Értékünk az ember
Humánerőforrás-fejlesztési Operatív Program

Keresztes Péter

DIGITÁLIS HÁLÓZATOK

SZÉCHENYI ISTVÁN EGYETEM
GYŐR

Magyarország célba ér
NEMZETI FEJLESZTÉSI TERV

Készült a HEFOP 3.3.1-P.-2004-09-0102/1.0 pályázat támogatásával.

Szerző: dr. Keresztes Péter
egyetemi docens

Lektor: dr. Sziray József
egyetemi docens

A dokumentum használata

Mozgás a dokumentumban

A dokumentumban való mozgáshoz a Windows és az Adobe Reader megszokott elemeit és módszereit használhatjuk.

Minden lap tetején és alján egy navigációs sor található, itt a megfelelő hivatkozásra kattintva ugorhatunk a használati útmutatóra, a tartalomjegyzékre, valamint a tárgymutatóra. A ◀ és a ▶ nyilakkal az előző és a következő oldalra léphetünk át, míg a Vissza mező az utoljára megnézett oldalra visz vissza bennünket.

Pozícionálás a könyvjelzőablak segítségével

A bal oldali könyvjelző ablakban tartalomjegyzékfa található, amelynek bejegyzéseire kattintva az adott fejezet/alfejezet első oldalára jutunk. Az aktuális pozíciónkat a tartalomjegyzékfában kiemelt bejegyzés mutatja.

A tartalomjegyzék és a tárgymutató használata

Ugrás megadott helyre a tartalomjegyzék segítségével

Kattintsunk a tartalomjegyzék megfelelő pontjára, ezzel az adott fejezet első oldalára jutunk.

Keresés a szövegben

A dokumentumban való kereséshez használjuk megszokott módon a Szerkesztés menü Keresés parancsát. Az Adobe Reader az adott pozíciótól kezdve keres a szövegben.

Tartalomjegyzék

1. Kombinációs hálózatok tervezése

1.1. Logikai értékek és alapműveletek

1.1.1. A logikai változók és értékeik

Több olyan megfigyelhető objektum van, amelynek mindössze két értéke létezik. Ilyenek például a logikai állítások, amelyek hamisak vagy igazak, az események, amelyek nem következnek be, vagy bekövetkeznek, a kapcsolók, amelyek nyitottak vagy zártak. A számítástechnikában az ilyen típusú változókat a nagy angol matematikusról Boole-féle változóknak (boolean) nevezzük, lehetséges értékeik a „false" és a „true". A modern digitális áramkörök technikájában a jelekhez rendelt vezetékeknek mindössze kétféle feszültségszintje lehet, egy a zérus szinthez igen közeli alacsony (low) és egy néhány volt nagyságú magas (high) szint. Az egyszerű ábrázolás kedvéért az egyik feszültségszinthez a „0", a másikhoz az „1" számot rendeljük. Leggyakrabban a „0"-t az alacsony, az „1"-et a magas szinthez rendelik. A digitális technikában ezeket a változókat, illetve jel-vezetékeket logikai változóknak nevezzük. A „0" és az „1" pedig logikai értékek.

1.1.2. A logikai alapműveletek

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

'·'	'+'	
$0 \cdot 0 = 0$	$0 + 0 = 0$	
$0 \cdot 1 = 0$	$0 + 1 = 1$	
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	$\overline{0} = 1 \quad \overline{1} = 0$

1.1.3. A logikai azonosságok

Az azonosságok olyan igazságok, amelyek a változók minden lehetséges értékére érvényesek. Bizonyításuk igen egyszerű, ha a változókat minden lehetséges értékkel helyettesítjük, és ellenőrizzük, teljesülnek-e a műveleti

táblák előírásai. Lássuk a legfontosabb azonosságokat Az első csoportban változók és konstansok (értékek), majd változók és változók között.

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

ASSZOCIATIVITÁS:

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

DISZTRIBUTIVITÁS:

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

DE-MORGAN AZONOSSÁGOK:

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

Adjunk értelmezést például az események körében az

$$A \cdot 1 = A$$

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

Különös jelentőséggel bír a két De-Morgan azonosság. Ezek közül a másodikat értelmezzük a kapcsolók körében. Legyen A és B két kapcsoló. A kettő ÉS kapcsolata egy sorosan kapcsolt kapcsolópárt reprezentál, amely csak akkor vezethet, ha mind az A, mind a B kapcsoló vezető állapotban van. A szorzat tagadása arra az állapotra utal, amikor az összetett kapcsoló nem vezet. Az azonosság jobb oldala megadja, hogy ez azzal egyenlő értékű, hogy vagy az A van nem vezető állásban, vagy a B van nem vezető állásban, vagy mindkettő nem vezető állásban van.

1.2. A kombinációs hálózat modellje

1.2.1. A kombinációs hálózat fekete doboz modellje

A kombinációs hálózat fekete doboz modelljét az 1.1. ábrán mutatjuk be. A kombinációs hálózatnak bemenetei és kimenetei vannak, valamennyi egy logikai változó, illetve logikai jel, és ennek megfelelően mindegyik csak a 0-t, vagy az 1-et veheti fel értékként. Ez a kombinációs hálózat fekete doboz modelljének egyik lényeges tulajdonsága. A bemeneteket az X_1, X_2, …, X_i, …, X_n, a kimeneteket Y_1, Y_2, …, Y_j, …, Y_m szimbólumokkal jelöltük.

A kombinációs hálózat a bemeneti jelek felett értelmezett bemeneti érték-variációkhoz a kimeneti jelek érték-variációit rendeli. (Minden bemenetihez legfeljebb csak egy kimenetit)

Például n = 3 esetben a bemeneti variációk: (0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1)

1.1. ábra. A kombinációs hálózat „fekete doboz” modellje.

Ha m = 2, akkor a kimeneti variációk halmaza: (0 0, 0 1, 1 0, 1 1).

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

Bemeneti variációk			Kimeneti variációk	
X1	X2	X3	Y1	Y2
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	1	1
1	0	1	0	1
1	1	0	1	1
1	1	1	1	0

Megjegyezzük, hogy a bemeneti értékvariáció helyett gyakran, általánosan elfogadott, de pongyola módon bemeneti kombinációt, illetve a kimeneti értékvariáció helyett kimeneti kombinációt mondanak. Pedig a szóban forgó fogalom a kombinatorikában variáció. Mivel ez a pongyola terminológia szakmai körökben is elterjedt, a következőkben mi is használni fogjuk.

Az előzőkből következik a kombinációs hálózat fekete doboz modelljének másik lényeges tulajdonsága, nevezetesen, hogy adott bemeneti kombinációra a hálózat mindig ugyanazt a kimeneti kombinációt szolgáltatja.

1.2.2. Teljesen specifikált és nem teljesen specifikált hálózat

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

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

Keressünk választ a következő kérdésre:

Hány n bemenettel és m kimenettel rendelkező teljesen specifikált kombinációs hálózat létezik?

A választ kombinatorikai módszerrel adhatjuk meg.

Mivel n bemeneti jelhez 2^n érték-variáció tartozik, m kimeneti jelhez pedig 2^m érték-variáció tartozik, annyi teljesen specifikált n bemenetű és m kimenetű különböző hálózat van, ahányszor a 2^m számú kimeneti értékvariációból ismétléssel ki tudunk választani egy 2^n hosszúságú sorozatot. Ez pedig:

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

1.3. Logikai függvények és megadási módjaik

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

1.3.1. Logikai függvény megadása igazság-táblázattal

Egy logikai függvényt igazság-táblázattal úgy adunk meg, hogy minden bemeneti variációt felsorolunk, és megadjuk a hozzájuk rendelt függvény-

értéket, azaz az (1, 0, –) hármas valamelyikét. Az ilyen táblázatot a függvény igazságtáblázatának nevezzük.

Az igazságtáblával való megadásra álljon itt egy példa:

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

1.3.2. Logikai függvény megadása algebrai kifejezéssel

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

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

Az így felírt bemeneti variációkat logikailag összeadjuk.

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

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

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

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

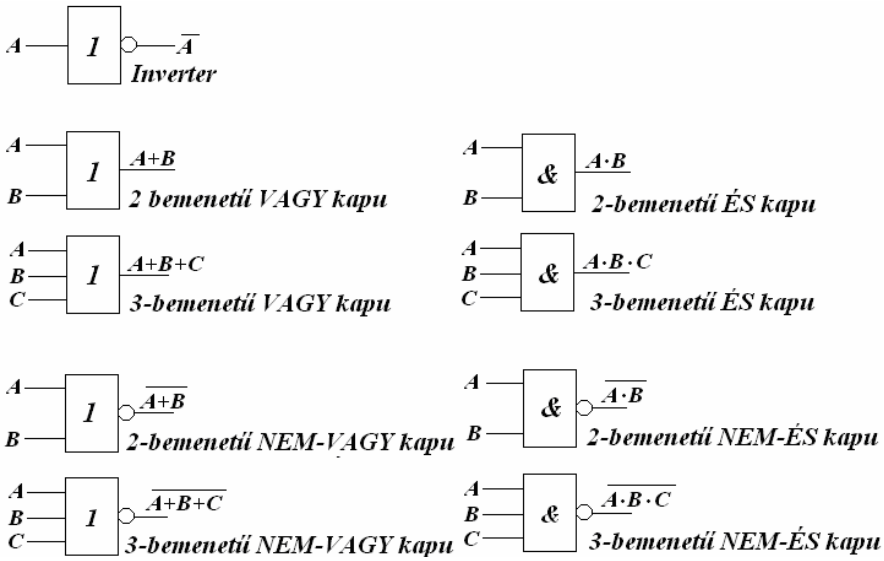
1.3.3. Logikai függvény megadása elvi logikai vázlattal

Igen gyakori megadási módszer, hogy a kanonikus alakot grafikus formában adjuk meg. A logikai műveleteket ilyenkor logikai szimbólumok rep-

rezentálják. A negálás műveletét INVERTEREK, a szorzattermeket ÉS szimbólumok, illetve ÉS kapuk, az összegzést VAGY szimbólumok, illetve VAGY kapuk reprezentálják.

