai értékkel beállítható a végrehajtandó művelet összeadásra vagy kivonásra! Tervezzük meg az így kialakuló műveleti egységet, amely az F.15. ábrán vázolt módon alkalmas az

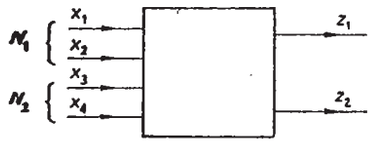
$$S = A + B \quad \text{vagy az} \quad S = A - B$$

műveletet végrehajtó rendszer felépítésére! (Feltételezhetjük, hogy $A \geqq B$ mindig fennáll.)

F.1.28. A funkcionális egységek meghibásodásának következményei ellen bizonyos mértékig védelmet nyújt, ha a különlegesen nagy megbízhatósági igényű logikai rendszerekben a hibakövetkezmény szempontjából veszélyes funkcionális egységeket vagy részeiket több példányban építik meg. Az F.16. ábrán vázolt módon az öt példányban megépített FE egység Z kimeneti jelének hibás értéke mindaddig hatástalan marad, amíg az öt példány közül a többség helyes kimeneti értéket szolgáltat. Ezt az ún. többségi (majoritás)



F.16. ábra. A többségi (majoritás) hálózat alkalmazásának vázlata az F.1.28. feladathoz



F.17. ábra. Az F.1.29. és F.1.30. feladatban megtervezendő kombinációs hálózat vázlata

hálózat biztosítja. Ez a kombinációs hálózat a Z_M kimenetén mindig azt az értéket állítja elő, amely az $x_1, \ldots, x_5$ bemenetek többségén van jelen. Tervezzük meg az F.16. ábrán vázolt többségi (majoritás) hálózatot! Vizsgáljuk meg a többségi (majoritás) hálózat által megvalósított logikai függvényt szimmetria szempontjából! Kíséreljük meg a felépítést diszjunkt dekompozícióval, ill. szimmetrikus építőelemekből!

F.1.29. Az F.17. ábrán vázolt négybemenetű, kétkimenetű kombinációs hálózat x_1 és x_2, valamint x_3 és x_4 bemeneteire jutó jelek egy-egy kétbites bináris számot képviselnek, amelyek decimális megfelelőit N_1: x_1x_2 és N_2: x_3x_4 jelöli. Tervezzük meg a kombinációs hálózatot, ha a megoldandó logikai feladat szerint

— z_1 akkor, és csak akkor legyen 1 értékű, ha $N_2^{N_1}$ páros;

— z_2 akkor, és csak akkor legyen 1 értékű, ha $N_1^{N_2}$ páratlan!

Szimmetrikus-e valamelyik kimeneti függvény? Létezik-e valamelyiknek egyszerű diszjunkt dekompozíciója?

F.1.30. Tervezzük meg az F.17. ábrán vázolt kombinációs hálózatot, ha a megoldandó logikai feladat szerint

— z_1 akkor, és csak akkor 1 értékű, ha $N_2 - N_1 \geqq 0$;

— z_2 akkor, és csak akkor 1 értékű, ha $N_2 - N_1 \leqq 0$!

A kimeneti függvényeket vizsgáljuk meg szimmetria szempontjából!

F.1.31. Hány helyértékű bináris számok összehasonlítására alkalmas komparátort tudunk megvalósítani az F.1.25. feladatban vázolt működést feltételezve egyetlen olyan PLA elemmel, amelynek 16 bemenete, 16 kimenete és 20 ÉS kapuja van?

F.1.31. Építsük fel az F.1.27. feladatban leírt műveleti egységet olyan PLA elemekből, amelyeknek 16 bemenetük, 8 kimenetük és 16 ÉS kapujuk van!

F.2. Sorrendi hálózatok (3. fejezet)

F.2.1. Végezzük el az F.18. ábrán szereplő előzetes aszinkron állapottábla összevonását! Válasszunk állapotkódokat és vizsgáljuk meg a hálózatot lényeges hazárd szempontjából! Milyen szinkron flip-flopot valósíthatunk meg a hálózattal, ha az x_2 bemenetet tekintjük órajelbemenetnek?

$y \backslash x_1x_2$	00	01	11	10
a	(a) 0	b 0	–	c 0
b	a 0	(b) 0	d 0	–
c	a 0	–	e –	(c) 0
d	–	f 0	(d) 0	c 0
e	–	h 1	(e) 1	g 1
f	a 0	(f) 0	d 0	–
g	i 1	–	d –	(g) 1
h	i 1	(h) 1	e 1	–
i	(i) 1	h 1	–	g 1

F.18. ábra. Állapottábla az F.2.1. feladathoz

$y \backslash x$	0	1
a	a 0	a 0
b	a 1	f 1
c	d 1	e 0
d	a 1	g 0
e	b 0	c 0
f	d 0	e 0
g	b 0	c 0

F.19. ábra. Állapottábla az F.2.3. feladathoz

F.2.2. Tervezzünk kétbemenetű (x_1, x_2), egykimenetű (z) szinkron sorrendi hálózatot, amelynek előírt működése az alábbi!

— z értéke akkor, és csak akkor legyen 1, ha x_1, x_2, x_1 sorrendben jut 1 érték a bemenetekre három egymást követő bemeneti kombinációban. Tervezzük meg a Mealy- és a Moore-modell szerint működő változatot!

F.2.3. Végezzük el az F.19. ábrán szereplő szinkron állapottábla összevonását! Válasszuk meg az állapotkódokat, és valósítsuk meg a hálózatot T flip-flop felhasználásával!

F.2.4. Tervezzünk kétbemenetű (x_1, x_2) egykimenetű (z) aszinkron sorrendi hálózatot, amelynek előírt működése az alábbi!

— z értéke akkor, és csak akkor legyen 1, ha a bemeneten $x_1x_2 = 10$ kombináció van jelen, feltéve, hogy ez a bemeneti kombináció az $x_1x_2 = 00, 01, 11$ kombinációsorozatot közvetlenül követve lépett fel.

A hálózat megvalósítását visszacsatolt kombinációs hálózattal végezzük, és vizsgáljuk meg ezt a lényeges hazárd szempontjából!

F.2.5. Tervezzünk kétbemenetű (x_1x_2) kétkimenetű (z_1, z_2) szinkron sorrendi hálózatot, amelynek előírt működése az alábbi!

— z_1 értéke akkor, és csak akkor legyen 1, ha legutoljára x_1 változott és x_1 összes addigi változásainak száma páratlan;

— z_2 értéke akkor, és csak akkor legyen 1, ha legutoljára x_2 változott és x_2 összes addigi változásainak száma páros.

Feltételezhetjük, hogy x_1 és x_2 nem változik egyidejűleg. A hálózat megvalósítását D flip-floppal végezzük!

F.2.6. Írjuk fel az F.2.5. feladatban előírt működésű aszinkron sorrendi hálózat előzetes állapottábláját, és végezzük el az állapotösszevonásokat!

F.2.7. Egyszerűsítsük az F.20. ábrán szereplő szinkron állapottáblát, válasszuk meg az állapotkódokat, és valósítsuk meg a hálózatot J—K flip-floppal!

F.2.8. Alakítsuk át az F.2.7. feladatban szereplő hálózatot Moore-modellé!

F.2.9. Tervezzünk kétbemenetű (x_1, x_2), egykimenetű (z) Mealy-modell szerint működő szinkron sorrendi hálózatot, amelynek előírt működése az alábbi:

— A két bemenetre jutó jelek kétbites bináris számnak tekintendők, amelynek decimális megfelelőjét jelöljük N-nel. A z kimeneten lépjen fel 1 érték, ha egy bemeneti változás N mindenkori értékét 1-gyel növeli. Ha a bemeneti változás 1-gyel csökkenti N mindenkori értékét, z változtassa meg ellenkezőjére pillanatnyi értékét!

F.2.10. Írjuk fel az F.2.9. feladatban előírt működésű aszinkron sorrendi hálózat előzetes állapottábláját, végezzük el az állapotösszevonást, és válasszunk állapotkódokat!

F.2.11. Alakítsuk át az F.2.9. feladatban szereplő hálózatot Moore-modellé!

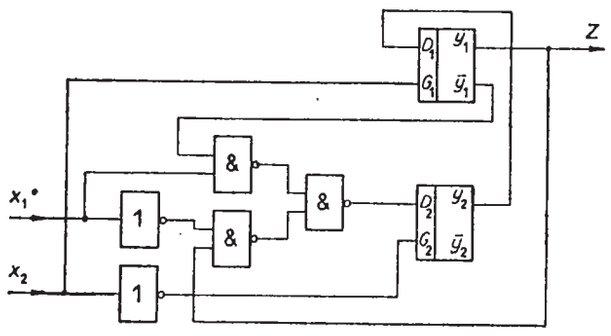
F.2.12. Végezzük el az F.21. ábrán szereplő aszinkron sorrendi hálózat analízisét!

— Egyszerűsíthető-e a hálózat?

— Tartalmaz-e kritikus versenyhelyezetet és lényeges hazárdot?

y \ x_1x_2	00	01	11	10
a	c 0	e 1	–	–
b	c 0	e –	–	b
c	b 0	c 0	a –	–
d	b –	c –	e –	–
e	–	e 0	a –	–

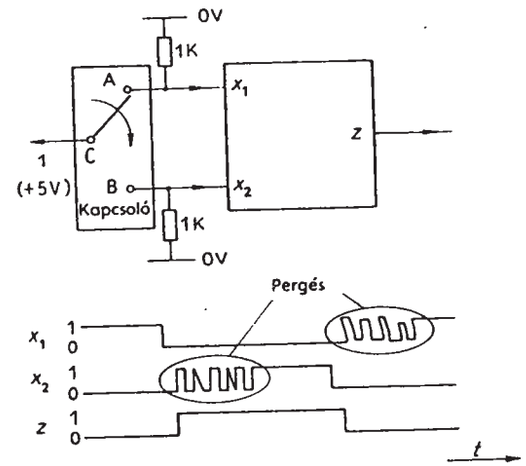
F.20. ábra. Állapottábla az F.2.7. feladathoz



F.21. ábra. Aszinkron sorrendi hálózat elvi logikai rajza az F.2.12. feladathoz

y \ x_1x_2	00	01	11	10
A	B –	C 0	–	D –
B	–	E 0	–	–
C	D –	F –	C –	–
D	E –	C 0	–	A –
E	–	F 0	–	–
F	–	– 1	D –	B –

F.22. ábra. Állapottábla az F.2.13. feladathoz



F.23. ábra. Pergésmentesítő aszinkron sorrendi hálózat vázlata az F.2.14. feladathoz

— Milyen szinkron flip-flopot valósíthatunk meg a hálózattal, ha az x_2 bemenetet tekintjük órajelbemenetnek?

— Hogyan biztosítható a hálózat alaphelyzete bekapcsoláskor?

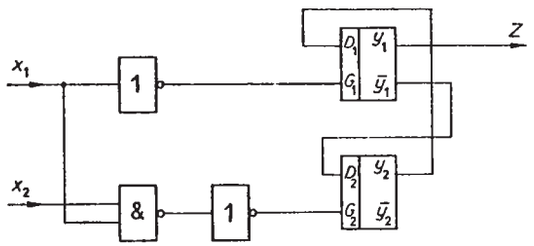
F.2.13. Egyszerűsítsük az F.22. ábrán szereplő szinkron állapottáblát, válasszunk állapotkódokat, és valósítsuk meg a hálózatot D flip-flop felhasználásával!

F.2.14. Tervezzük meg az F.23. ábrán vázolt ún. pergésmentesítő aszinkron sorrendi hálózatot! Az x_1 és x_2 bemenetekre jutó jelek értékét a kapcsoló állása határozza meg. Ha például a kapcsoló az A és C pontokat köti össze, akkor az állandóan 1 értékre kapcsolt C pont miatt az x_1 bemenetre is 1 jut. Ekkor a B pontra csak a 0 V feszültségérték hat kívülről az ellenálláson keresztül, miáltal az x_2 bemeneten a 0 logikai értéknek megfelelő feszültségérték jön létre. Tételezzük fel, hogy az F.23. ábra idődiagramján vázolt módon a kapcsoló érintkezőinek csak a záródáskor van pergésük. A feladat az, hogy a z kimeneten a kapcsoló oda-vissza kapcsolásakor ne jelentkezzen pergés, vagyis az idődiagram szerinti négyszögimpulzust állítsa elő a hálózat. A hálózatot NAND kapukból felépített S—R flip-flop felhasználásával valósítsuk meg!