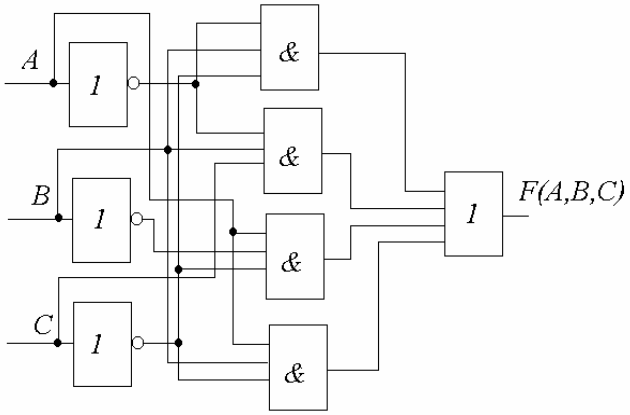
A későbbiekben gyakran fogjuk használni azokat a kapuszimbólumokat, amelyek a VAGY és az ÉS kapuktól abban különböznek, hogy a kimenetük azok logikai negáltjai. Tehát a kétváltozós NEM-VAGY kapu kimenete akkor 0, ha a bemenetek valamelyike, vagy mindkettő 1, a kétváltozós NEM-ÉS kapu kimenete pedig akkor 0, ha mindkét bemenet értéke 1. Könnyen belátható, hogy egy két-bemenetű NEM-ÉS egyik bemenetéről INVERTER-ként működik, ha a másik bemenetére állandó 1-est kapcsolunk, a NEM-VAGY pedig akkor, ha a másik bemenetére állandó 0-t kapcsolunk.

Az 1.2. ábrán megmutatjuk az elvi logikai vázlatok kapukészletének egy részét. Rajzolással nagyobb bemenetszámú kapuszimbólumokat könnyen alkothatunk, de tudnunk kell, hogy nem mindegyik általunk rajzolt kapuszimbólumhoz tartozik gyártott logikai kapu.



1.2. ábra. A logikai szimbólumok

Az 1.3. ábrán mutatjuk meg példánk inverterekkel, ÉS, valamint VAGY kapuszimbólumokkal megadott elvi logikai vázlatát.



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

1.3.4. A kétváltozós logikai függvények

A következő táblázatban megadjuk az összes kétváltozós logikai függvény definícióját.

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

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

A „0” és „1” generátorok

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

Az ÉS és a NÉS függvény, illetve kapu

Az f1 és az f14 értékei éppen egymás negáltjai. Az f1 nevezetes, csak akkor szolgáltat 1-et, ha mindkét bemeneti változó értéke 1, az f14 pedig

ekkor éppen 0-t szolgáltat. Az előbbi neve ÉS, az utóbbié NEM-ÉS, röviden NÉS.

A VAGY és NVAGY függvény, illetve kapu

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

Az ANTIVALENCIA és az EKVIVALENCIA függvény és kapu

Ha a kétváltozós függvény csak abban az esetben szolgáltat 1-et, ha a két bemenet különböző, akkor a függvény neve: ANTIVALENCIA, vagy KIZÁRÓ-VAGY. Ez az f6 függvény. Negáltja az f9, éppen ezekben az esetekben 0, és egyezés esetén 1. Ez utóbbi neve EKVIVALENCIA, vagy KIZÁRÓ-NEM-VAGY. Az ezeknek megfelelő kapuáramköröket igen gyakran használjuk. A KIZÁRÓ-VAGY művelet jelölésére gyakran használják a „⊕” szimbólumot.

1.4. A logikai függvények kanonikus alakjai

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

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

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

$$= \overline{\overline{A}B\overline{C}} \cdot \overline{\overline{A}BC} \cdot \overline{A\overline{B}\overline{C}} \cdot \overline{AB\overline{C}} =$$

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

Azt kaptuk, hogy a mintermekkel, azaz logikai szorzatok összegével adott eredeti függvény negáltja felírható olyan logikai összegek szorzataként, amelyekben ugyancsak minden változó szerepel. Ezeket az összegeket

maxtermeknek nevezzük. Vegyük nyilvántartásba a kapott maxtermeket a következő módon:

Első lépésként rendeljünk a ponált változókhoz 1-et, a negált változókhoz 0-t. Ezután rendeljük sorszámként ezekhez a maxtermekhez a kapott bináris számokhoz tartozó decimális számokat. Tegyük ugyanezt a mintermekkel is. A mintermeket kis m betűvel, egy felső és egy alsó indexszámmal jelöljük. A felső index a változók számát adja meg, az alsó az előbb kiszámított sorszám. Hasonlóan, nagy M betűvel jelöljük a maxtermeket, ugyanolyan értelmű indexekkel. Például:

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

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

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

2. $f(X_1, X_2, \ldots X_n) = m_i^n + m_j^n + \ldots m_k^n =$

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

1.5. Teljesen határozott logikai függvények egyszerűsítése

A teljesen határozott logikai függvények egyszerűsítésére lehetőségeinek négy alapvető módszert mutatunk be, nem egyforma súllyal és mélységben. Ezek:

- Egyszerűsítés algebrai módszerrel,
- Quine módszere,
- A Karnaugh-táblás módszer,
- A Quine–McCluskey-módszer.

1.5.1. Egyszerűsítés algebrai módszerrel

Egyszerűsítsük a bemutatott azonosságokat kihasználva a már megismert háromváltozós logikai függvényünket! A disztributivitást kifejező azonosságok arra mutatnak, hogy a klasszikusan kiemelésnek nevezett átalakítás

itt is végrehajtható. A logikai összeg első két szorzattermjéből és a második két szorzattermjéből is kiemelhető egy-egy kétváltozós szorzat. Ezután felismerhetjük a zárójelben lévő összegek 1 értékét, valamint azt, hogy a 1-gyel való beszorzást nem kell feltüntetni. Az eredmény kifejezés tehát azonos a kiindulásival, de jóval egyszerűbb annál.

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

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

1.5.2. Quine módszere

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

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

1.4. ábra. A Quine-módszer oszlopai

Gondoljunk arra, hogyan hajtottuk végre az algebrai egyszerűsítést az

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

és az

$$\overline{A}BC$$

mintermek közös tényezőjének, az

$$\overline{A}B$$

kiemelésével. A zárójelen belül a

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

marad. Ezzel a két összevont minterm helyén a közös szorzótényező marad, a zárójelen belül maradó változó eltűnik. A Quine-módszer lényege ennek az eljárásnak a szisztematikus ismétlése mindaddig, amíg ilyen eltüntethető változó már nem marad. Ha a változók száma n, először az n tényezős minterm párokat vonjuk így össze n−1 tényezős termekké, azután a kiadódó n−1 tényezős term-párokat n−2 tényezős termekké, azután a kiadódó n−2 tényezős termpárokat n−3 tényezős termekké, és így tovább.

Az 1.4. ábrán mutatjuk a szisztematikus eljárást ismert példánkra. Azok a mintermek, amelyekhez a függvény 1-et rendel, az I.-gyel jelölt oszlopban láthatók, bináris értékeikkel megcímezve. A II. oszlopban az összevonható mintermpárok címei és az összevonások eredményei láthatók. Ha a II. oszlopban lennének összevonható kéttényezős szorzatok, akkor a harmadik oszlop nem lenne üres.

Figyeljük meg, a kiemeléssel összevonható termek sajátossága, hogy azok mindig csak egyetlen változóban különböznek egymástól. Azt mondjuk, hogy az ilyen termek ún. Hamming-féle távolsága egységnyi.

Miután nincs több oszlop, ahová új összevonásokat írhatnánk, az oszlopokban szereplő, további összevonásokba már nem bevonható termeket prímimplikánsoknak nevezzük. Példánk prímimplikánsai:

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

A Quine-módszer következő lépése a feltétlenül szükséges prímimplikánsok kiválasztása. Ezt a prímimplikánsok lefedési táblázatának segítségével végezhetjük el (1.5. ábra). A prímimplikánsok kijelölik a táblázat sorait, az oszlopokat pedig azok a mintermek, amelyekhez a függvény 1-et rendel. Ezután „*” karakterrel bejelöljük az egyes prímimplikánsok által lefedett mintermeket.

Prímimplikáns táblázat

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

1.5. ábra. Prímimplikánsok lefedési táblája

Nyilvánvaló például, hogy az

$$\overline{A}B$$

prímimplikáns az

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

és a

$$\overline{A}BC$$

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

$$B\overline{C}$$

A lényeges, elhagyhatatlan prímimplikánsok tehát:

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

A végeredmény a lényeges prímimplikánsok logikai összege, tehát:

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

1.5.3. Logikai függvények Karnaugh-táblás egyszerűsítése

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

Háromváltozós K-tábla

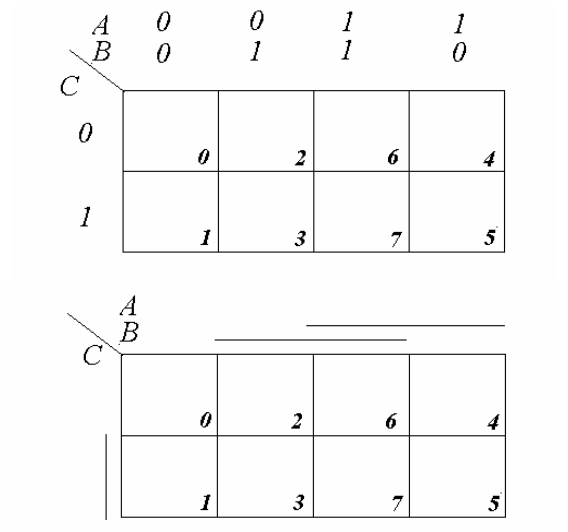
Három változó esetén 8 mintermünk van. Ezeket egy 4×2-es téglalapon helyezzük el, ügyes peremezéssel A peremezés azt jelenti, hogy a változókat két csoportra osztjuk, egy kéttagú és egy egytagú csoportra. Vízszintesen elrendezzük az első csoport lehetséges kombinációit, (0 0, 0 1, 1 1, 1 0), függőlegesen pedig a második egyváltozós csoport két lehetséges értékét, (0, 1). Figyeljünk fel arra, hogy a mintermek geometriai szomszédossága csak úgy biztosítható, ha a csoportok kombinációi is egységnyi távolságra

vannak egymástól. Ezért a különleges és szokatlan sorrend, azaz (0 1) után az (1 1)!

Nézzük az 1.6. ábrát. Ezzel a peremezéssel a négyzetek megcímezhetők a változókhoz rendelt bináris vektorokkal, ugyanakkor a négyzetek (cellák) jelölhetők is a bináris vektornak megfelelő decimális számjeggyel. Például a felső négyzetsor harmadik négyzete az (1 1 0) bináris vektorral címezhető, és a 6-os decimális értékkel jelölhető. A minterm algebrai alakját az ismert módon olvassuk ki:

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

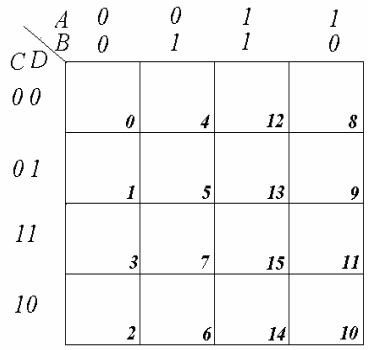
Figyeljünk fel arra is, hogy nem minden logikai szomszédosság jelenik meg geometriai szomszédosságként. Beláthatjuk, hogy a (0 0 0) szomszédos az (1 0 0) mintermmel, a (0 0 1) pedig az (1 0 1) mintermmel, bár nincsenek egymás mellett. Ezt szem előtt kell tartanunk a használatnál, de beláthatjuk, hogy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



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

Négyváltozós K-tábla

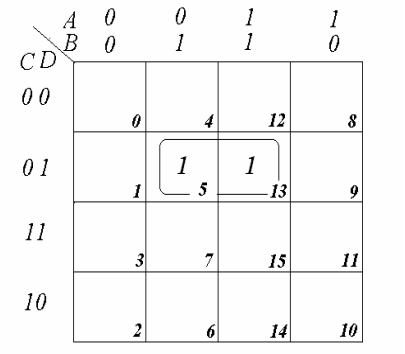
A 4-változós K-tábla összeállítása és peremezése elvi újdonságot nem jelent, de a széleken elhelyezkedő mintermek közötti szomszédosságokat érdemes megvizsgálni (1.7. ábra).