F.2.15. Egyszerűsítsük az F.24. ábrán szereplő aszinkron előzetes állapottáblát!

$y \backslash x_1x_2$	00	01	11	10
a	a 0	b 0	–	e 0
b	–	b 0	c 0	–
c	–	d –	c 0	i 0
d	a –	d 1	–	–
e	–	–	f 0	e 0
f	–	g 0	f 0	h 0
g	a 0	g 0	–	–
h	a 0	–	–	h 0
i	a 0	–	–	i 0

F.24. ábra. Állapottábla az F.2.15. feladathoz



F.25. ábra. Aszinkron sorrendi hálózat elvi logikai rajza az F.2.18. feladathoz

Válasszunk állapotkódokat, és valósítsuk meg a hálózatot D—G flip-flop felhasználásával!

F.2.16. Tervezzünk kétbemenetű (x_1, x_2), kétkimenetű (z_1, z_2) Mealy-modell szerint működő aszinkron sorrendi hálózatot, amelynek előírt működése az alábbi:
— $z_1z_2=01$ akkor, és csak akkor lépjen fel, ha a mindenkori bemeneti kombinációt megelőzően fennálló két bemeneti kombinációban $x_2=1$ volt;
— $z_1z_2=10$ akkor, és csak akkor lépjen fel, ha a mindenkori bemeneti kombinációt megelőzően fennálló két bemeneti kombinációban $x_1=1$ volt.
A fenti két feltétel egyidejű teljesítése esetén tekintsük x_2 hatását elsődlegesnek;
— Minden más esetben fennállhat akár $z_1z_2=11$, akár $z_1z_2=00$.

F.2.17. Tervezzük meg az F.2.16. feladatban előírt működésű sorrendi hálózatot szinkron változatban!

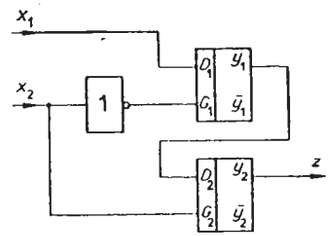
F.2.18. Végezzük el az F.25. ábrán szereplő aszinkron sorrendi hálózat analízisét az F.2.12. feladatban megadott szempontok szerint!

F.2.19. Válasszunk állapotkódokat az F.26. ábrán megadott szinkron állapottáblához, törekedve a kialakuló hálózat egyszerűségére! Valósítsuk meg a hálózatot T flip-flop felhasználásával!

F.2.20. Tervezzünk kétbemenetű (x_1, x_2) egykimenetű (z) Moore-modell szerint működő szinkron sorrendi hálózatot, amelynek előírt működése az alábbi:

$y \backslash x$	0	1
A	B 0	B 1
B	F 0	D 1
C	E 1	G 1
D	A 0	C 0
E	D 1	G 0
F	F 0	A 0
G	C 1	B 0

F.26. ábra. Állapottábla az F.2.19. feladathoz



F.27. ábra. Aszinkron sorrendi hálózat elvi logikai rajza az F.2.21. feladathoz

$y \backslash x_1x_2$	00	01	11	10
A	A 0	A 0	B 0	A 0
B	–	D –	B 0	D –
C	–	A –	C 1	A –
D	D 1	D 1	C 1	D 1

F.28. ábra. Állapottábla az F.2.23. feladathoz

— A z kimeneten csak minden harmadik bemeneti kombináció tartama alatt lehet 1 érték, de csak akkor, ha a három bemeneti kombinációban az x_1 bemenetre érkezett 1 értékek száma páratlan.
— Ha az x_2 bemenetre 1 érték jut, akkor z váljon 0 értékűvé.
A hálózat megvalósításához használjunk D flip-flopot!

F.2.21. Végezzük el az F.27. ábrán szereplő aszinkron sorrendi hálózat analízisét az F.2.12. feladatban megadott szempontok szerint!

F.2.22. Egyszerűsíthető-e az alábbi logikai függvényekkel adott aszinkron sorrendi hálózat? Tartalmaz-e kritikus versenyhelyzetet és lényeges hazárdot?

$$Y_1 = y_1x_1+y_2x_1+\bar{x}_1x_2,$$

$$Y_2 = y_2x_1+y_1x_1\bar{x}_2+\bar{y}_1x_1x_2,$$

$$z = y_1.$$

F.2.23. Válasszunk állapotkódokat az F.28. ábrán megadott aszinkron állapottáblához, és valósítsuk meg a hálózatot D—G flip-flop felhasználásával!

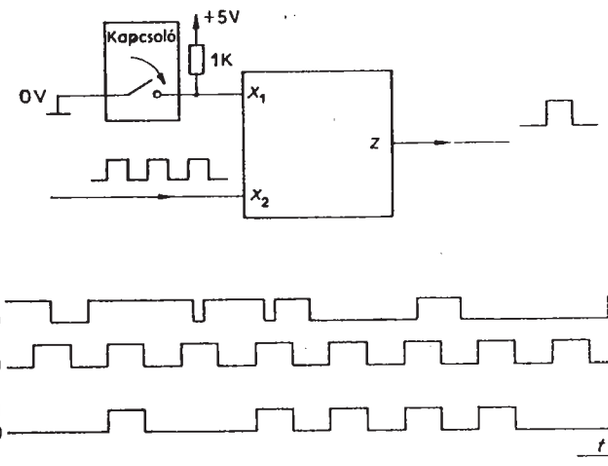
F.2.24. Tervezzünk kétbemenetű (x_1, x_2), egykimenetű (z) Moore-modell szerint működő aszinkron sorrendi hálózatot, amelynek előírt működése az F.29. ábra alapján az alábbi!

— A kapcsoló zásárának hatására (x_1 értéke 0-vá válik) a z kimeneten jelenjen meg az x_2 bemenetre jutó állandó órajelnek a kapcsoló zárásának pillanatától számított első *teljes* impulzusa.

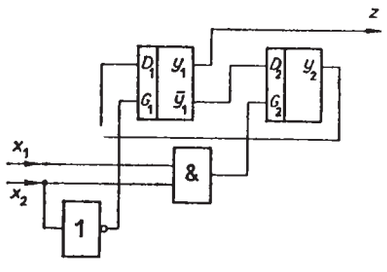
— Feltételezhetjük, hogy a kapcsoló zárási időpontjai között legalább órajel periódusnyi idő telik el.

— A kapcsolót pergésmentesnek tételezhetjük fel. Az előírt működést az F.29. ábra idődiagramja szemlélteti.

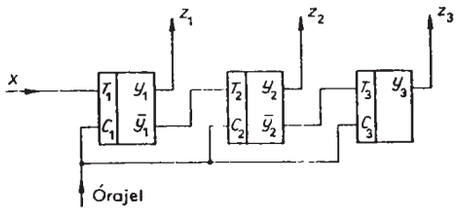
F.2.25. Alakítsuk át az F.30. ábrán megadott aszinkron sorrendi hálózatot visszacsatolt kombinációs hálózattá! Határozzuk meg a kimeneti jelsorozatot, ha a bemeneten az $y_1y_2=00$ állapotból kiindulva az
$x_1x_2=00, 01, 00, 10, 11, 01, 00, 10, 11$
kombinációsorozat játszódik le! Milyen szinkron flip-flopot valósíthatunk meg a hálózattal, ha az x_2 bemenetet tekintjük órajelbemenetnek? Tartalmaz-e a hálózat lényeges hazárdot?



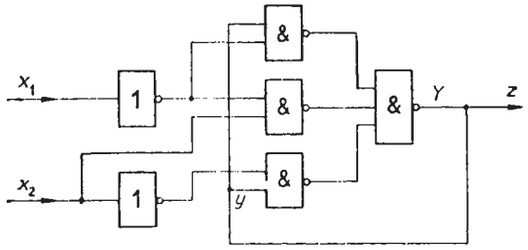
F.29. ábra. Szemléltető ábra az F.2.24. feladat működési előírásához



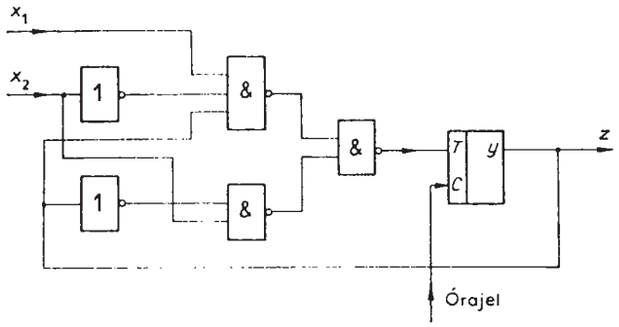
F.30. ábra. Aszinkron sorrendi hálózat elvi logikai rajza az F.2.25. feladathoz



F.31. ábra. Szinkron sorrendi hálózat elvi logikai rajza az F.2.26. feladathoz



F.32. ábra. Aszinkron sorrendi hálózat elvi logikai rajza az F.2.28. feladathoz



F.33. ábra. Szinkron sorrendi hálózat elvi logikai rajza az F.2.29. feladathoz

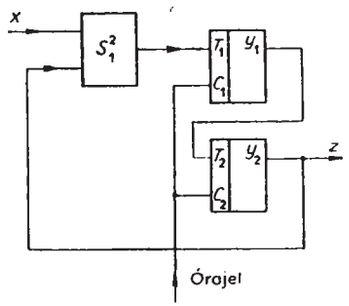
F.2.26. Határozzuk meg az F.31. ábrán megadott szinkron sorrendi hálózat által szolgáltatott kimeneti kombinációsorozatot, ha az $y_1y_2y_3=000$ állapotból kiindulva a bemeneten az
$x = 0, 1, 0, 1, 1, 1, 0, 1, 0$ jelsorozat játszódik le!

F.2.27. Felléphetnek-e hazárdjelenségek az F.31. ábrán szereplő szinkron sorrendi hálózatban, ha az alkalmazott flip-flopok egyszerű élvezérelt, vagy master—slave működésűek?

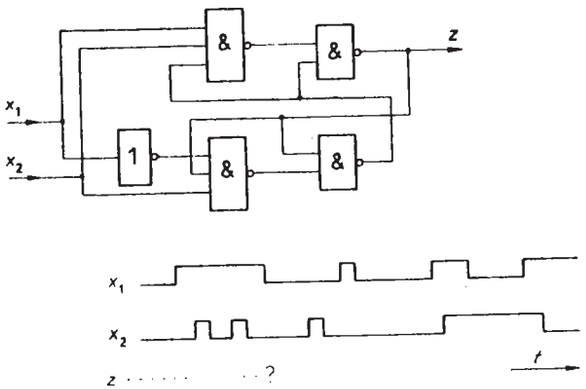
F.2.28. Egyszerűsíthető-e az F.32. ábra szerinti visszacsatolt kombinációs hálózattal felépített aszinkron sorrendi hálózat? Határozzuk meg a hálózat elvi logikai rajzát D—G flip-flop alkalmazásával!

F.2.29. Építsük fel az F.33. ábrán megadott szinkron sorrendi hálózatot J—K flip-flop felhasználásával! Felléphetnek-e hazárdjelenségek a hálózatban egyszerű élvezérelt működésű flip-flop alkalmazása esetén?

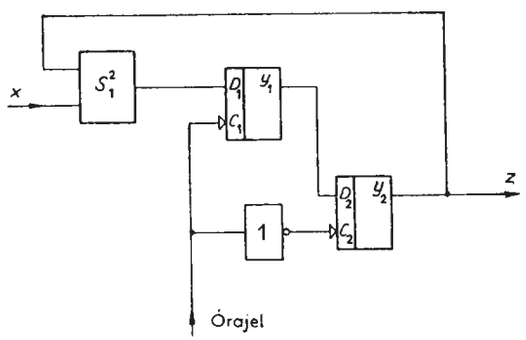
F.2.30. Szerkesszük meg a reteszelt működésű (data-lock-out) D flip-flopot megvalósító aszinkron sorrendi hálózat előzetes állapottábláját! Végezzük el az



F.34. ábra. Szinkron sorrendi hálózat elvi logikai rajza az F.2.31. feladathoz



F.35. ábra. Elvi logikai rajz és idődiagram az F.2.32. feladathoz



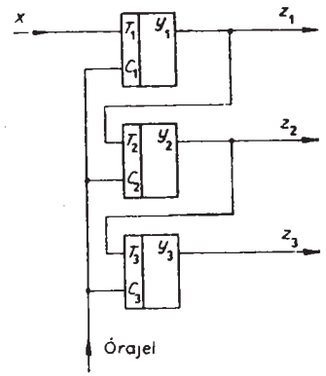
F.36. ábra. Szinkron sorrendi hálózat elvi logikai rajza az F.2.33. feladathoz

állapot-összevonásokat, és válasszunk állapotkódokat a lehető legnagyobb működési sebesség elérését peremfeltételnek tekintve!

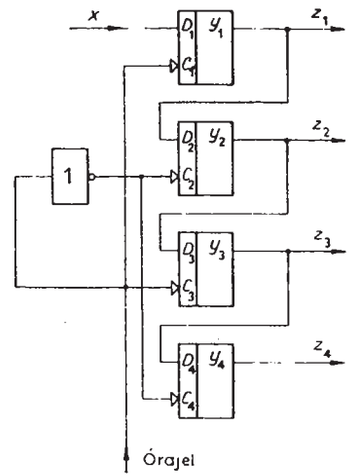
F.2.31. Felléphetnek-e hazárdjelenségek az F.34. ábrán szereplő szinkron sorrendi hálózatban master—slave működésű flip-flopok alkalmazása esetén?

F.2.32. Egyszerűsíthető-e az F.35. ábrán szereplő aszinkron sorrendi hálózat? Rajzoljuk fel *z* időbeli változását az ábrázolt bemeneti változások hatására!

F.2.33. Felléphetnek-e hazárdjelenségek az F.36. ábrán látható szinkron sorrendi há-



F.37. ábra. Szinkron sorrendi hálózat elvi logikai rajza az F.2.34. feladathoz



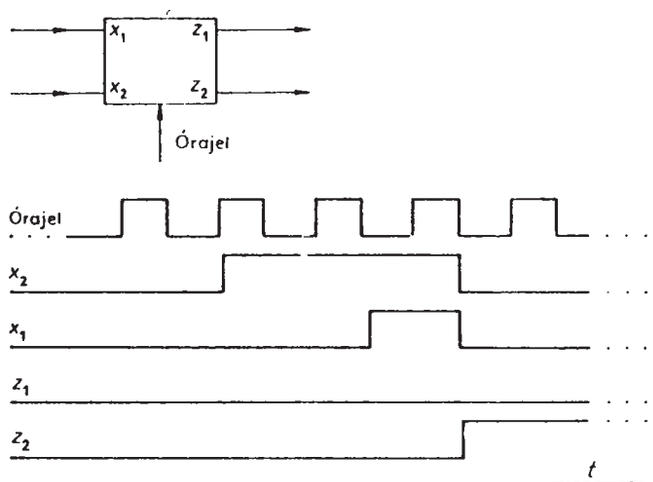
F.38. ábra. Szinkron sorrendi hálózat elvi logikai rajza az F.2.35. feladathoz

lózatban, ha az alkalmazott flip-flopok egyszerű élvezérelt működésűek? Milyen típusú és működési elvű szinkron flip-floppal egyenértékű a hálózat?

F.2.34. Rajzoljuk fel az F.37. ábrán szereplő szinkron sorrendi hálózat elvi logikai rajzát D flip-flopok felhasználásával! Határozzuk meg a kimeneti kombinációsorozatot, ha *x* értéke állandóan 1 és $z_1z_2z_3=000$ a kiindulási kimeneti kombináció! Felléphetnek-e hazárdjelenségek a hálózatban egyszerű élvezérelt vagy master—slave működésű flip-flopok alkalmazása esetén?

F.2.35. Rajzoljuk fel az F.38. ábrán szereplő szinkron sorrendi hálózat elvi logikai rajzát T flip-flopok felhasználásával! Határozzuk meg a kimeneti kombinációsorozatot, ha *x* értéke állandóan 1 és $z_1z_2z_3z_4=0000$ a kiindulási kimeneti kombináció! Felléphetnek-e hazárdjelenségek a hálózatban egyszerű élvezérelt vagy master—slave működésű flip-flopok alkalmazása esetén?

F.2.36. Tervezzünk impulzushelyzet-azonosító szinkron sorrendi hálózatot, amely Moore-modell szerint működik és működési előírása az F.39. ábra alapján az alábbi!



F.39. ábra. Az F.2.36. feladatban előírt impulzushelyzet-azonosító működését szemléltető idődiagram

— Az x_2 jel 1 értéke — ha fellép — három órajel-periódus időtartamára áll fenn.
— Az x_1 jel 1 értéke — ha fellép — egyetlen órajel-periódus időtartamára áll fenn.

A hálózat feladata annak jelzése, hogy $x_2=1$ idején fellépett-e $x_1=1$ és ha igen, akkor az $x_2=1$ alatti három órajel-periódus közül melyiknek a tartama alatt.

— A kimeneti kombináció értelmezése legyen az alábbi:

$z_1 z_2$
00 $x_2=1$ idején $x_1=1$ nem lépett fel;
01 $x_2=1$ idején $x_1=1$ a bal oldali órajel-periódus tartama alatt állt fenn;
10 $x_2=1$ idején $x_1=1$ a jobb oldali órajel-periódus tartama alatt állt fenn;
11 $x_2=1$ idején $x_1=1$ a középső órajel-periódus tartama alatt állt fenn.

— z_1 és z_2 értéke $x_2=1$ alatt ne változzon, de $x_2=0$ megjelenésekor azonnal álljon be az előírt kimeneti kombináció.

Az F.39. ábra idődiagramja a $z_1 z_2=01$ kimeneti kombináció előállításának feltételét szemlélteti.

F.2.37. Tervezzük meg az F.2.36. feladatban előírt működésű impulzushelyzet-azonosítót Moore-modell szerint működő aszinkron sorrendi hálózatként! Szerkesszük meg az előzetes állapottáblát, végezzük el az állapot összevonásokat, és válasszunk állapotkódokat a lehető legnagyobb működési sebesség elérését peremfeltételnek tekintve!

F.2.38. Vizsgáljuk meg az F.2.37. feladatban adódott összevont állapottáblát dekompozíció szempontjából! Kíséreljük meg a hálózat felépítését aszinkron sorrendi hálózatként felhasználva az egyszerű élvezérelt D flip-flopot!

F.2.39. Tervezzünk ún. soros paritásgenerátort egybemenetű (x), egykimenetű (z) Moore-modell szerint működő szinkron sorrendi hálózatként, amelynek előírt működése az alábbi!

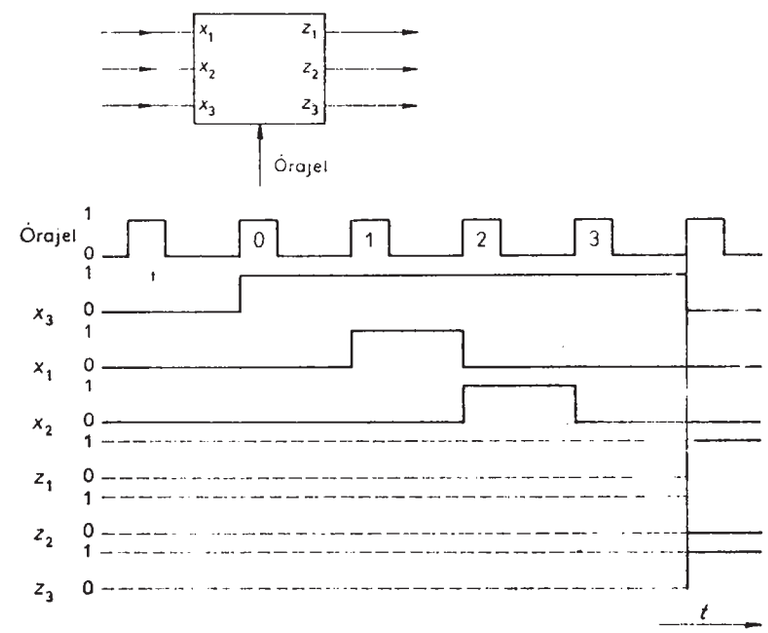
— Minden negyedik óraimpulzus hatására egyetlen órajel-periódus idejére jelenjen meg 1 érték a z kimeneten, de csak akkor, ha a négy óraimpulzussal páros számú 1 érték jutott a bemenetre.

F.2.40. Tervezzünk ún. mintadiszkriminátort egybemenetű (x), egykimenetű (z) Moore-modell szerint működő szinkron sorrendi hálózatként, amelynek előírt működése az alábbi!

A z kimenet akkor és csak akkor váljon 1 értékűvé, ha a bemeneti jelsorozatban négy, egymást közvetlenül követő 1 érték fordul elő! Az azonosítandó minta tehát $x=1, 1, 1, 1$. Minden más esetben a z kimenet értéke legyen 0!

F.2.41. Tervezzünk ún. impulzushelyzet-modulációval ábrázolt értékek közötti különbségképzést végrehajtó, Moore-modell szerint működő, hárombemenetű (x_1, x_2, x_3); háromkimenetű (z_1, z_2, z_3) szinkron sorrendi hálózatot, amelynek előírt működése az F.40. ábra alapján az alábbi!

— Az x_3 jel 1 értéke — ha fellép — négy órajel-periódus időtartamára áll fenn.
— Az x_1 és x_2 jelek 1 értékei $x_3=1$ idején egyetlen órajel-periódus idejéig állnak fenn.
— Az x_1 és x_2 bemeneti jelek 1 értékei az impulzushelyzet-moduláció elvén



F.40. ábra. Az F.2.41. feladatban előírt különbségképző működését szemléltető idődiagram

rendre aszerint jelölik ki a 0, 1, 2, 3 számok valamelyikét, hogy $x_3=1$ tartama alatti óraimpulzusok közül melyeket fedik át. Az F.40. ábra idődiagramja az 1 és 2 számok kijelölését szemlélteti.

A hálózat feladata a kijelölt számok különbségének kijelzése a kimeneti kombinációval. A kivonás során az x_1-gyel ábrázolt szám legyen a kisebbítendő!

A kimeneti kombináció értelmezése legyen az alábbi!

— A z_2 és z_3 kimenetek értéke kétbites bináris számként ábrázolja a különbség értékét. Az z_1 kimenet értéke akkor és csak akkor legyen 1, ha a különbség negatív előjelű!

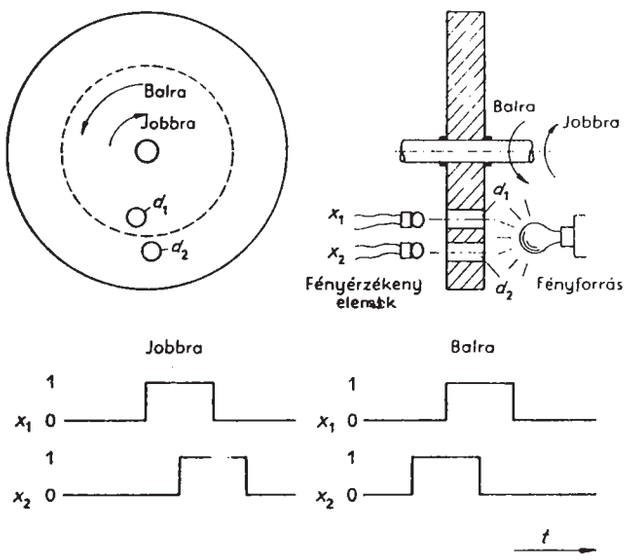
— Az $x_3=1$ fennállása idején a kimeneti kombináció ne változzon! A fentiekben előírt működésnek megfelelő kimeneti kombináció $x_3=0$ megjelenésekor azonnal lépjen fel!

F.2.42. Tervezzünk forgásirány-érzékelő aszinkron sorrendi hálózatot, amelynek előírt működése az F.41. ábra alapján az alábbi!

— A forgó tárcsán egymáshoz képest kissé eltolt helyzetben két furat (d_1, d_2) helyezkedik el. A tárcsa előtt, ill. mögött a furatoknak megfelelő helyzetben tételezzünk fel álló fényforrást, illetve fényérzékeny elemeket.

— A tárcsa forgásirányától függően a furatok más-más időbeli sorrendben engednek fényt a fényérzékeny elemekre, amelyek által előállított időben eltolt helyzetű impulzusok (x_1, x_2) sorrendje a forgásirányra jellemző.

A tervezendő aszinkron sorrendi hálózatnak legyen két bemenete (x_1, x_2) és két kimenete ($z_1 z_2$). A z_1 kimeneten akkor és csak akkor jelenjen meg 1 érték, ha a tárcsa jobbra forog. A z_2 kimenet hasonló módon azt jelezze, ha a tárcsa balra forog.