1.7. ábra. Négyváltozós K-tábla

Összevont termek kiolvasása K-táblából

Miután észrevettük két minterm összevonásának lehetőségét, ki kell olvasni az összevont termet. Tegyük fel például, hogy a fenti 4-változós K-táblán ábrázolt mintermek közül az 5-ös és a 13-as mintermek mindegyikéhez 1-et rendel egy teljesen határozott logikai függvény (1.8. ábra). Az összevont term itt egy kétnégyzetes téglalap, amelynek a címéből az A változó értéke már hiányzik, hiszen az A az a változó, amely a VAGY művelettel kiesik.



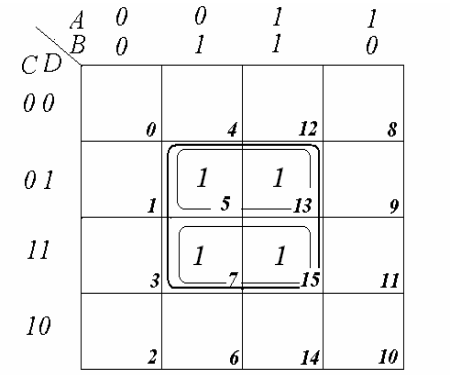
1.8. ábra. Összevont term kiolvasása

Így a cím algebrai alakja:

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

Tegyük most fel, hogy az eddigi mintermeken kívül szerepel a függvényben a 7-s és a 15-ös is (1.9. ábra). Ez utóbbiakat egymással összevonva

felismerjük, hogy az így kialakult kétmintermes két term ugyancsak szomszédos, és a C változó is kiejthető. Ezzel egy geometriailag még nagyobb, négy mintermes term adódott az összevonással, amely már csak kétváltozós. Kiolvasva: BD



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

Teljesen határozott függvények egyszerűsítése K-táblán

A K-táblás minimalizálás lépései következők:

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

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

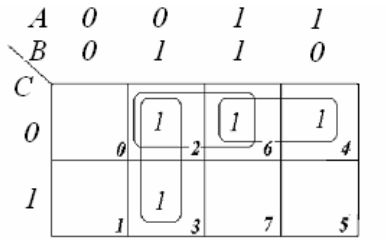
$$F(A,B,C) = \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$$

Az 1. a 2. és a 3. lépést illusztráljuk az 1.10. ábrán. Az 1-es mintermek bejelölése után megkezdődhet a párosával történő összevonás. Mivel tagjaik szomszédosak, összevonhatók a következő párok:

(0 1 0, 0 1 1), (0 1 0, 1 1 0), (1 1 0, 1 0 0).

Ezután azt vizsgáljuk, hogy a három összevont kettes csoportok között vannak-e szomszédos, nagyobbakká összevonhatók. Szembetűnő, hogy nincsenek ilyenek, tehát a három kétváltozós term mindegyike prímimplikáns.

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



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

Prímimplikánsok:

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

Ezzel a technikával befejezhetjük feladatunk egyszerűsítését: A

$$B\overline{C}$$

term felesleges, elhagyható.

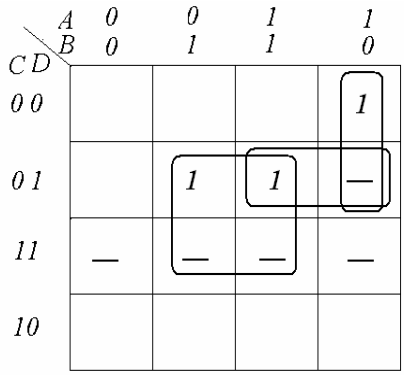
Nem teljesen határozott függvények egyszerűsítése K-táblán

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

A prímimplikánsok között egy négymintermes, tehát kétváltozós, és két kétmintermes, azaz háromváltozós term szerepel. Kiolvasva:

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

Ezek közül a második elhagyása nem változtat a teljes lefedésen.



1.11. ábra. Prímimplikánsok,
közömbös bejegyzések közötti válogatással

1.5.4. A Quine–McCluskey-módszer

A Quine–McCluskey-módszer lényegét a teljesség kedvéért bemutatjuk, de nem tárgyaljuk részletesen. Ahogyan a név is mutatja, a módszer a Quine eljárásból alakult ki, de a mintermek összevonhatóságát nem a bináris kódok közötti távolság, hanem a decimális értékek alapján vizsgálja. Könnyen megfogalmazhatók ugyanis azok a kritériumok, amelyek fennállása esetén két minterm, illetve két term összevonható. Ennek a megközelítésnek nagy előnyei, hogy a 4-5-nél nagyobb bemeneti változó szám esetén is könnyen alkalmazható, és egyszerű a számítógépes megvalósítás. A Quine–McCluskey-módszer a számítógépes logikai szintézis eljárások előfutárává vált a múlt század hatvanas éveiben.

1.6. Egykimenetű kombinációs hálózatok tervezése

1.6.1. Teljesen specifikált, egy-kimenetű hálózatok tervezése

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

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

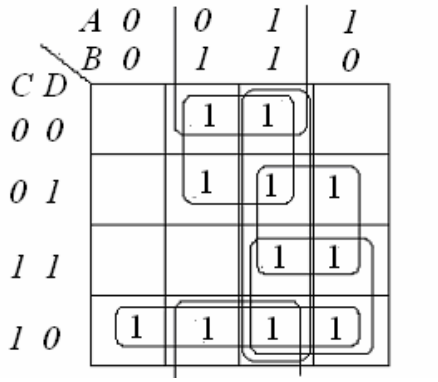
Igazságtábla vagy számjegyes minterm felsorolás esetén az 1-es mintermek táblázatba vitele után megindulhat a prímimplikánsok megkeresése, és a szükséges prímimplikánsok kiválasztása. Algebrai alak esetén a megadott szorzattermek K-táblán történő ábrázolása után megpróbálunk egy egyszerűbb lefedést találni.

1.6.2. Tervezési példa

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

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

Az 1.12. ábra mutatja a K-táblát az összevonásokkal, illetve a prímimplikánsokkal.



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

Kiolvasva a prímimplikánsokat:

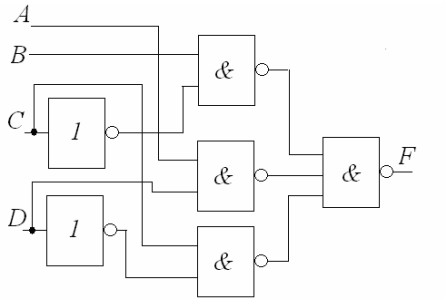
$$B\overline{D},\quad B\overline{C},\quad AD,\quad AC,\quad C\overline{D},\quad AB$$

Meghatározva a minimális irredundáns lefedést:

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

Átalakítva a kapott függvényt a De-Morgan azonosság felhasználásával az alábbi algebrai kifejezést, és az 1.13. ábra szerinti realizációt kapjuk.

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



1.13. ábra. Realizáció NÉS kapukkal

1.6.3. Nem teljesen specifikált, egykimenetű hálózatok tervezése

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

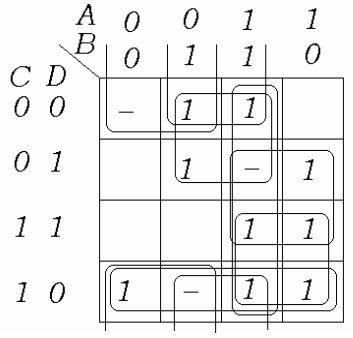
1. Egyszerűsítés Karnaugh-táblával
2. Döntés a logikai építőelemek választékáról
3. Realizáció

1.6.4. Tervezési példa nem teljesen specifikált esetre (1)

Felsoroljuk az 1-es és közömbös mintermeket:

F1: (2, 4, 5, 9, 10, 11, 12, 14, 15)
Fdc: (0, 6, 13)

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



1.14. ábra. Az F1: (2, 4, 5, 9, 10, 11, 12, 14, 15), Fdc: (0, 6, 13) függvény prímimplikánsai a K-táblán

A K-táblból kiolvasható prímimplikánsok:

$$\overline{A}\,\overline{D},\quad B\overline{D},\quad B\overline{C},\quad AD,\quad AC,\quad C\overline{D},\quad AB$$

A minimális irredundáns lefedés termjei:

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

Így az F1 logikai függvény termek logikai összegével felírt alakja:

$$F1 = B\overline{C} + AD + C\overline{D}$$

1.6.5. Tervezési példa nem teljesen specifikált esetre (2)

Ezt a feladatot igazságtáblával adjuk meg:

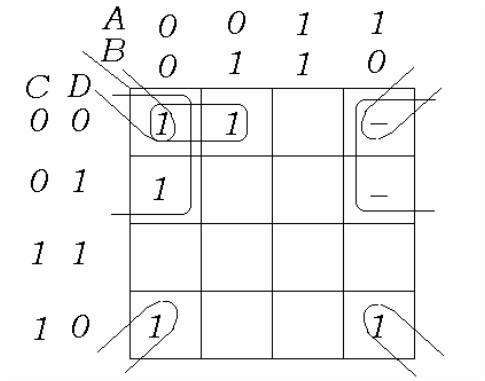
A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	–
1	0	0	1	–
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

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

A tábla alapján kapott irredundáns lefedés:

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

Láthatjuk, hogy itt nincs felesleges prímimplikáns.