F.41. ábra. Az F.2.42. feladatban előírt forgásirány-érzékelő működésének szemléltetése

F.2.43. Építsük fel az F.2.42. feladatban előírt működésű aszinkron sorrendi hálózatot az egyszerű élvezérelt D flip-flopnak aszinkron hálózatként történő felhasználásával!

F.2.44. Tervezzünk kétbemenetű (x_1, x_2), egykimenetű (z) Moore-modell szerint működő aszinkron sorrendi hálózatot, amelynek előírt működése az alábbi!

— A z kimenet értéke akkor és csak akkor változzon 1-re, ha az $x_1x_2=10, 11, 01$ bemeneti kombinációsorozat jut a hálózatra.

— A z kimenet értéke akkor, és csak akkor változzon 0-ra, ha az $x_1x_2=10, 00$ bemeneti kombinációsorozat jut a hálózatra.

F.2.45. Tervezzünk az F.2.44. feladatban előírt működés alapján szinkron sorrendi hálózatot!

F.2.46. Végezzük el az F.42. ábrán szereplő aszinkron állapottábla összevonását, és válasszunk állapotkódokat! Tartalmaz-e a hálózat lényeges hazárdot?

F.2.47. Tervezzünk ún. soros komparátort kétbemenetű (x_1, x_2), egykimenetű (z) szinkron sorrendi hálózatként, amelynek előírt működése az alábbi!

— A két bemenetre érkező jelek az órajellel ütemezve egy-egy bináris szám összetartozó bitjeiként értelmezendők.

— A z kimenet értéke akkor és csak akkor legyen 1, ha a bemenetre érkező bináris számok öt egymást követő helyértéken azonosak.

F.2.48. Tervezzünk ún. soros műveleti egységet egybemenetű (x), egykimenetű (z) szinkron sorrendi hálózatként az alábbi célfeladatra!

— Jelölje N_1 az egymás után fellépő $x=1$ értékek számát, N_0 pedig az egymás utáni $x=0$ értékekét.

— A z kimenet értéke akkor és csak akkor legyen 1, ha $N_1-N_0=7-a\cdot 8$, ahol a tetszőleges egész számot jelöl.

— A $z=1$ érték csak egyetlen órajel-periódus idejére álljon fenn.

Vizsgáljuk meg, hogy létezik-e a hálózatnak soros vagy párhuzamos dekompozíciója?

F.2.49. Egyszerűsítsük az F.43. ábrán szereplő szinkron állapottáblát! Válasszunk állapotkódokat, és építsük fel a hálózatot J—K flip-flopok alkalmazásával!

y \ x_1x_2	00	01	11	10
a	(a) 0	b 0	–	f 0
b	a 0	(b) 0	c –	–
c	–	d 0	(c) 1	h 1
d	a 0	(d) 0	e 0	–
e	–	d 0	(e) 0	f 0
f	a 0	–	g 0	(f) 0
g	–	b 0	(g) 0	f 0
h	a 0	–	c –	(h) 1

F.42. ábra. Állapottábla az F.2.46. feladathoz

$y \backslash x_1x_2$	00	01	11	10
A	E 1	C 0	B 1	E 1
B	C 0	F 1	E 1	B 0
C	B 1	A 0	D 1	F 1
D	G 0	F 1	E 1	B 0
E	C 0	F 1	D 1	E 0
F	C 1	F 1	D 0	H 0
G	D 1	A 0	B 1	F 1
H	B 1	C 0	E 1	F 1

F.43. ábra. Állapottábla az F.2.49. feladathoz

$y \backslash x_1x_2$	00	01	11	10
A	C 1	D 1	D 1	–
B	D 0	D 1	C 0	–
C	D 0	B 0	A 1	–
D	C 1	A 0	B 0	–

F.44. ábra. Állapottábla az F.2.50. feladathoz

F.2.50. Alakítsuk át az F.44. ábrán szereplő állapottáblával adott, Mealy-modell szerint működő szinkron sorrendi hálózatot Moore-modellé!

F.2.51. Vizsgáljuk meg, hogy az alábbi logikai függvényekkel D—G flip-flopokból felépített aszinkron sorrendi hálózat

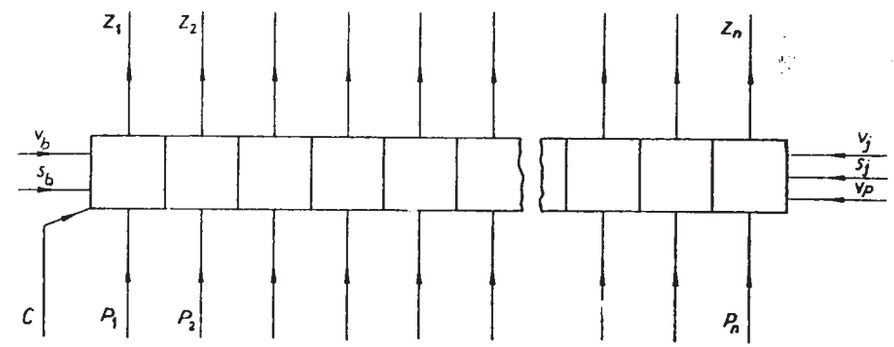
— egyszerűsíthető-e?
— tartalmaz-e kritikus versenyhelyzetet és lényeges hazárdot?
— milyen szinkron flip-flopot valósít meg, ha az x_2 bemenetet tekintjük órajelbemenetnek?

$$D_1 = x_2\bar{y}_1, \qquad G_1 = \bar{y}_1 y_2 + x_2 y_2,$$

$$D_2 = x_1\bar{x}_2, \qquad G_2 = 1.$$

$$z = y_1.$$

F.2.52. A logikai rendszerekben *regiszereknek* nevezik azokat a speciális sorrendi hálózatokat, amelyek a bináris információ átmeneti tárolására szolgálnak. Lényegében a memóriához hasonló a funkciójuk, de rendszerint gyorsabb működésűek, ugyanakkor tárolási kapacitásuk általában kisebb. Gyakran előnyös, ha a regiszterek az információ tárolásán kívül még arra is képesek, hogy azt helyértékenként jobbra vagy balra tudják léptetni egy léptetőjel hatására. Az ilyen regisztereket *léptető-(shift) regiszereknek* nevezik. Az F.45. ábrán egy léptetőregiszter vázlata látható. Az egyes bemeneti és kimeneti jeleknek az alábbi értelmezést adhatjuk:



F.45. ábra. A léptetőregiszter vázlata az F.2.52. feladathoz

$z_1 \dots z_n$ A regiszter elemeinek kimenetei, amelyek a tárolt bináris információ egy-egy bitjét jelentik.

$p_1 \dots p_n$ A regiszter párhuzamos beírómenetei. Az egyes vezetékekre jutó jelek a regiszterbe beírandó bináris információ bitjei.

C Léptetőjel, általában impulzusokat kell erre a vezetékre juttatni, amelyek mindegyikének hatására a regiszterben tárolt bináris információ egy helyértékkel eltolódik jobbra, ill. balra:

$$z_{i,t_1} = z_{i-1,t_2}, \qquad \text{ill.} \qquad z_{i,t_1} = z_{i+1,t_2}$$

ahol t_1 a léptetőjel előtti időpontbeli z értékre utal, t_2 a léptetőjel utáni időpontbeli z értékre utal, i a helyérték indexe.

s_b Soros beíróbemenet. Minden egyes léptetőjel (C) hatására a z_1 kimenet értéke egyenlő lesz az s_b vezetéken levő jel mindenkori értékével. Így tehát, ha a léptetőimpulzusokkal szinkronban egy bináris információ egyes bitjeit juttatjuk az s_b-vezetékre, akkor soros beírást végezhetünk a regiszterbe balról.

s_j Soros beíróbemenet. Hatása s_b-vel azonos jobbról történő soros beírás esetén.

v_j, v_b, v_p Vezérlőjelek, amelyek a regiszter üzemmódját állítják be. Egyidejűleg csak az egyik vezérlőbemenetre juthat 1 érték.

$v_j = 1$ esetén jobbra léptetés,

$v_b = 1$ esetén balra léptetés,

$v_p = 1$ esetén a $p_1 \dots p_n$ bemenetekről párhuzamos beírás történik.

A vezérlőjelek kombinációira természetesen másfajta értelmezést is adhattunk volna, de ez a regiszter tulajdonságait lényegében nem változtatná meg. Megjegyezzük, hogy vannak olyan, építőelemként kapható léptetőregiszterek is, amelyek a párhuzamos beíráshoz a $v_p = 1$ érték mellett óraimpul-

zust is igényelnek a C bemeneten vagy külön órajel-bemeneten. Az ilyen megoldásoknak a hazárdjelenségek elkerülése szempontjából számos előnyük van.

A regiszter eddig megismert működési feltételeit megvalósító sorrendi hálózat megtervezése céljából induljunk ki egy olyan regiszterből, amely a felsorolt üzemmódok közül csak a jobbra léptetésre és a tárolásra képes. Könnyen belátható, hogy ebben az esetben a regiszter elemi sorrendi hálózatokra bontható, amelyek mindegyike a bináris információ egyetlen bitjének tárolására képes. Ezek az elemi sorrendi hálózatok két bemenettel és egy kimenettel képzelhetők el. Az x_1 bemenet jelentse a balról megelőző bitnek megfelelő sorrendi hálózat kimenetét, x_2 a léptetőjelet, z pedig az illető bitnek megfelelő regiszter kimenetet. Ezek után megfogalmazhatjuk az elemi sorrendi hálózat működési feltételét:

— z kimenet értéke legyen egyenlő x_1-nek azzal az értékével, amely x_2 hatásának pillanatában fennállt, és ezután mindaddig maradjon változatlan, amíg új x_2 nem hat a bemeneten. Ha ennek a működési feltételnek megfelelő sorrendi hálózatot megtervezzük, akkor az F.46. ábra alapján építhetjük fel a jobbra léptetésre és tárolásra alkalmas regisztert. Mivel ebben az esetben a regiszter csak egyetlen üzemmódban működhet, üzemmódvezérlő jelre nincs szükség. Az elemi sorrendi hálózatra megfogalmazott működési előírásból látszik, hogy a feladat egyetlen D flip-flop-pal megoldható, $D=x_1$, $C=x_2$, $z=y$ vezérléssel.
Felléphetnek-e hazárdjelenségek az így felépített léptetőregiszterben, ha egyszerű élvezérelt vagy master—slave működésű flip-flopokat alkalmaztunk?

F.2.53. Tervezzük meg az F.2.52. feladatban vázolt összes tulajdonsággal rendelkező léptetőregiszter egy elemét reteszelt működésű (data-lock-out) flip-flop alkalmazásával! Felléphetnek-e hazárdjelenségek az így felépített léptetőregiszterben?

F.2.54. Az ún. *számlálók* olyan speciális sorrendi hálózatok, amelyek kimeneteiken azt jelzik, hogy egy adott bemenetükön hányszor változott a jel értéke. Ha ennek a bemenetnek a jelét impulzussorozatnak tekintjük, akkor a mindenkori kimeneti kombináció azt jelzi, hogy a vizsgálat időpontjáig hány impulzus érkezett a bemenetre. Előnyös, ha a számlálót könnyen tudjuk alaphelyzetbe hozni (reset), vagyis a kimeneti kombinációt olyan értékre állítani, amely azt jelzi, hogy egyetlen impulzus sem érkezett a bemenetre. Sok esetben előfordul, hogy előre beállított kimeneti kombinációval megadott számtól kell az impulzusokat tovább számlálni. Ilyenkor tetszőleges értékre be kell tudnunk állítani a kimeneti kombinációt (preset). Az is követelmény lehet a számlálóval szemben, hogy egy előre beállítható kimeneti kombinációnak megfelelő impulzusszámtól kezdve a továbbiakban beérkező impulzusokat visszaszámlálja, vagyis a kimeneti kombinációt minden egyes beérkező impulzus úgy módosítsa, mintha mindig eggyel kevesebb impulzus érkezett volna a bemenetre a mindenkori kimeneti kombinációhoz képest (hátraszámlálás).