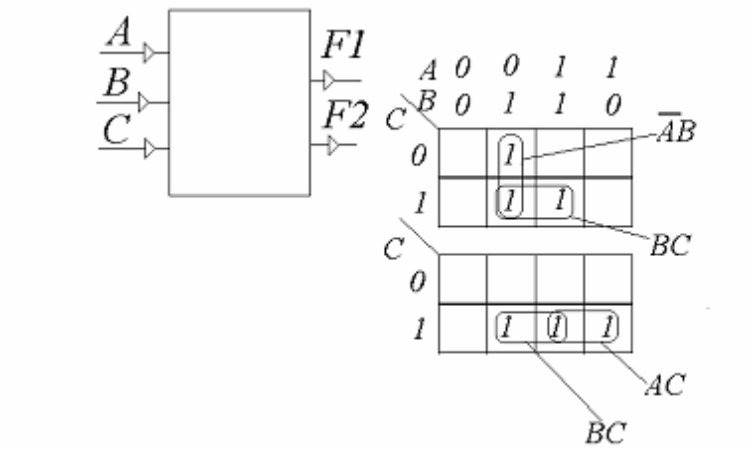
1.15. ábra. A 2-es tervezési példa K-táblája

1.7. Többkimenetű kombinációs hálózatok tervezése

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

1.7.1. Egy bevezető példa

Képzeljük el, hogy egy három-bemenetű, két-kimenetű hálózat függvényeinek lefedésekor az 1.16. ábra szerinti K-táblákra jutottunk:

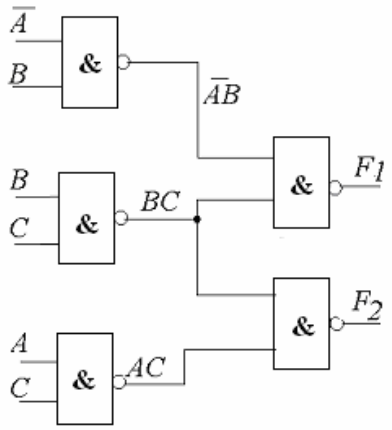


1.16. ábra. Két kimenetű hálózat függvényeinek lefedése

Ha a két kimenet függvényének prímimplikánsait egymástól függetlenül keressük meg, a következő két kifejezést kapjuk:

$$F_1 = \overline{A}B\overline{C} + \overline{A}BC + ABC = \overline{A}B + BC$$
$$F_2 = \overline{A}BC + A\overline{B}C + ABC = AC + BC$$

A két függvény prímimplikánsai között találunk egy közös, mindkét lefedésben prímimplikáns szerepet játszó termet, a BC-t.



1.17. ábra. Realizáció a közös prímimplikáns egyszeri megvalósításával

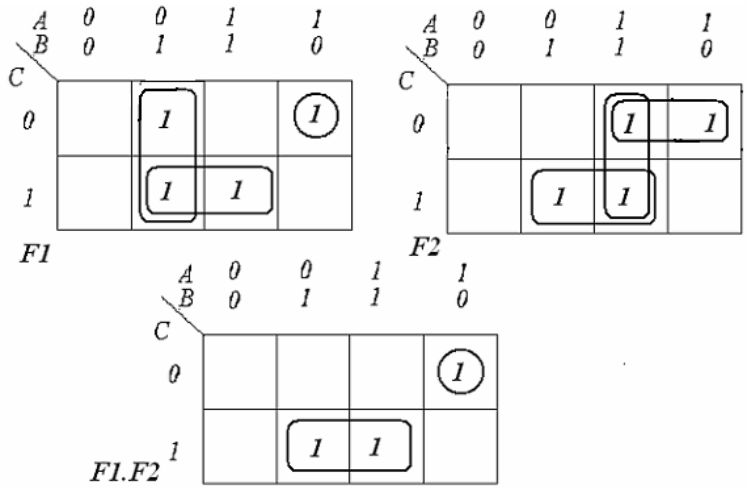
Ezt nyilvánvalóan csak egyszer, azaz egy ÉS vagy egy NÉS kapuval realizáljuk, hiszen mindkét második szinten levő VAGY, illetve NÉS kapuba bevezethető az ezt reprezentáló logikai jel (1.17. ábra). A közös prímimplikánsokat tehát célszerű megkeresni. Arra is gondolnunk kell azonban, hogy nemcsak a közös prímimplikánsok egyszeri megvalósítása egyszerűsítheti a realizációt, hanem a közös implikánsok is. Ezek közül a legnagyobbakat érdemes megkeresni.

Fontos igazság, hogy a legnagyobb közös implikánsokat a függvények szorzatának lefedésével találjuk meg. A szorzatfüggvény prímimplikánsai között ott vannak a kért függvény legnagyobb közös implikánsai, amelyek között természetesen a közös prímimplikánsok is ott vannak. A szorzatfüggvény lefedése nagyon egyszerű, a K-táblába bejegyezzük azokat a mindkét függvényben 1-es értékkel szereplő mintermeket. Így minden egyes függvény lefedéséhez nemcsak a saját prímimplikánsokat, hanem a legnagyobb közös implikánsokat is számításba vesszük.

1.7.2. Prímimplikáns készlet többkimenetű kombinációs hálózatok egyszerűsítéséhez

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

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



1.18. ábra. A legnagyobb közös implikánsok megkeresése a szorzatfüggvény prímimplikánsainak kijelölésével

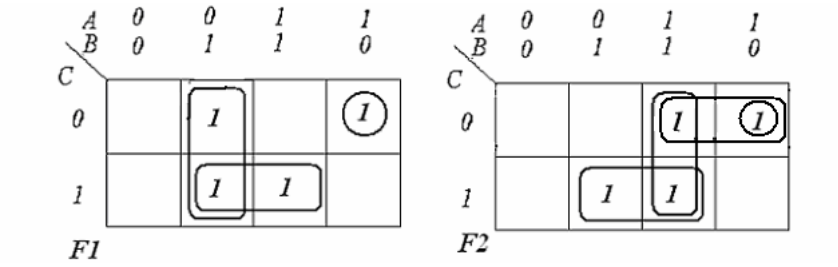
A prímimplikáns készleteket tehát a következő term-halmazokból állítjuk össze:

$$F_1: \quad \overline{A}B, \quad BC, \quad A\overline{B}\,\overline{C}$$

$$F_2: \quad BC, \quad AB, \quad A\overline{C}$$

$$F_1 \cdot F_2: \quad BC, \quad A\overline{B}\,\overline{C}$$

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



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

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

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

helyett a közös,

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

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

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

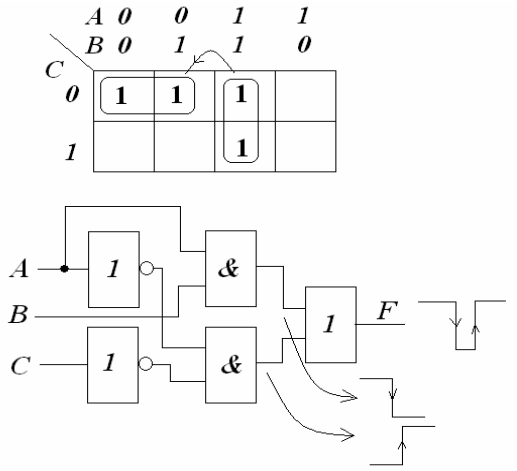
1. lépés: Megkeressük valamennyi kimenethez rendelt függvény prímimplikánsait.
2. lépés: Megkeressük valamennyi lehetséges függvény-szorzat prímimplikánsait.
3. lépés: Minden egyes kimeneti függvény mintermjeit megpróbáljuk lefedni a következő készletből:
   - a saját, más kimenetekhez nem tartozó prímimplikánsokkal,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

1.8. Hazárdok

Az eddigiek során a kombinációs hálózatokat statikusan szemléltük, nem foglalkoztunk a tranziensekkel, azaz az átmeneti jelenségekkel. Általában megköveteljük, hogy bemenet változásának hatására a kimenet a specifikációnak megfelelően reagáljon. A specifikálttól való eltérések a hazárdok.

1.8.1. A statikus hazárd keletkezése

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



1.20. ábra. A statikus hazárd keletkezése

1.8.2. A statikus hazárd meghatározása

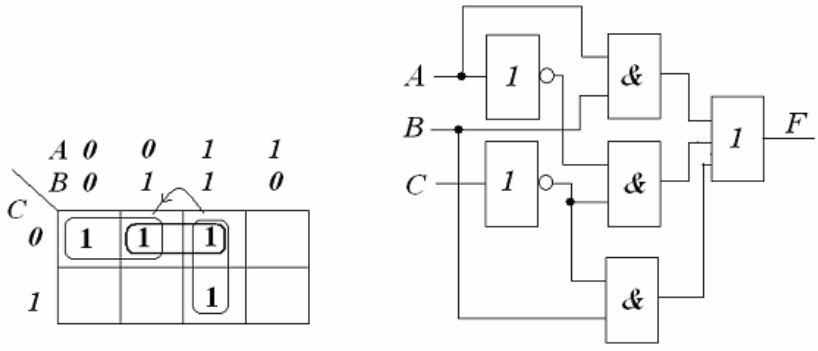
Ha egyetlen bemeneti változó logikai értékének megváltozásakor a kimenet a specifikáció szerint nem változna, a realizált hálózat kimenetén mégis átmeneti változás zajlik le, statikus hazárdról beszélünk.

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

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

1.8.3. A statikus hazárd kiküszöbölése

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



1.21. ábra. A statikus hazárd kiküszöbölése

A redundáns, ugyanakkor hazárdmentesítő prímimplikánssal együtt felírt kifejezés:

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

1.8.4. Dinamikus hazárd

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

1.8.5. Funkcionális hazárd

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

2. Sorrendi hálózatok tervezése

2.1. Elemi sorrendi hálózatok, tárolók

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

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

Az aszinkron S-R tárolónak két bemenete (S, R) és egy kimenete (Y) van. A tároló viselkedését a következőképpen adjuk meg:

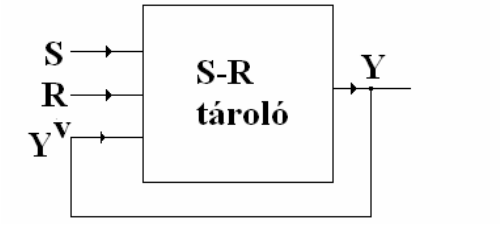
Ha a tároló mindkét bemenetére 0-t kapcsolunk, a kimenet nem változik. Ha csak az R bemenetet emeljük fel, akkor ennek hatására a kimenet, aktuális szintjétől függetlenül 0-ra áll. Ha csak az S bemenetet emeljük fel, akkor a kimenet, aktuális szintjétől függetlenül 1-re áll.

Azt, hogy mindkét bemenetet egyidejűleg felemeljük, megtiltjuk.

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

A 2.1. ábrán látjuk, hogy az áramkör kimenete visszakerül a bemenetek közé.

A specifikáció alapján megalkotható a tároló egyszerű igazságtáblája (2.2. ábra, bal oldal). Az egyszerű igazságtábla csak a bemeneti kombinációkhoz tartozó kimeneteket, azaz állapotokat tartalmazza, de ezúttal úgy,



2.1. ábra. Az S-R tároló logikai sémája

hogy a kimenetek értékei között a nullákon és az egyeseken kívül az aktuális állapot, vagy negáltja is megjelenhet. Az egyszerű igazságtábla alapján könnyen generálható az úgynevezett összetett igazságtábla (2.2. ábra, jobb oldal), amelyben a kimenet aktuális értékei a bemeneti kombinációk közé kerülnek, és az igazságtábla kimeneti változóját a kimenet következő értékeinek meghatározására használjuk. Ezt úgy is felfoghatjuk, hogy egy olyan speciális kombinációs hálózatot tervezünk, amelyen ugyanaz a változó kimenetként és bemenetként is szerepel – azaz a kimenet a bemenetre vissza van vezetve – de az igazságtáblába írt értékek különbözhetnek, hiszen időbeli eltolódás van közöttük. Formálisan meg is különböztetjük a bemenetként szereplő változót a kimenetként szereplő változótól. A bemeneten az Y szimbólumot ellátjuk egy felső „v" (visszacsatolt) felső indexszel. Ennek megfelelően az Y^v szintjeit tekintjük aktuális kimeneti értékeknek, és az Y szintjeit következő kimeneti értékeknek.

Az összetett igazságtábla kitöltésekor fontos, hogy az S=1, R=1 bemeneti kombináció tiltott, ezért ott élhetünk a közömbös kimenet előírással.

egyszerű igazságtábla

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

összetett igazságtábla

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

2.2. ábra. Az S-R tároló egyszerű és összetett igazságtáblája

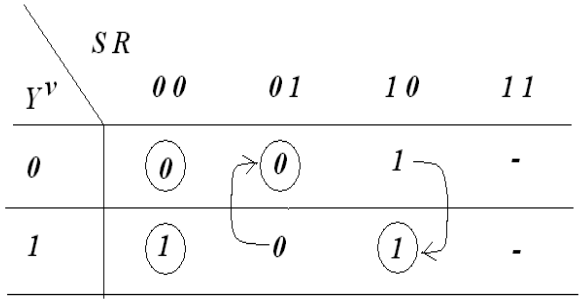
2.1.2. Az S-R tároló állapot-átmeneti táblája

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

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

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

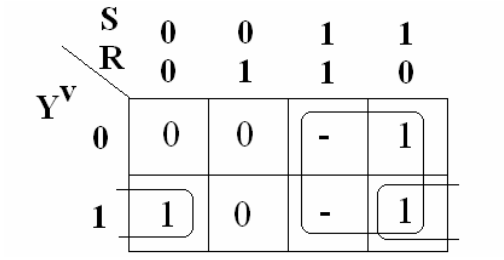
Ennek alapján az állapottábla azon kimeneti állapotai stabilak, amelyeknél a bejegyzett következő kimeneti állapot megegyezik az aktuális kimeneti állapottal.



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

2.1.3. K-tábla az S-R tároló megvalósítására

Akár az igazságtábla, akár az állapottábla könnyen átrajzolható minimalizálásra alkalmas K-táblává is (2.4. ábra).



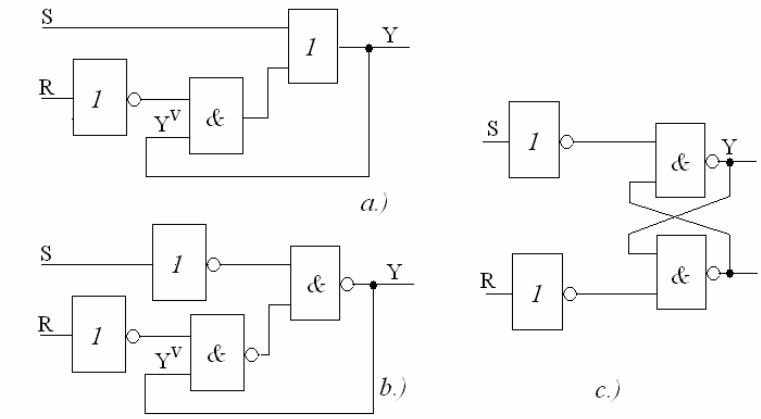
2.4. ábra. Az S-R tároló K-táblája és annak lefedése

A K-táblán elvégzett lefedésből adódó ÉS-VAGY és NÉS-NÉS realizáció algebrai kifejezése:

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

2.1.4. Az S-R tároló realizációi

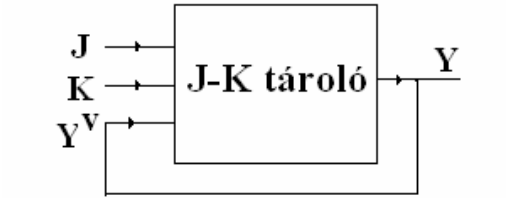
A realizációkat bemutató ábrák közül a legismertebb forma a c. részábra szerinti. Lássuk be, hogy az S-R tároló minden stabil állapotában az Y kimeneten megjelenő érték negáltja jelenik meg a másik NÉS kapu kimenetén. Ezért ezt a kimenetet gyakran jelölik az Y negáltjával.



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

2.1.5. Kísérlet J-K tároló megvalósítására

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



2.6. ábra. A J-K tároló logikai sémája

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

egyszerű igazságtábla

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

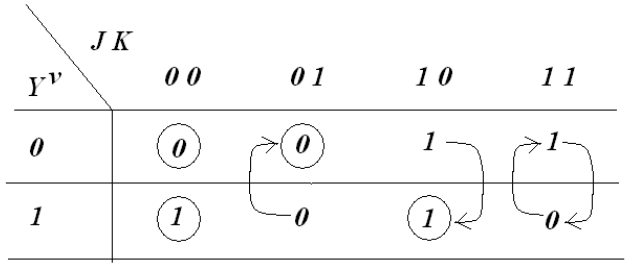
összetett igazságtábla

J	K	Y^v	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

2.7. ábra. A kísérleti J-K tároló igazságtáblái

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

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

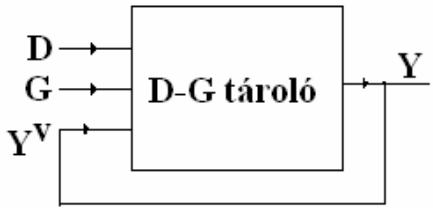


2.8. ábra. A kísérleti J-K tároló állapottáblája

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

2.1.7. A D-G tároló

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



2.9. ábra. A D-G tároló logikai sémája

Az igazságtáblák, az állapottábla a stabil állapotok bejelölésével, valamint a K-táblák a vázolt működés alapján könnyen felírhatók (2.10–2.12. ábrák).

egyszerű igazságtábla

D	G	Y
0	0	Y^V
0	1	0
1	0	Y^V
1	1	1

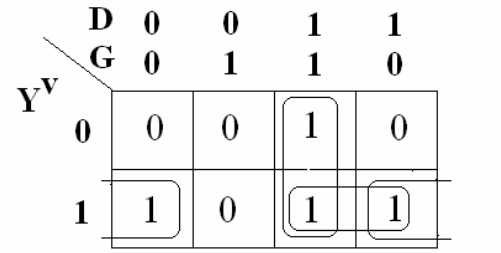
összetett igazságtábla

D	G	Y^V	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2.10. ábra. A D-G tároló egyszerű és összetett igazságtáblái

Y^V \ D G	0 0	0 1	1 0	1 1
0	(0)	(0)	(0)	1
1	(1)	0	(1)	(1)

2.11. ábra. A D-G tároló állapottáblája



2.12. ábra. A D-G tároló K-táblája

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

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

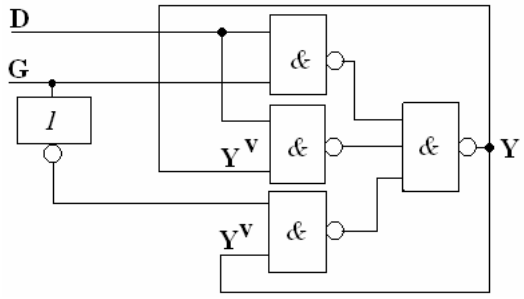
megoldás zavart okoz-e a tároló működésében.

Tegyük fel, hogy a D, G - Y^v jelek rendjében (11-1)-ben vagyunk, és G leemelésével az (10-1) helyzetbe akarunk átmenni. Ha G előbb lefut, minthogy a negált G felfutna, beállhat az (10-0) állapot, és ennek hatására a hálózat 0-ban stabilizálódna!

A helyes működés érekében kell tehát a statikus hazárdtól való mentesítés. Ezzel a realizáció ÉS-VAGY alakja:

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

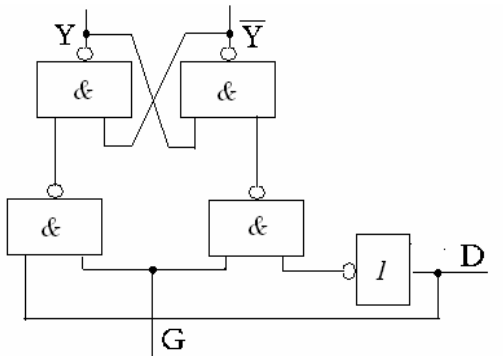
A NÉS-NÉS realizációt a 2.13. ábrán láthatjuk.



2.13. ábra. A D-G tároló statikus-hazárdmentesített NÉS-NÉS realizációja

2.1.8. A D-G tároló egy ekvivalens alakja

A 2.14. ábrán látható megoldás könnyen kialakítható az ismert S-R tároló sémájából. Belátható, hogy ez a megoldás teljesen ekvivalens a statikus hazárdmentes lefedéssel előállított változattal. Bizonyítsuk be az ekvivalenciát algebrai módszerrel!



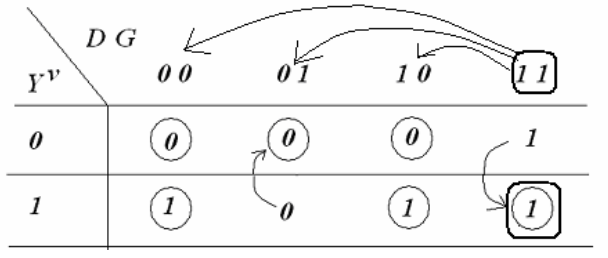
2.14. ábra. A D-G tároló egy ismert alakja

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

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

2.1.10. Többszörös bemeneti váltás a D-G tárolón

Vizsgáljuk a 2.15. ábra alapján realizált D-G működését olyan bemeneti kombináció váltásoknál, amikor egyszerre mindkét bemenetet változtatjuk.



2.15. ábra. A D-G viselkedése többszörös bemenetiszint-váltásoknál

Vizsgáljuk például a DG – Y^v jelek rendjében az 1 1-1 helyzetből előidézett 0 0 bemeneti átmenet hatását. Azt várjuk, hogy a hálózat megtartja az 1-est, hiszen az állapottábla szerint a 0 0-1 nél stabil 1-es bejegyzés szerepel. Sajnos ezt két okból sem garantálhatjuk. Ezek a következők:
Két bemenetet tökéletesen egy időben nem tudunk változtatni.