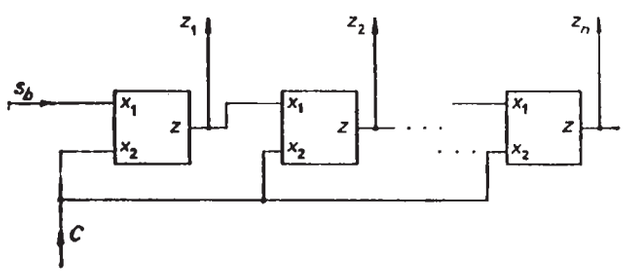
Az F.47. ábrán a számláló vázlata látható. Az egyes bemeneti és kimeneti jeleknek az alábbi értelmezést adhatjuk:

$z_1 \dots z_n$ Az egyes kimeneti jelek. Értékkombinációjuk alapján állapítható meg a bemenetre érkezett impulzusok száma;

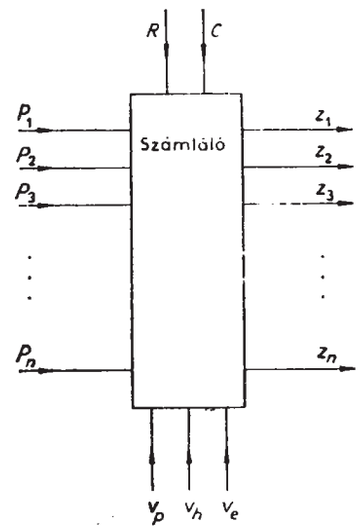
$p_1 \dots p_n$ Preset bemenetek, amelyekre juttatott jelekkel a kimeneti kombinációt adott értékre lehet beállítani;

C A számlálandó jelváltozások bemenete;

R Reset bemenet, amelyre juttatott 1 alaphelyzetbe állítja a kimeneti kombinációt.



F.46. ábra. Jobbra léptetésre és tárolásra alkalmas regiszter vázlata az F.2.52. feladathoz



F.47. ábra. A számláló vázlata az F.2.54. feladathoz

v_e, v_h, v_p Vezérlőjelek, amelyek segítségével a számláló üzemmódját lehet beállítani. Egyidejűleg csak az egyik vezérlőbemenetre juthat 1 érték

$v_e=1$ esetén előre számlálás,

$v_h=1$ esetén hátra számlálás,

$v_p=1$ esetén a $p_1 \dots p_n$ bemenetekről beírás (preset) történik.

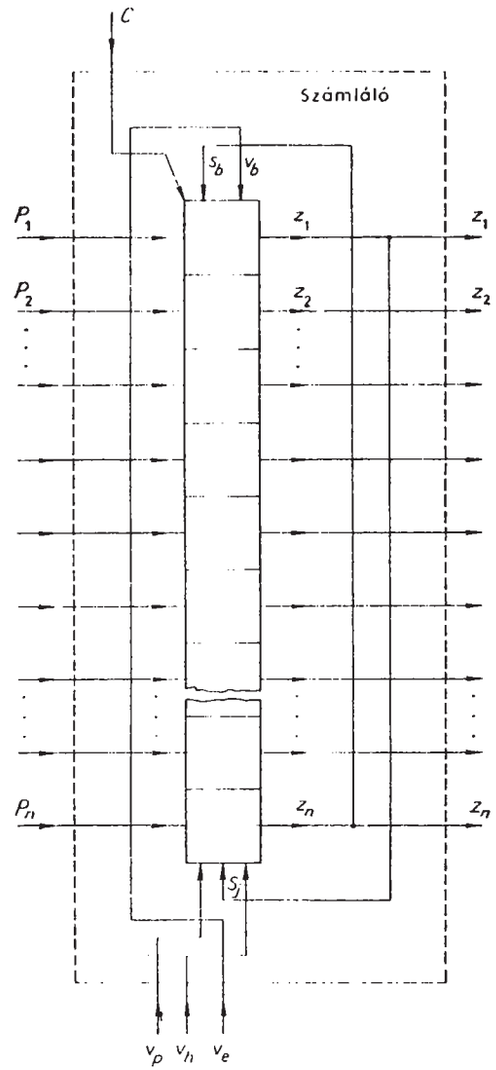
A vezérlőjelek kombinációira természetesen másfajta értelmezést is adhattunk volna, de ez a számláló tulajdonságait lényegében nem változtatná meg. Vannak olyan számlálót tartalmazó építőelemek is, amelyek a párhuzamos beíráshoz a $v_p=1$ érték mellett óraimpulzust is igényelnek. Ezáltal számos alkalmazásban egyszerűbben biztosítható a hazárdmentes működés.

Regiszter felhasználásával könnyen valósíthatunk meg számlálót. A számlálandó impulzusokat a léptetőjel (C) bemenetre vezetjük, és a regiszter utolsó kimenetét a soros bemenettel (s_b, ill. s_j) összekötjük.

Az F.48. ábrán a szaggatott vonallal körülhatárolt részben a regiszter ismert jelölései, azonkívül pedig a számláló bemeneti és kimeneti jelölései szerepelnek. A számlálást párhuzamos beírásnak kell megelőznie: előre számlálás esetén ($v_e=1$) $p_1=1$ és $p_2 \dots p_n=0$, hátra számlálás esetén ($v_h=1$) $p_1 \dots \dots p_{n-1}=0$ és $p_n=1$ információval. Ez tulajdonképpen a számláló alaphelyzetbe állítását is jelenti, ezért nem szerepel az ábrán külön R vezeték. Minden egyes léptetőjel hatására a beírt 1 érték tovább kerül egy kimenettel. Így a beérkezett impulzusok száma mindig megállapítható annak alapján, hogy melyik z kimenet értéke 1. Előre számlálás esetén az $n-1$-edik számlálandó jel hatására z_n kimenet értéke lesz 1. Az n-edik impulzus hatására a számláló a z_n-s_b összekötés miatt alaphelyzetbe ($z_1=1$) kerül, és a számlálás elölről kezdődik a további számlálandó jelek hatására. Az ilyen fajta működést *modulo* jellegű számlálásnak nevezik. Jelen esetben a számlálás modulo-n jellegű. Hátraszámlálás esetén ugyanez érvényes a z_1-s_j összekötés miatt.

A regiszter felhasználásával kialakított számláló tehát úgy működik, hogy a párhuzamos beírás során az egyetlen regiszterelembe beírt 1 érték a számlálandó jelek hatására kering a „visszacsatolt" regiszterben. Ezért az így felépített számlálókat *gyűrűs számlálóknak* nevezik.

A gyűrűs számláló esetén annyi kimenetre van szükségünk, ahány beérkezett számlálandó jelet kell jelezni (n-ből 1 kód). Ha a számláló kimenetén valamilyen más bináris kódban ábrázolva keletkezik a beérkezett impulzusok száma, akkor általában kevesebb kimenet is elegendő. Ha pl. normál bináris kódban értelmezzük a számláló kimeneti kombinációját, akkor a számláló működését az F.49. ábrán adott táblázattal írhatjuk le. Az ilyen számlálókat *bináris számlálóknak* nevezzük. A táblázat alapján megállapítható, hogy z_2-től z_n-ig az egyes kimenetek értéke csak akkor változik, ha a megelőző kimenet értéke 1-ről 0-ra változik. Tehát ezek a kimenetek egymás után kap-

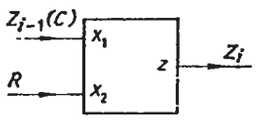


F.48. ábra. Gyűrűs számláló vázlata az F.2.54. feladathoz

csolt azonos elemi sorrendi hálózatokkal megvalósíthatók. Ha z_1 kimenetet is ezzel az elemi sorrendi hálózattal valósítjuk meg, akkor a számláló mindig akkor fog eggyel tovább számlálni, ha a bemeneten egy impulzus eltűnt (1—0 átmenet). Alaphelyzetbe állítás (reset) esetén az összes elemi sorrendi hálózatnak 0 értékű kimenetet kell adnia. Egy elemi sorrendi hálózatnak (F.50. ábra) tehát az alábbi működési feltételeket kell kielégítenie:

A számlálandó impulzusok sorszáma	Z_1	Z_2	Z_3	Z_4		Z_{n+1}	Z_n
0	0	0	0	0		0	0
1	1	0	0	0		0	0
2	0	1	0	0		0	0
3	1	1	0	0		0	0
4	0	0	1	0		0	0
5	1	0	1	0		0	0
6	0	1	1	0		0	0
7	1	1	1	0		0	0
8	0	0	0	1		0	0
9	1	0	0	1		0	0
10	0	1	0	1		0	0
11	1	1	0	1		0	0
12	0	0	1	1		0	0

F.49. ábra. A bináris számlálás szemléltetése az F.2.54. feladathoz



F.50. ábra. A bináris számláló egy elemének értelmezése az F.2.54. feladathoz

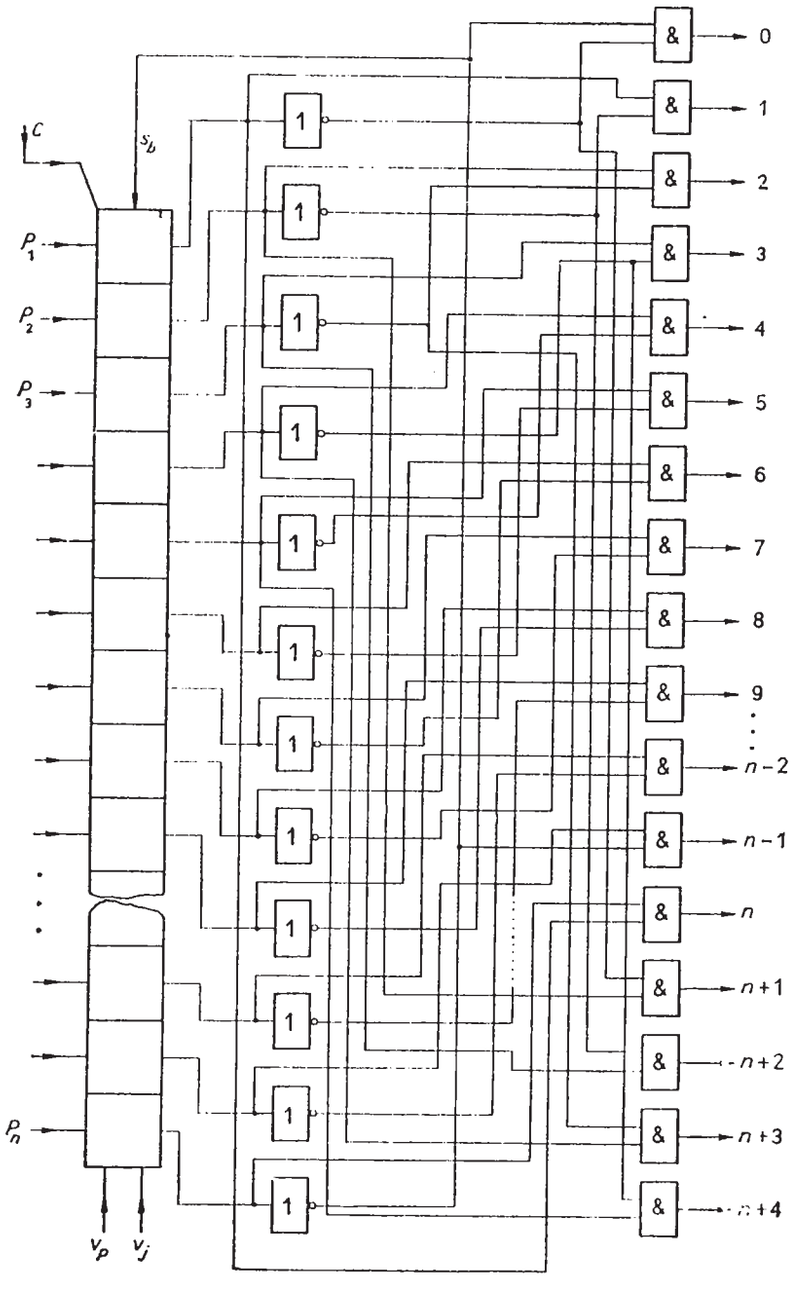
— z kimenet értéke változzon az ellentettjére, ha x_1 bemenet értéke 1-ről 0-ra változik.

— z kimenet értéke x_1 értékétől és változásaitól függetlenül legyen 0, ha x_2 bemenetre 1 jut.

— Minden más esetben z értéke maradjon változatlan. Könnyen belátható, hogy ilyen működési feltételekkel a bináris számláló elemi sorrendi hálózata egyetlen master—slave működésű T flip-floppal felépíthető a $T=1$, $C=x_1$, $R'=\bar{x}_2$, $z=y$ vezérléssel.

A regiszterhez hasonlóan ezúttal is viszonylag egyszerűen megvalósíthatók a számláló további üzemmódjai a T flip-floppal felépített elemi sorrendi hálózat kiegészítésével. Felléphetnek-e hazárdjelenségek az így felépített bináris számlálóban, ha egyszerű élvezérelt működésű flip-flopokat alkalmazunk?