A két bemeneti változás hatása különböző késleltetésű utakon érvényesül

A valóságban tehát nem lehet kizárni, hogy vagy az 1 0, vagy a 0 1 bemeneti kombináció átmenetileg beáll. Így ha a 0 1 jelentkezik, azaz D lefutásának hatása előbb érvényesül, a kimeneten beállhat a 0 tranziens, amely végül, miután G is lefut, az Y = 0 kimenetet stabilizálja.

Legjobb, ha megtiltjuk, a többszörös bemeneti váltásokat, azaz egyszerre csak egyetlen egy bemeneti jel értéke változhat meg.

2.1.11. A D-G tároló „átlátszósága”

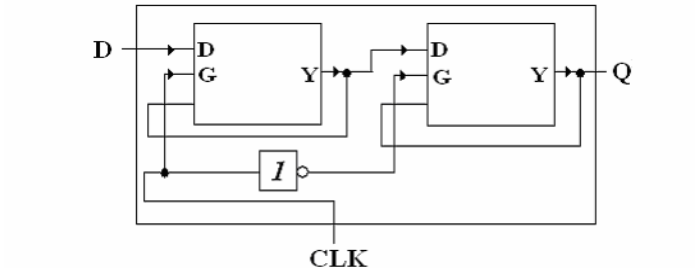
A D-G tároló sajátossága, hogy a G=1 helyzetben a D-re adott változások kijutnak a kimenetre. A G=1 helyzetben tehát a tároló a D-bemenet felől „átlátszó” (transzparens).

Felmerült az igény olyan tároló előállítására, amely a beírási folyamat alatt sem átlátszó. Az ilyen tárolókkal ismerkedünk meg a következő pontokban.

2.2. Mester-szolga tárolók (Flip-flopok)

2.2.1. A D- típusú mester-szolga tároló

Kapcsoljunk össze két D-G tárolót a 2.16. ábra szerinti módon, és az így létrehozott egység bemenetét jelöljük D-vel, kimenetét Q-val. Azt a bemenetet, amelyről az első tároló G bemenetét vezéreljük, és amelynek negált változója a másik tároló beírását vezérli, speciális funkcióval ruházzuk fel: ÓRA (CLOCK, CLK) lesz a neve, illetve funkciója. A működést analizálva beláthatjuk, hogy az ÓRA magas értékénél a D bemenet szintje beíródik az első D-G tárolóba, de eközben a második tároló kimenete változatlan, hiszen a G bemenetére 0 jut. Az ÓRA lefutásakor az első fokozat átlátszatlanná válik, a kimenetének értéke azonban átíródik a második tároló kimenetére. A beírás egy óra-ciklussal megtörtént, de úgy, hogy a teljes tároló ez alatt átlátszatlan maradt, hiszen egyik fokozata mindkét órajel-helyzetben átlátszatlan volt. Az ilyen két ütemben beírható tárolókat nevezzük MESTER-SZOLGA (MASTER-SLAVE) tárolóknak, mivel az első fokozatba beírt értéket a második szolgai módon bemásolja.



2.16. ábra. A D-MS tároló megvalósítása D-G tárolókból

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

- A D bemenet mintavételezése és a mintavételezett érték tárolása, miközben a Q kimenet változatlan, őrzi az utolsóként beállt értéket.
- A Q kimenetre a mintavételezett érték rákapcsolása és tárolása, miközben a D bemenet változásai már hatástalanok maradnak.

Az analízis eredményeképpen felvehetjük a D-MS tároló egyszerű igazságtábláját. Az ilyen órajel-vezérelt tárolóknál a következő kimeneti állapot jelölésére az (n+1) felső indexet szokás alkalmazni, mivel az aktuális kimeneti állapotot az n. órajel ciklus eredményének tekintjük, és így n felső indexet alkalmazunk a jelölésre. Azaz az aktuális kimenet jele Q^n, a következőé Q^{n+1}.

Feltűnő, hogy a D-MS tároló következő kimeneti állapota nem függ az aktuális kimeneti állapottól, csakis a D bemenettől. Az összetett igazságtábla tehát azonos az egyszerűvel (2.17. ábra).

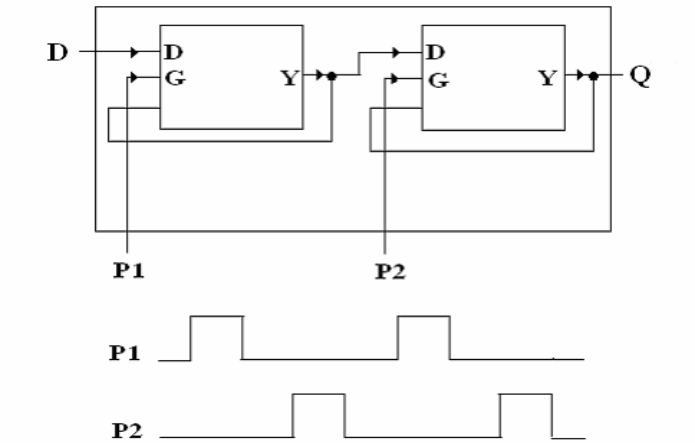
egyszerű igazságtábla

D	Q^{n+1}
0	0
1	1

2.17. ábra. A D-MS tároló igazságtáblája

2.2.2. A D-MS tároló, kétfázisú órajellel

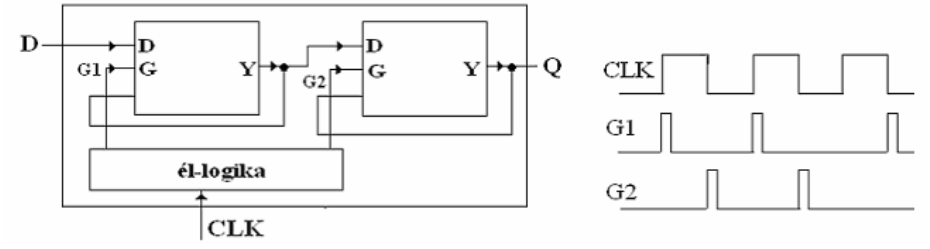
Ugyanez a tároló másféle, az átlátszatlanságot biztonságosabban garantáló órajel elrendezéssel is megvalósítható. A 2.18. ábrán látható kétfázisú, nem átlapolt órajelekkel a két komponens-tároló működése időben egymástól biztonságosan elválasztható.



2.18. ábra. Kétfázisú D-MS

2.2.3. Az élvezérelt D-MS tároló

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



2.19. ábra. Élvezérelt D-MS

2.2.4. A J-K MS tároló

A D-MS tárolóból kiindulva valósítsuk meg a J-K-MS tárolót. A kimenet visszacsatolása a bemenetre nem okozhat instabilitást, hiszen a D-MS tároló nem átlátszó, így minden egyes állapotváltás az órajel ütemében megy végbe. A J-K-MS egyszerű és összetett igazságtáblái a 2.20. ábrán láthatók.

egyszerű igazságtábla

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

összetett igazságtábla

J	K	Q^n	Q^{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

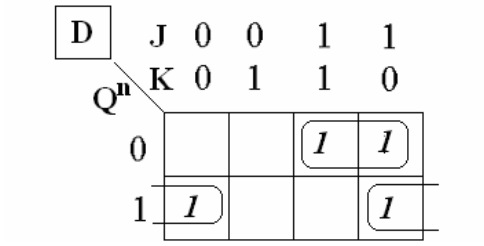
2.20. ábra. A J-K-MS egyszerű és összetett igazságtáblája

A D-MS tárolót a következő módon használjuk fel: tudjuk, hogy az órajel felfutása előtt a D bemenetre kell kapcsolnunk azt a logikai szintet amit a lefutáskor, mint a következő kimeneti állapotot, a kimeneten látni kívánunk. Ez azt jelenti, hogy egy olyan kombinációs hálózatot kell tervezni a J, K bemenetek és a D közé, amelynek igazság-tábláját a J-K tároló összetett táblájából könnyűszerrel megkapunk, ha a Q^{n+1}-et D-re cseréljük (2.21. ábra).

J	K	Q^n	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

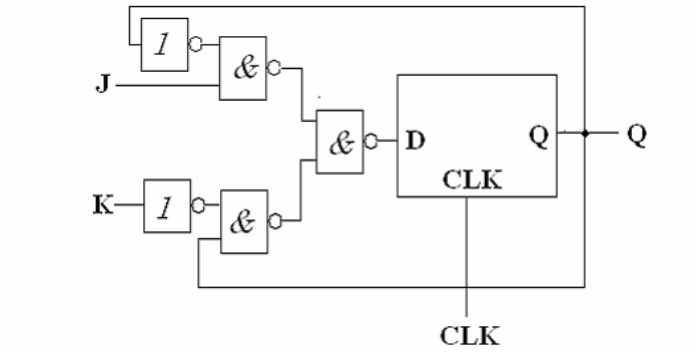
2.21. ábra. A D-bemenetre érvényes igazságtábla

Ezután a 2.22. ábrán látható K-tábla segítségével realizálható a D-t meghajtó hálózat (2.23. ábra).



2.22. ábra. A J-K MS K táblája és lefedése

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



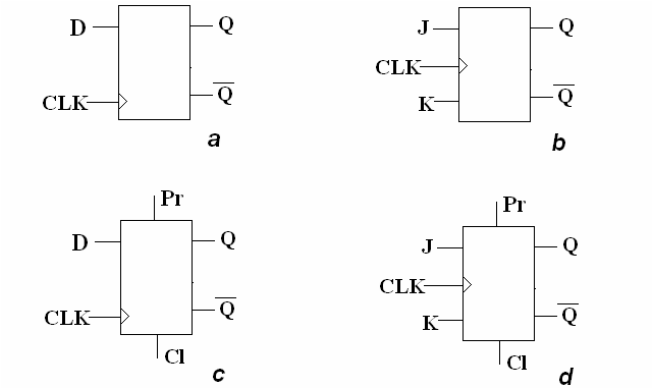
2.23. ábra. A J-K-MS megvalósítása D-MS felhasználásával

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

2.2.5. Flip-flopok segéd-bemenetei és szimbólumaik

A 2.24. ábrán bemutatjuk azokat a szimbólumokat, amelyekkel az élvezérelt MS tárolókat jelölni szokták. Tudnunk kell, hogy mindkét fajta flip-flopot gyakran kiegészítik két bemenettel. A PRESET (Pr) bemenet a tárolót a funkcionális bemenetektől függetlenül 1-be, a CLEAR (Cl) a funkcionális bemenetektől függetlenül 0-ba állítja. Ezekkel a bemenetekkel igen egyszerűen állíthatjuk be a szinkron sorrendi hálózatok kezdeti állapotát.