F.2.55. Tervezzük meg az F.2.54. feladatban vázolt összes tulajdonsággal rendelkező bináris számláló egy elemét master—slave működésű flip-flop alkalmazásával! Felléphetnek-e hazárdjelenségek az így felépített számlálóban?



F.51. ábra. Johnson-számláló vázlata az F.2.57. feladathoz

F.2.56. Ha a bináris számláló kimeneteire megfelelő átkódolóhálózatot kapcsolunk, akkor annak kimenetén n-ből 1 kódban is rendelkezésünkre állhat a mindenkori beérkezett számlálandó impulzusok száma, akárcsak gyűrűs számláló esetén. Hasonlítsuk össze a két megoldást a kimeneteken lejátszódó hazárdjelenségek szempontjából!

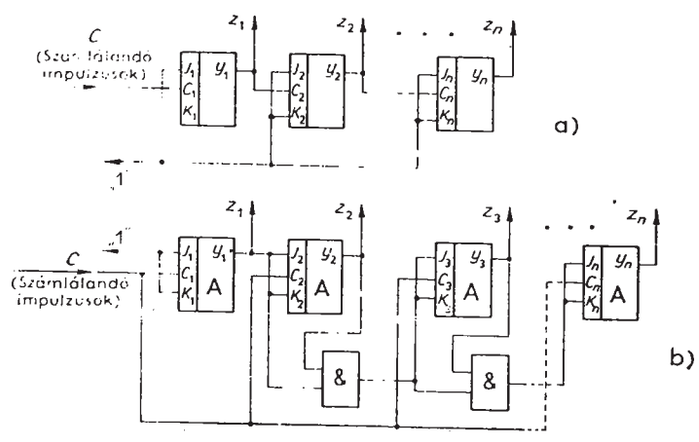
F.2.57. Az F.51. ábrán regiszterből kialakított ún. Johnson-számláló vázlata látható. Alaphelyzetben az összes léptetőregiszter-elem kimenete 0 értékű. Adjuk meg a számláló jellegű működés magyarázatát! Hasonlítsuk össze a megoldást a kimeneti hazárdjelenségek és a gazdaságosság szempontjából a bináris és a gyűrűs számlálóval!

F.2.58. Hasonlítsuk össze az F.52. ábrán vázolt kétfajta bináris számlálót a működési sebesség, a kimeneteken lejátszódó hazárdjelenségek és a gazdaságosság szempontjából!

F.2.59. Milyen hazárdjelenségeket okoz az F.52*a)* és *b)* ábrán vázolt számlálók működésében, ha egyszerű élvezérelt működésű flip-flopokat alkalmazunk? Master—slave működésű flip-flopokkal kiküszöbölhetjük-e mindezeket a hazárdjelenségeket?

F.2.60. Tervezzünk az F.52*a)* vagy *b)* ábra szerinti felépítésben olyan számlálót, amelynek modulo jellege adott értékre beállítható! Az alkalmazott flip-flopok Preset (*ST*) és Clear (*R'*) bemeneteit is felhasználhatjuk!

F.2.61. Tervezzünk az F.52*b)* ábra szerinti felépítésben olyan számlálót, amely hátraszámlálásra is képes!



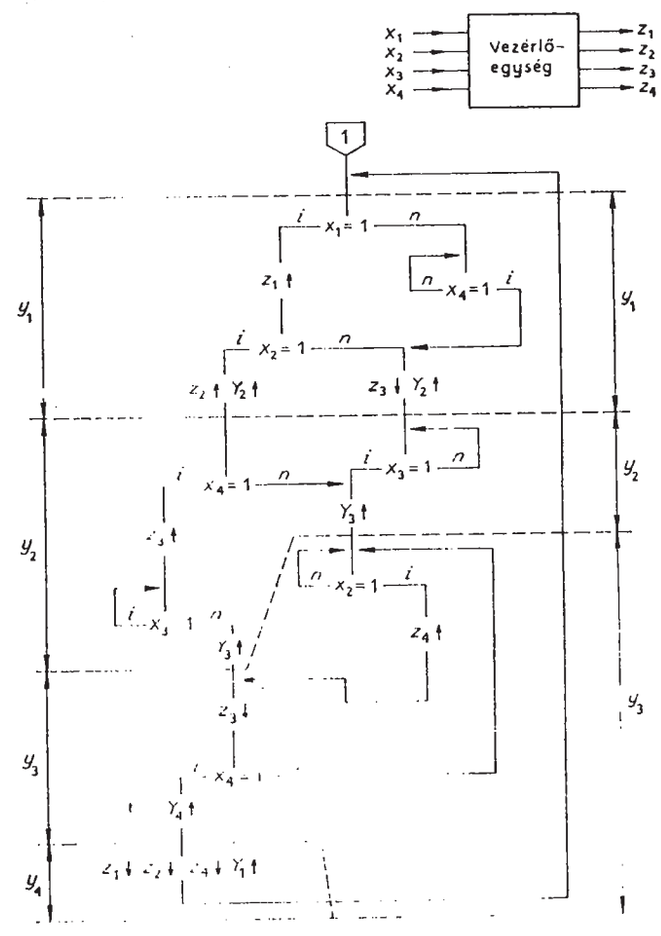
F.52. ábra. Számlálómegoldások az F.2.58. feladathoz

F.3. Vezérlőegységek (4. fejezet)

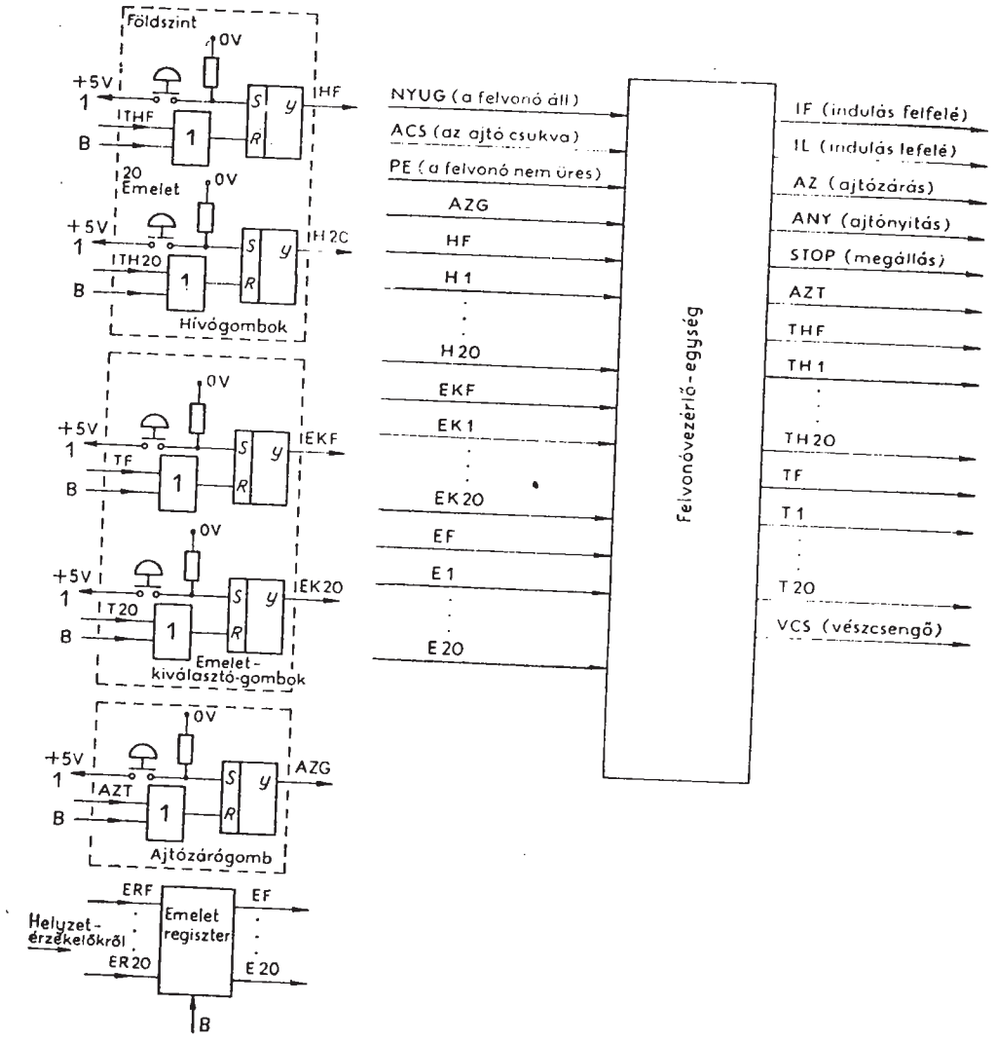
F.3.1. Írjuk fel az F.53. ábrán szereplő folyamatábrával megadott működésű vezérlőegység szinkron fázisregiszteres megvalósításának logikai függvényeit! Lehet-e csökkenteni a fázisok számát az ábrán bejelölt megoldáshoz képest a specifikációs gráf ismerete nélkül?

F.3.2. Alakítsuk át az F.53. ábrán szereplő folyamatábrát a kimeneti jeleket beállító műveletek aszinkron értelmezése alapján! Írjuk fel a vezérlőegység aszinkron fázisregiszteres megvalósításának logikai függvényeit!

F.3.3. Az F.2.36. feladatban leírt impulzushelyzet-azonosító hálózatot tekintsük



F.53. ábra. Folyamatábra az F.3.1. feladathoz



F.54. ábra. Rendszertechnikai vázlat az F.3.6. feladatban előírt mikroprogramozott felvonóvezérlő-egység tervezéséhez

vezérlőegységnek és az előírt működés alapján szerkesszük meg a specifikációs gráfot, az órajelet is bemeneti jelnek tekintve!

F.3.4. Az F.3.3. feladatban meghatározott specifikációs gráf alapján szerkesszük meg a folyamatábrát és írjuk fel a szinkron fázisregiszteres megvalósítás logikai függvényeit!

F.3.5. Az F.3.4. feladatban megszerkesztett folyamatábrát alakítsuk át a kimeneti

jeleket beállító műveletek aszinkron értelmezése alapján, és írjuk fel az aszinkron fázisregiszteres megvalósítás logikai függvényeit!

F.3.6. Tervezzünk mikroprogramozott felvonóvezérlő-egységet, amelynek előírt működését az F.54. ábra alapján a következőképpen fogalmazhatjuk meg!

— A felvonó induljon el, ha a felvonóban levő emeletkiválasztó gonbok, vagy az egyes szinteken levő hívógombok flip-flopjai közül legalább egy (EKF, ..., EK20, HF, ..., H20) $y=1$ állapotban van és ez nem arra a szintre vonatkozik, amelyen a felvonó éppen tartózkodik (EF...E20).

— Az indulás iránya attól függ (IF=1: indulás felfelé, IL=1: indulás lefelé), hogy az emeletkiválasztó-gombok vagy hívógombok $y=1$ állapotban levő flip-flopjai közül melyik vonatkozik a felvonó pillanatnyi helyzetéhez (EF...E20) képest a legközelebbi szintre.

— Az elindulás feltétele az ajtók zárása (AZ=1), amelynek megtörténtét az ACS=1 jelérték nyugtázza.

— Ha a hívás vagy az emeletkiválasztás arra a szintre vonatkozik, amelyen a felvonó pillanatnyilag tartózkodik, akkor elindulás helyett ajtónyitás (ANY=1) történjen. A kinyitott ajtók 5 s-ig maradjanak nyitva, ha a felvonóban nem tartózkodik senki, amit a padlóérzékelő által szolgáltatott jelértékből (PE) állapíthat meg a felvonó vezérlőegysége. Ha a felvonóban már tartózkodnak utasok (PE=1), akkor az ajtózáró gomb megnyomása zárja az ajtókat (AZG=1). Az ajtók záródása után (ACS==1) a vezérlőegységnek törölnie kell az ajtózáró gombhoz tartozó flip-flopot (AZT=1). Ezután az elindulás feltételét és irányát a korábbiakban már megfogalmazott feltételek írják elő.

— Ha a felvonó olyan szinthez közeledik, amelyen a hívás vagy az emeletkiválasztás miatt meg kell állnia, akkor a vezérlőegységnek már a haladási irányban megelőző szinten ki kell adnia a STOP=1 jelértéket, amelynek hatására a felvonó éppen a kívánt szinten kerül nyugalomba (NYUG=1). Az ajtók nyitása csak ezután következhet. A NYUG=0 jelérték az ajtók nyitását tiltja.