Megjegyezzük, hogy a Pr és Cl bemenetek egyes flip-flopoknál az órajeltől függetlenül állítják be a kimenetet, másoknál az órajel valamelyik élének hatására. Előbbieket aszinkron Pr-Cl bemeneteknek, utóbbiakat szinkron Pr-Cl bemeneteknek nexvezzük.

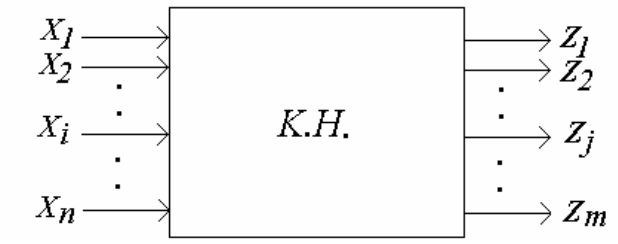


2.24. ábra. Preset és Clear bemenetekkel nem, és azokkal is rendelkező, élvezérelt flip-flopok szimbólumai

2.3. A sorrendi hálózatok modelljei, alaptípusai

2.3.1. A kombinációs hálózatok egy újabb fekete doboz modellje

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



2.25. ábra. A kombinációs hálózat fekete doboz modellje

A modellt a következő logikai egyenletrendszerrel írhatjuk le:

$$Z_1 = f_{z_1}(X_1, X_2, \ldots X_i, \ldots X_n)$$
$$Z_2 = f_{z2}(X_1, X_2, \ldots X_i, \ldots X_n)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Z_j = f_{zj}(X_1, X_2, \ldots X_i, \ldots X_n)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Z_m = f_{zm}(X_1, X_2, \ldots X_i, \ldots X_n)$$

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

$$f_z : \mathbf{X} \Rightarrow \mathbf{Z}$$
$$\mathbf{X} : \textit{a bemenetikombinációk halmaza}$$
$$\mathbf{Z} : \textit{a kimeneti kombinációk halmaza}$$

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

$$f_z : x_i \rightarrow z_j$$
$$x_i : \textit{egy bemeneti kombináció}$$
$$z_j : \textit{egy kimeneti kombináció}$$

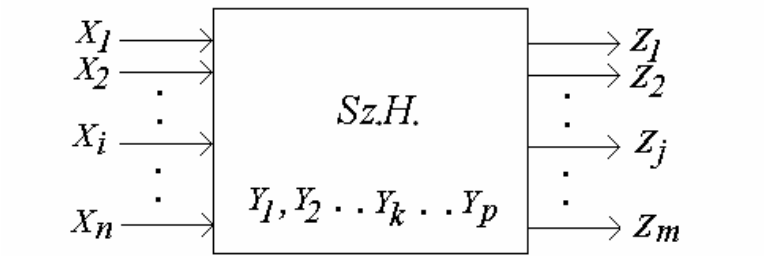
A kombinációs hálózat viselkedésének legfontosabb sajátossága, hogy egy meghatározott bemeneti kombináció ismételt rákapcsolásaira a tranziens idő eltelte után mindig ugyanazt a kimeneti kombinációt szolgáltatja, függetlenül attól, hogy az adott bemeneti kombináció két rákapcsolása között milyen más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

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

2.3.2. A sorrendi hálózatok modelljei

A sorrendi hálózatok „fekete doboz" modellje formájában nem, csak viselkedésében különbözik a kombinációs modelltől. Ez azt jelenti, hogy a sorrendi hálózat ugyanarra a bemeneti kombinációra rendre más és más kimenő-kombinációt szolgáltathat a kimenetein. Másképpen megfogalmazva: a kimeneti kombináció nem csak a pillanatnyi bemeneti kombinációtól függ, hanem a korábbi bemeneti kombinációktól, sőt azok sorrendjétől is függ.

Ez csak úgy lehetséges, a kimeneti kombinációk nem csak a bemenetektől függenek, hanem a dobozon belül „elrejtett" szekunder változóktól (Y_1 . . . .Y_p) is. A modellt leíró logikai függvények tehát összetettebbek. Két logikai függvénycsoportot kell megadnunk. Az első csoport a kimeneteket adja meg a bemenetek és a szekunder változók függvényében, a másik az állapotváltozók új értékeit határozzák meg a bemenetek és az éppen fennálló (aktuális) szekunder változó értékek alapján. A 2.26. ábra mutatja a modellt.



2.26. ábra. A szekvenciális hálózat modellje

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

$$Z_1 = f_{z_1}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$
$$Z_2 = f_{z2}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$
. . . . . .
$$Z_j = f_{zj}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$
. . . . . .
$$Z_m = f_{zm}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$

$$Y'_1 = f_{y_1}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$
$$Y'_2 = f_{y2}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Y'_k = f_{yk}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Y'_p = f_{yp}(X_1,\ldots X_i,\ldots X_n, Y_1,\ldots Y_k,\ldots Y_p)$$

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

$$f_z : \mathbf{X}\times\mathbf{Y} \Rightarrow \mathbf{Z}$$
$$f_y : \mathbf{X}\times\mathbf{Y} \Rightarrow \mathbf{Y}$$
$$\mathbf{X} : \textit{a bemenetikombinációk halmaza}$$
$$\mathbf{Z} : \textit{a kimeneti kombinációk halmaza}$$
$$\mathbf{Y} : \textit{a szekundér változók halmaza}$$

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

$f_y : (x_i^t, y_k^t) \rightarrow y_l^{t+\Delta t}$

$f_z : (x_i^t, y_k^t) \rightarrow z_j^t$

x_i^t : *egy bemeneti kombináció a t pillanatban*

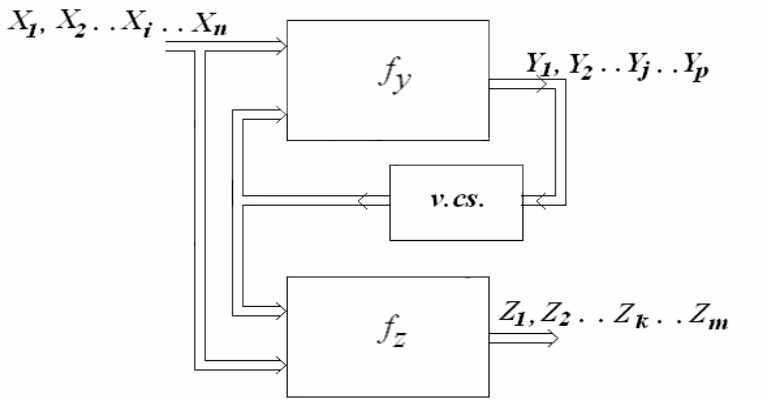
z_j^t : *egy kimeneti kombináció a t pillanatban*

y_k^t : *egy szekundér változó kombináció a t pillanatban*

$y_l^{t+\Delta t}$: *egy szekundér változó kombináció a* $(t+\Delta t)$ *pillanatban*

2.3.3. A Mealy-típusú sorrendi hálózat

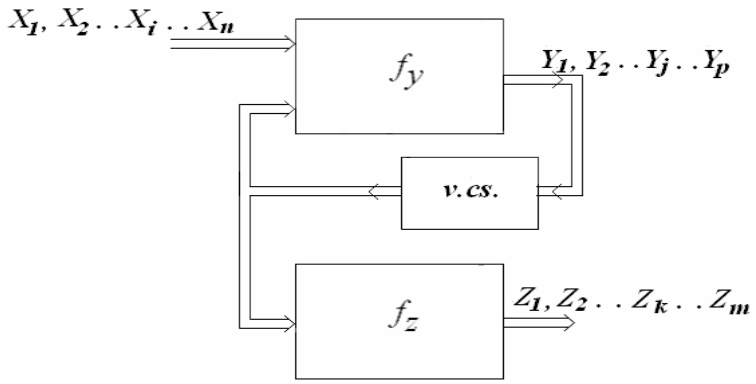
A sorrendi hálózat legáltalánosabb belső struktúráját mutatja a 2.27. ábra. A struktúra az előző pontban bemutatott függvény-kapcsolatokat is tükrözi. Azt a sorrendi hálózatot, amelynek f_z kimeneti hálózatára – az egyenletekkel is bemutatott modellnek megfelelően – mind a bemenetek, mind az állapot-változók rácsatlakoznak, Mealy-típusú hálózatnak nevezzük.



2.27. ábra. A Mealy típusú sorrendi hálózat struktúrája

2.3.4. A Moore-típusú sorrendi hálózat

Azt a speciális sorrendi hálózatot, amelynek kimeneti kombinációira csak a belső állapotok hatnak, Moore-típusú sorrendi hálózatnak nevezzük. Azt is mondhatjuk, hogy a Moore-féle sorrendi hálózatban a kimeneti kombinációk a belső állapotok átkódolt formáit szolgáltatja a kimeneteken (2.28. ábra).



2.28. ábra. A Moore-típusú sorrendi hálózat struktúrája

2.3.5. Az aszinkron sorrendi hálózat

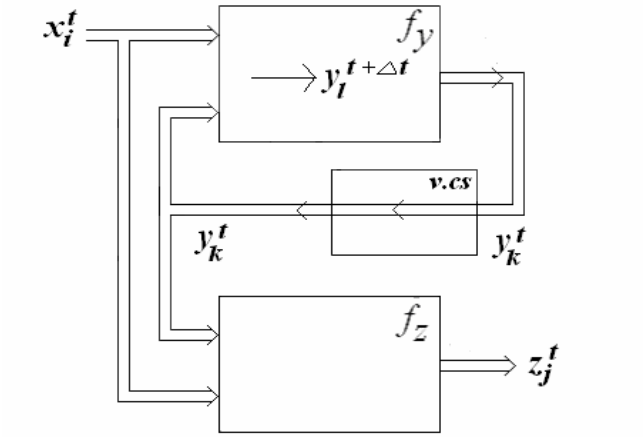
Az aszinkron sorrendi hálózat f_y hálózatának visszacsatolása órajel nélküli, direkt visszacsatolás. Ezt a direkt visszacsatolást vagy közvetlenül, huzalozással hozzuk létre, vagy S-R tárolókat helyezünk el a visszacsatoló körben. Mindkét módszer közös sajátossága, hogy a visszacsatolás a hálózat saját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

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

Közvetlen visszacsatolású aszinkron sorrendi hálózat

A 2.29. ábra a közvetlenül, vezetékekkel visszacsatolt aszinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor egy új bemeneti kombinációt már rákapcsoltunk a hálózatra, és az f_y kombinációs hálózat belsejében már megindult a következő állapot kombináció generálása. Az f_y hálózat kimenetein azonban a változás még nem jelent meg, az egyenlőre még az aktuális állapot kombinációt őrzi. Az f_z kimenetein ugyancsak átmeneti állapot van, hiszen a bemeneti kombináció már megváltozott, az aktuális állapot-kombináció ugyanakkor még uralkodik a bemenetein.