— Ha egy adott szinten az emeletkiválasztás vagy a hívás hatására a felvonó megállt és megtörtént az ajtók nyitása, zására, akkor a továbbindulást megelőzően a vezérlőegységnek törölnie kell az adott szintnek megfelelő, emeletkiválasztó ill. hívó gombok flip-flopjait (TF...T20, THF...TH20).

— A felvonóban elhelyezett veszélyhelyzet nyomógomb megnyomásának hatására a mindenkori haladási irányba eső legközelebbi szinten kezdődjön el a megállási folyamat, történjen meg az ajtónyitás és a vészcsengő működtetése. A STOP és NYUG jelek kapcsolata ilyenkor is a már leírt módon jelezze a nyugalmi helyzetet, valamint az ajtónyitás lehetőségét. A vészcsengő mindaddig szóljon (VCS=1), amíg az ajtónyitás meg nem történik.

— A felvonó fülkéjében és az egyes szinteken a haladási irány, valamint a mindenkori tartózkodási szint kijelzésére az IF, IL, valamint az EF... ...EF20 jelek használhatók. Ehhez külön vezérlési funkció nem tartozik. Hasonló módon az egyes nyomógombok megnyomásának tényét az illető nyomógombhoz tartozó flip-flop állapota jelezheti (pl. izzólámpás nyomógomb). Így amíg a kért funkciót végre nem hajtja a vezérlőegység, vagyis amíg nem ad törlőjelet a megfelelő flip-flopra, addig pótlólagos vezérlési funkció nélkül is láthatóvá válik, hogy mely nyomógombokon jelzett kérések nincsenek még kiszolgálva. Az EMELET REGISZTERre azért van szükség, mert az egyes szinteken elhelyezett érzékelők jele (ERF...ER20) a felvonó mozgásából következően impulzus jellegű is lehet.

— A fenti működési feltételekből és a rendszertechnikai vázlatból következően a THF...TH20, TF...T20, AZT jelek impulzus jellegűek. PE=0 esetén ANY is impulzus jellegű. Értelemszerűen: STOP (IF+IL)=0, vagyis a STOP=1 előállításakor az IF, ill. IL jelet 0 értékre kell beállítani. Jelenkénti párbeszéd jellegű kapcsolat áll fenn az alábbi jelpárok között:

STOP—NYUG

AZ—ACS

ANY—AZG

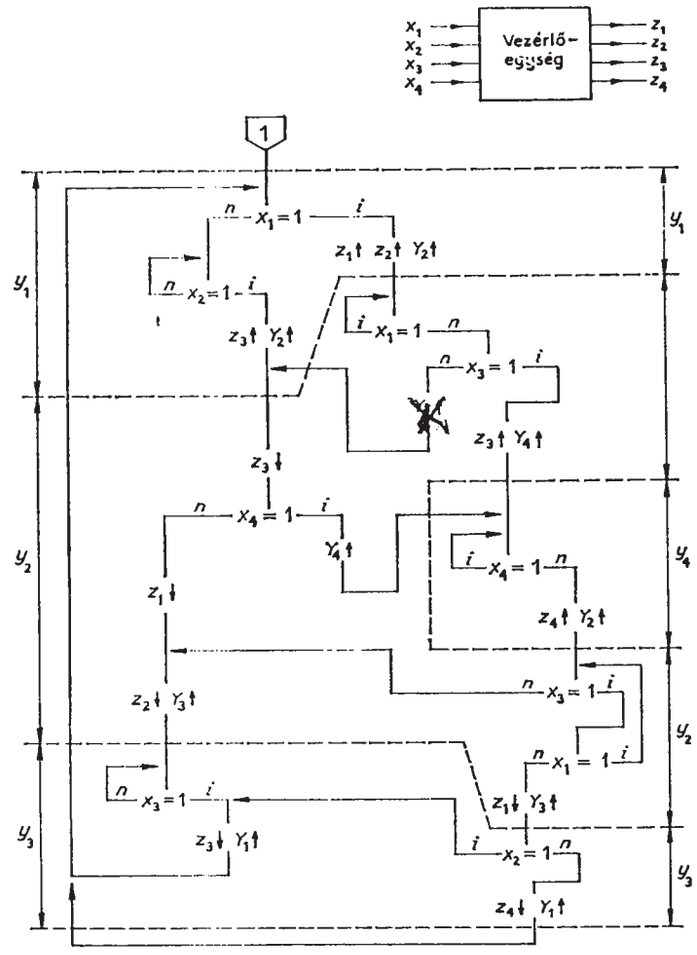
A rendszertechnikai vázlaton az alaphelyzetbe állító jel (*B*) hatása is követhető az egyes funkcionális egységekben.

Szerkesszük meg a fentiekben leírt működésű vezérlőegység folyamatábráját, és írjuk meg a mikroprogramot a 4.3. fejezetben szereplő mikroutasítás-szószervezést és processzort feltételezve! Az F.54. ábrán adott rendszertecnikai vázlatot kiegészíthetjük további funkcionális egységekkel (pl. komparátor), amelyek esetleg egyszerűsítik a mikroprogramot.

F.3.7. Írjuk fel az F.3.7. feladat megoldása során megszerkesztett folyamatábra alapján a felvonóvezérlő-egység szinkron fázisregiszteres megvalósításának logikai függvényeit!

F.3.8. Alakítsuk át az F.3.7. feladat megoldása során megszerkesztett folyamatábrát az aszinkron fázisregiszteres megvalósítás céljából! Határozzuk meg a felvonóvezérlő-egység aszinkron fázisregiszteres elvi logikai rajzát!

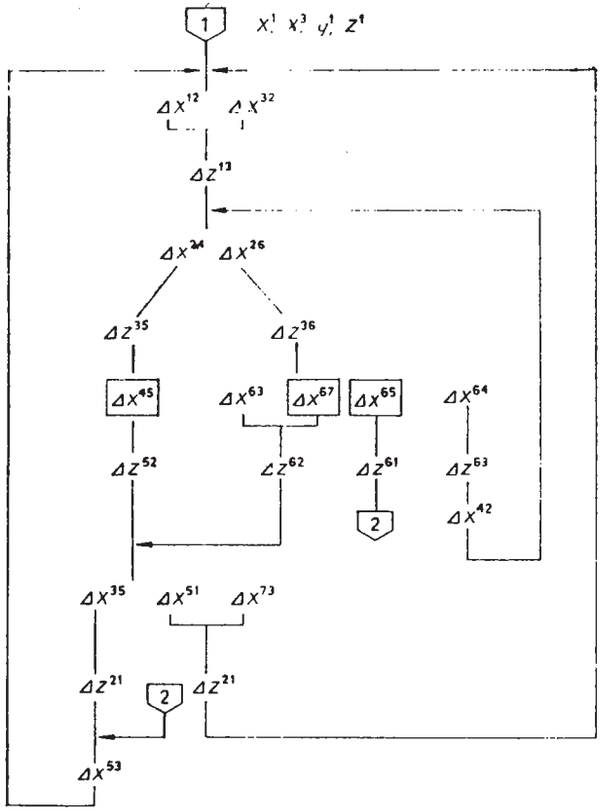
F.3.9. Az F.3.7. feladat megoldása során megszerkesztett folyamatábra alapján határozzuk meg egy olyan magasabb mikroprogramozási szint mikroutasítás-készletét és szószervezését, amely a 4.72. ábrán vázolt egymásba ágyazott



F.55. ábra. Folyamatábra az F.3.10. feladathoz

felépítés szerint kialakítva (P_1 processzor szintje) jelentősen egyszerűsíti a felvonóvezérlés mikroprogramját! Írjuk meg a magasabb szintű mikroutasításoknak megfelelő executív szintű programokat (M_0 tartalma), és tervezzük meg a 4.72. ábra szerinti FE_1 funkcionális egységeket! Írjuk meg a felvonóvezérlés mikroprogramját a 4.72. ábra szerinti P_1 processzor szintjén alkalmazható mikroutasítások felhasználásával (M_1 tartalma)!

A kapott megoldást hasonlítsuk össze az F.3.7. feladat megoldása során kapott executív szintű megoldással a működési sebesség, a szükséges memóriakapacitás és a gazdaságosság szempontjából!



F.56. ábra. Specifikációs gráf az F.3.13. feladathoz

F.3.10. Lehet-e csökkenteni a fázisok számát az F.55. ábrán megadott folyamatábrán a bejelölt megoldáshoz képest a specifikációs gráf ismerete nélkül? Ha igen, akkor szerkesszük át a folyamatábrát ennek megfelelően! Lehet-e további fázisösszevonásokat végezni, ha a specifikációs gráfot a lehető legkedvezőbbnek tételezzük fel ilyen szempontból? Ha igen, akkor szerkesszük át ennek megfelelően a folyamatábrát és rajzoljuk fel a feltételezett specifikációs gráfot! Adjuk meg a feltételezett specifikációs változásokat is!

F.3.11. Írjuk fel az F.3.10. feladat megoldása során kapott folyamatábra alapján a vezérlőegység szinkron fázisregiszteres megvalósításának logikai függvényeit!

F.3.12. Határozzuk meg az F.3.10. feladat megoldása során kapott folyamatábra szerint működő aszinkron fázisregiszteres vezérlőegység elvi logikai rajzát!

F.3.13. Egészítsük ki az F.56. ábrán szereplő specifikációs gráfot a szekunder változásokkal! Szerkesszük meg a folyamatábrát alábbi bemeneti és kimeneti kombinációk feltételezésével:

	x_1	x_2	x_3	x_4
X^1	0	0	0	0
X^2	1	0	0	0
X^3	1	0	0	1
X^4	1	1	0	1
X^5	0	1	0	1
X^6	1	1	1	1
X^7	0	0	1	1

	z_1	z_2	z_3	z_4
Z^1	1	0	0	0
Z^2	1	1	0	0
Z^3	1	1	1	0
Z^4	1	1	1	1
Z^5	1	0	1	0
Z^6	0	1	0	1

Végezzük el a lehetséges fázisösszevonásokat, és írjuk fel a szinkron fázisregiszteres megvalósítás logikai egyenleteit!

F.3.14. Szerkesszük meg a folyamatábrát az F.3.13. feladat megoldása során kiegészített specifikációs gráf alapján, ha a bemeneti kombinációk az alábbiak:

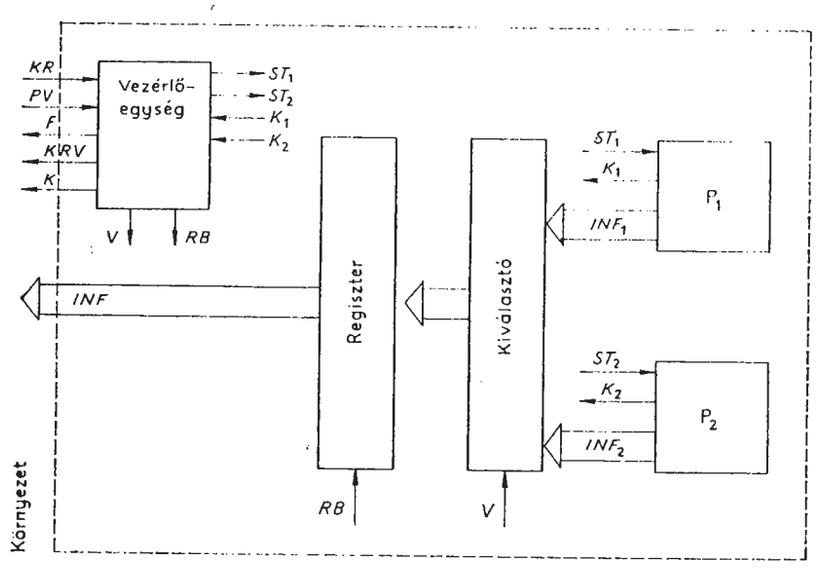
	x_1	x_2	x_3	x_4
X^1	0	0	0	0
X^2	0	1	0	0
X^3	0	1	0	1
X^4	1	1	0	0
X^5	0	0	0	1
X^6	1	1	0	1
X^7	0	1	1	1

A kimeneti kombinációkat tételezzük fel változatlanoknak az F.3.13. feladatban megadottakhoz képest. Határozzuk meg az így megszerkesztett folyamatábra szerint működő aszinkron fázisregiszteres vezérlőegység elvi logikai rajzát!

F.3.15. Tervezzünk mikroprogramozott vezérlőegységet az F.3.13. feladat megoldása során megszerkesztett folyamatábra alapján! Írjuk meg a mikroprogramot a 4.3 fejezetben szereplő mikroutasítás-szószervezést és processzort feltételezve! Határozzuk meg a vezérlőegység működési sebességét 5 MHz frekvenciájú órajel és 150 ns ciklusidejű mikroprogramtár esetén!

F.3.16. Tervezzük meg az F.57. ábrán látható rendszertechnikai vázlat vezérlőegységét, amelynek előírt működése a következőképpen fogalmazható meg:

— A P_1 és P_2 berendezésekről jövő információ (INF_1, ill. INF_2) választhatóan jut a rendszer INF kimeneteire. A kiválasztó hálózat $V=1$ esetén INF_1-et ($V=0$ esetén INF_2-t) juttatja a regiszterre, amelybe az $RB=1$ jelérték hatására történik beírás. A P_1 és P_2 berendezések megfelelő ST_i és K_i jelei jelenkénti párbeszéd jellegű párok. Az $ST_i=1$ érték hatására $K_i=1$ akkor jelenik meg, ha az INF_i vezetékekről már biztonságosan vehető az információ.



F.57. ábra. Rendszertechnikai vázlat az F.3.16. feladathoz

— A vezérlőegység a környezetből érkező igény ($KR=1$) hatására indítja P_1 vagy P_2 működését a PV jel értékétől függően. ($PV=1$ esetén P_1-et, $PV=0$ esetén P_2-t.) A KRV jel a KR jel párbeszéd jellegű nyugtázó párja, és 1 értékével csupán azt jelzi, hogy a vezérlőegység tudomásul vette a $KR=1$ által jelzett környezeti igényt. A környzet számára a $K=1$ érték megjelenése jelzi, hogy a PV jel értéke által kijelölt berendezésről érkezett információ már biztonságosan hozzáférhető az INF vezetékeken. A $K=1$ érték újabb $KR=1$ fellépéséig maradjon fenn. Tételezzük fel, hogy PV értéke is csak ekkor változik.

— Ha a $KR=1$ érték olyankor lép fel, egy adott berendezésre vonatkozóan, amikor vezérlőegység egy korábbi $KR=1$ hatására még a másik berendezéssel foglalkozik, akkor ez a tény a környezet számára úgy legyen észrevehető, hogy KRV helyett az F (foglaltság) jel nyugtázza KR-t. Ilyenkor a folyamatban levő jelfolyamat $K=1$ megjelenéséig zavartalanul játszódjon le, és ezután jelenjen meg a $KRV=1$ érték, amivel a vezérlőegység azt jelezze a környezet számára, hogy a foglaltság miatt korábban meghiúsult kiválasztás már lehetséges.

— Ha a $KR=1$ érték egyszer vagy többször olyankor lép fel egy adott berendezésre vonatkozóan, amikor a vezérlőegység egy korábbi $KR=1$ hatására éppen az adott berendezéssel foglalkozik, akkor ez a tény a környezet számára ne legyen észrevehető. Ilyenkor tehát KRV a szokásos módon nyugtázza KR-t, és a folyamatban levő jelfolyamat zavartalanul játszódjon le, de $K=1$ ne jelenjen meg. Ehelyett a vezérlőegység az adott berendezést még egyszer indítsa el, és csak ennek az újabb jelfolyamatnak a végén állítsa elő a $K=1$ értéket.

— Fentiek alapján az alábbi jelenkénti párbeszéd jellegű párok sorolhatók fel:

$$ST_1-K_1$$
$$ST_2-K_2$$
$$KR-KRV,\ F,\ (K)$$

Az RB és V jelek impulzus jellegűek. A PV és a K jelek a leírt módon adott feltételek teljesülésekor változnak. A K jel 1 értékének fellépését a fenti leírásból következően $KR=1$ megjelenése nyugtázza, de K-nak már nincs hatása KR további változásaira. Ezért nem áll fenn közöttük teljes párbeszéd jellegű kapcsolat.

Szerkesszük meg a leírt működés alapján a vezérlőegység specifikációs gráfját, majd ennek alapján a folyamatábráját! Kíséreljük meg a működési leírás alapján közvetlenül is létrehozni a folyamatábrát. Hasonlítsuk össze a kétféle módon kapott folyamatábrát, és végezzük el a lehetséges fázisösszevonásokat!

F.3.17. Az F.3.16. feladat megoldása során kapott folyamatábra alapján írjuk fel a vezérlőegység szinkron fázisregiszteres megvalósításának logikai függvényeit!

F.3.18. Alakítsuk át az F.3.16. feladat megoldása során kapott folyamatábrát Moore-modellé, és írjuk fel a vezérlőegység szinkron fázisregiszteres megvalósításának logikai függvényeit!

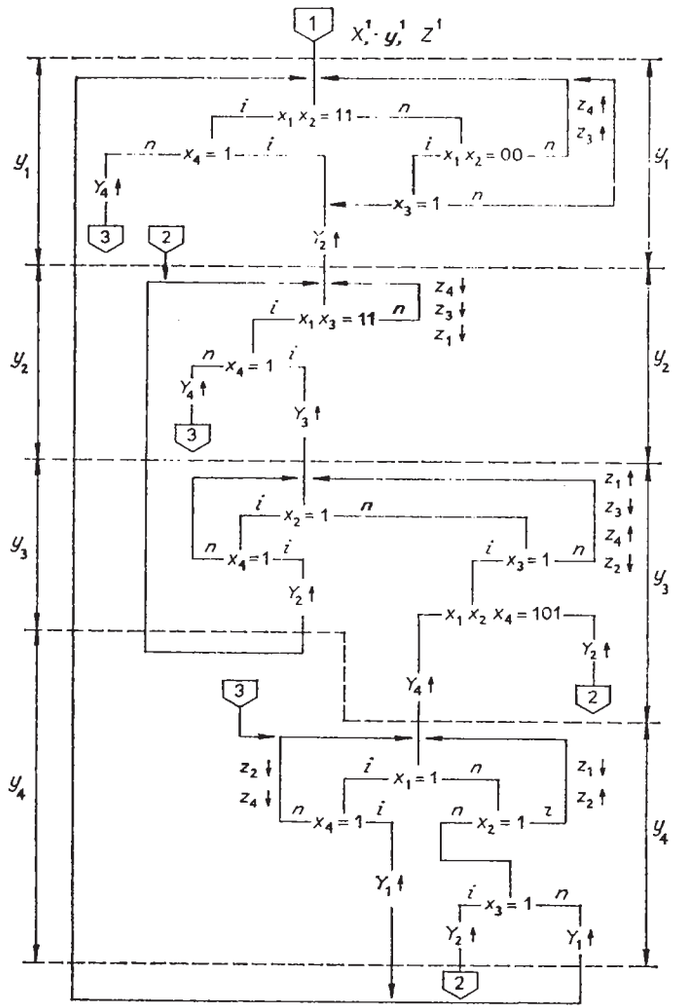
F.3.19. Alakítsuk át az F.3.16. feladat megoldása során kapott folyamatábrát az aszinkron fázisregiszteres megvalósításhoz, és rajzoljuk fel az így adódó vezérlőegység elvi logikai rajzát!

F.3.20. Az F.3.16. feladat megoldása során kapott folyamatábra alapján tervezzünk mikroprogramozott vezérlőegységet! Írjuk meg a mikroprogramot a 4.3. fejezetben szereplő mikroutasítás szószervezést és processzort feltételezve! Határozzuk meg, hogy milyen gyakori környezeti igényekre képes válaszolni a mikroprogramozott vezérlőegység 5 MHz frekvenciájú órajel és 150 ns ciklusidejű mikroprogramtár esetén, ha a P_1, ill. P_2 berendezést legfeljebb 10^5 információ/s sebességgel lehet működtetni.

F.3.21. Egészítsük ki az F.57. ábrán szereplő rendszertechnikai vázlatot és az F.3.16. feladatban előírt működést kétirányú információáramlás esetére! Szerkesszük meg a folyamatábrát, és ebből kiindulva is oldjuk meg az F.3.17., F.3.18., F.3.19., F.3.20. feladatokat!

F.3.22. Különböztessük meg a billentő- és az elágazóműveleteket az F.58. ábrán szereplő folyamatábrán. Írjuk fel a V^i_{js} és az E^i_{js} függvényeket!

F.3.23. Rajzoljuk fel az F.58. ábrán szereplő folyamatábra szerint működő aszinkron fázisregiszteres vezérlőegység elvi logikai rajzát!



F.58. ábra. Folyamatábra az F.3.22. és az F.3.23. feladathoz

Irodalomjegyzék

[1] *H. Peterson:* Introduction to Switching Theory and Logical Design (Wiley, 1968.)
[2] *D. Lewin:* Logical Design of Switching Circuits (Nelson, 1968.)
[3] *E. J. McCluskey:* Introduction to the Theory of Switching Circuits (McGraw Hill, 1965.)
[4] *S. H. Unger:* Asynchronous Sequential Circuits (Wiley, 1968.)
[5] *R. Miller:* Switching Theory I–II (Wiley, 1965.)
[6] *F. C. Hennie:* Finite-state Models for Logical Machines (Wiley, 1968.)
[7] *S. H. Unger:* State Assignment of Asynchronous Sequenital Circuits Not Containing Essential Hazard (IEEE Transactions on Computers 1968.)
[8] *S. Wendt:* Entwurf komplexer Schaltwerke (Springer, 1974.)
[9] *W. Grass:* Steuerwerke (Entwurf von Schaltwerken mit Festwertspeichern) (Springer, 1978.)
[10] *Janovics S. – Dr. Tóth M.:* A logikai tervezés módszerei (Műszaki Könyvkiadó, 1973.)
[11] *J. H. Tracey:* Internal state assignment for asynchronous sequential machines (IEEE Transactions on Computers EC–15, 1966.)
[12] *P. H. Starke:* Abstract Automata (North-Holland, 1972.)
[13] *Yahiko Kambayashi:* Logic Design of Programmable Logic Arrays (IEEE Transactions on Computers C–28, No. 9. 1979.)
[14] *J. D. Greenfield:* Practical Digital Design Using IC'S (Wiley, 1977.)
[15] *T. L. Booth:* Sequential Machines and Automata Theory (Wiley, 1967.)
[16] *Gécseg F. – Peák I.:* Algebraic Theory of Automata (Akadémiai Kiadó, 1972.)
[17] *S. S. Husson:* Microprogramming: Principles and Practices (Prentice-Hall, 1970.)
[18] *Fuchs L.:* Algebra (ELTE TTK jegyzet) (Tankönyvkiadó, J3–331)
[19] *W. Grass:* Aufteilung von Schaltfunktionen auf programmierbare Logic Array (PLA) (Digital Processes, Vol. 6., 1980.)
[20] *Kalmár P.:* Vezérlő egységek folyamatábra alapján történő logikai tervezésének egy módszere. (Kandidátusi értekezés, 1978.)
[21] *Terplán S.:* Aszinkron jellegű logikai hálózatok tervezésének egy módszere folyamatábra alapján (Kandidátusi értekezés, 1981.)

Kedves Jegyzetfelhasználó!

A jó jegyzet nagyon hatékony segítség a tanulásban. A legjobb jegyzeteket pedig még aktív mérnökként is használni lehet. Egyetemi tanulmányai alatt valószínűleg különböző színvonalú jegyzetekkel találkozott eddig, és fog találkozni ezután. ***Kérjük, hogy ennek a kérdőívnek a kitöltésével segítse alábbi törekvéseinket:***

- ennek a jegyzetnek a következő kiadásában kevesebb sajtóhiba legyen és indokolt esetben készüljön el az átdolgozott kiadása,
- a jegyzeteket értékelni lehessen, amelynek eredményeként a legjobb jegyzetek szerzői nívódíjat kaphatnak.

Kérjük, hogy a kiküldött kérdőívet a Jegyzetbolt bejárata (V2 földszint) mellett elhelyezett gyüjtőládába dobja be.

Fáradozását köszöni az *Egyetemi Jegyzetbizottság.*

A jegyzet címe: **LOGIKAI RENDSZEREK TERVEZÉSE**

A jegyzet szerzője: **Arató Péter**

A jegyzet azonosítója: **55013**

Melyik tárgyhoz használta a jegyzetet:

Kar:

Félév:

Tárgy neve:

A jegyzet hány százalékát tudta használni (pl. 75%):

A jegyzet a tárgy anyagának hány százalékát fedte le (pl. 50%):

A jegyzet minősítése:
(0: használhatatlan, 1: nagyon rossz, 2: rossz, 3: türhető, 4: jó, 5: nagyon jó)

Javaslat átdolgozásra:

A megtalált sajtóhibák:

(A túloldalon folytatható)