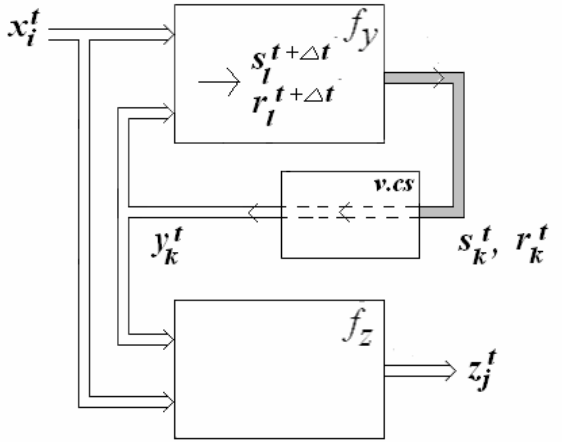
Tegyük fel, hogy a következő állapot kombináció az új bemeneti kombinációval olyan párost alkot az f_y bemenetein, amelyhez az f_y hálózat az ugyanezt a következő állapot-kombinációt rendeli. Ez azt jelenti, hogy az új állapot-kombináció stabilizálódik. Így végül a kimeneti kombináció is nyugalomba jut,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



2.29. ábra. Közvetlen visszacsatolású aszinkron sorrendi hálózat

S-R tárolókkal visszacsatolt aszinkron sorrendi hálózat

A 2.30. ábra az S-R tárolókkal visszacsatolt aszinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor egy új bemeneti kombinációt már rákapcsoltunk a hálózatra, és az f_y kombinációs hálózat belsejében már megindult a következő állapotot generáló S-R kombináció kialakulása. Az f_y hálózat kimenetein azonban a változás még nem jelent meg, az egyenlőre még az aktuális állapotot generáló S-R kombinációt őrzi. Az f_z kimenetein ugyancsak átmeneti állapot van, hiszen a bemeneti kombináció már megváltozott, az aktuális állapot-kombináció ugyanakkor



2.30. ábra. Az S-R tárolókkal visszacsatolt aszinkron sorrendi hálózat

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

2.3.6. A szinkron sorrendi hálózat

A szinkron sorrendi hálózat f_y hálózatának visszacsatolása órajellel, MS tárolókon keresztül érvényesül. A gyakorlatban vagy D-MS, vagy J-K-MS tárolós visszacsatolást használnak. Mindkét módszer közös sajátossága, hogy az órajel minden állapot kombináció fennállásának idő-intervallumát egyértelműen kijelöli, függetlenül attól, hogy az f_y visszaadja-e a rákapcsolt aktuális állapot-kódot, vagy nem. Így feleslegessé válik az instabil és a stabil állapotok megkülönböztetése.

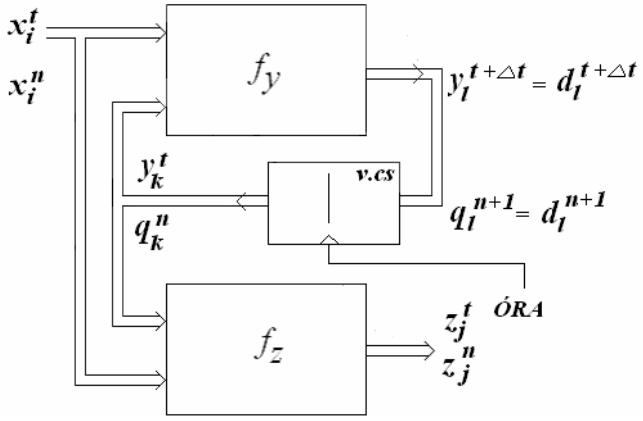
A szinkron hálózatok az egymást követő állapotokat és kimeneti kombinációkat időben diszkrét sorozatokká alakítja. Ennek megfelelően sajátos jelöléseket alkalmazhatunk, a t időbeli kombinációkat inkább az n természetes számmal, a $t+\Delta t$ időbeli kombinációkat inkább az $n+1$ természetes számmal, mint sorszámokkal jelöljük. A szinkron tárolók kimeneti kombinációinak jelölésére inkább a már megismert q szimbólumot használjuk. Megjegyezzük, hogy az ábrákon mindkét jelölés-rendszer látható, de a későbbiekben szinkron hálózatok esetén csak a most bevezetett jelölés-rendszert alkalmazzuk.

D-MS flip-flopokkal visszacsatolt szinkron sorrendi hálózat

A 2.31. ábra egy D-MS flip-flopokkal visszacsatolt szinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor az $(n-1)$. órajelciklus már lejátszódott, és az n. felfutó élre várunk. Ennek megfelelően a hálózatra már rákapcsoltuk az n. ütemnek megfelelő bemeneti kombinációt, és megjelent az ennek megfelelő kimeneti kombináció is.

Az f_y kombinációs hálózat a kimenetein előállítja a tárolók bemeneteinek kombinációját. Ezek azonos kombinációk a megkívánt következő állapot kombinációkkal, hiszen a D típusú tárolók kimeneteiken megismétlik a bemeneteikre kerülő értékeket. Míg egy adott d_l^{n+1} kombináció a visszacsatoló flip-flopok bemenetein várakozik, az f_y hálózatra csatlakozó kimeneteik változatlanok, és az aktuális állapot-kombinációt, a q_k^n-t őrzik.

Az f_z kombinációs hálózat a bemenetére jutó aktuális állapot-kombináció és a bemeneti kombináció eredményeképpen szolgáltatja az aktuális kimeneti kombinációt. Ekkor érkezik az órajel felfutó éle. A következő állapotkód a MESTER tárolókba kerül, miközben a SZOLGA kimenetek változatlanok.



2.31. ábra. D-flip-flopokkal visszacsatolt szinkron sorrandi hálózat

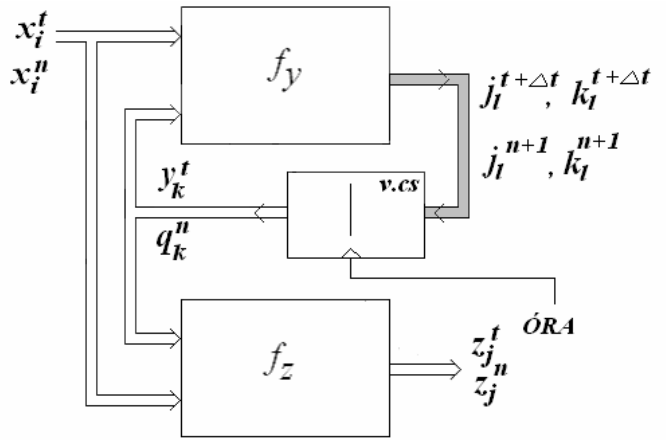
A következő változás akkor következik be, amikor az órajel lefutó éle megérkezik. Ennek hatására a D-MS tárolók kimenetein megjelenik az eddig következő állapotkombinációnak nevezett kombináció, és aktuális állapottá válik. Ha ezt követően megváltoztatjuk a bemeneti kombinációt, akkor az f_z kimenetein egy új aktuális kimeneti kombináció, és az f_y kimenetein egy újabb következő állapot kódja jelenik meg.

J-K MS flip-flopokkal visszacsatolt sorrendi hálózat

A 2.32. ábra J-K MS flip-flopokkal visszacsatolt szinkron sorrendi hálózat struktúráját egy olyan pillanatban ábrázolja, amikor az (n−1). órajel-ciklus már lejátszódott, és az n. felfutó élre várunk. Ennek megfelelően a hálózatra már rákapcsoltuk az n. ütemnek megfelelő bemeneti kombinációt, és megjelent az ennek megfelelő kimeneti kombináció is.

Az f_y kombinációs hálózat a kimenetein előállítja a tárolók bemeneteinek kombinációját. Ezek nem azonos kombinációk a megkívánt következő állapot kombinációkkal, hiszen a J-K tárolók bemeneteire olyan kombinációkat kell adnunk, amelyek a következő állapot kombinációkat majd az órajel ciklus lejátszódásakor generálják. Míg egy adott j_l^{n+1}, k_l^{n+1} kombináció a visszacsatoló flip-flopok bemenetein várakozik, az f_y hálózatra

csatlakozó kimeneteik változatlanok, és az aktuális állapot-kombinációt, a q_k^n-t őrzik. Az f_z kombinációs hálózat a bemenetére jutó aktuális állapot-kombináció és a bemeneti kombináció eredményeképpen szolgáltatja az aktuális kimeneti kombinációt. Ekkor érkezik az órajel felfutó éle. A következő állapotkód a MESTER tárolókba kerül, miközben a SZOLGA kimenetek változatlanok.



2.32. ábra. J-K flip-flopokkal visszacsatolt szinkron sorrendi hálózat

A következő változás akkor következik be, amikor az órajel lefutó éle megérkezik. Ennek hatására a J-K MS tárolók kimenetein megjelenik az eddig következő állapot-kombinációnak nevezett kombináció, és aktuális állapottá válik. Ha ezt követően megváltoztatjuk a bemeneti kombinációt, akkor az f_z kimenetein egy új aktuális kimeneti kombináció, és az f_y kimenetein egy újabb következő állapot kódja jelenik meg.

2.4. Szinkron sorrendi hálózatok tervezési folyamata mintapéldákon bemutatva

A következő fejezetekben két sorrendi hálózat modellel is megvalósítjuk a következő feladatot:

Egy hálózatra egy órajel ütemében az X1, X2 jelek érkeznek.

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

2.4.1. Az első szinkron feladat megoldása a Mealy-modell szerint

A feladat megoldásának minden lépését két alpontban mutatjuk be. Ezek közül az első az általános megfontolásokat, a második azok konkrét, az adott feladatra való alkalmazását mutatja be.

Egy Mealy-modell felvázolása állapot-átmeneti gráffal és előzetes állapot-átmeneti táblával

A verbálisan megadott szinkron hálózattervezési feladat megoldásának első lépése, hogy a működést megpróbáljuk egy állapot-átmeneti gráfon, majd egy állapot-átmeneti táblán is megfogalmazzuk. Meg kell jegyeznünk, hogy a szóbeli specifikáció minden többértelműségét csak ezek a módszerek tisztázhatják. Értelmezzük a feladatot úgy, ahogyan azt a 2.33. ábrán látható gráf, illetve táblázat tartalmazza.

Állapot-átmeneti gráf

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

Előzetes, szimbolikus állapottábla

Az előzetes szimbolikus állapottáblában ugyanazokat az információkat rögzítjük, mint az állapotgráfon. Ennek szerkezetét a tárolók, illetve flip-flopok tervezéséből már ismerjük. Ez azonban az állapotokat absztrakt formában tartalmazza, a konkrét állapot-kombinációk megállapítását, azaz az állapot-kódolást később végezzük el. A állapottáblában tehát bemeneti-kombinációnként minden állapothoz megadjuk a következő állapot szimbólumát, és a „/” jellel elválasztva a megkívánt kimeneti kombinációt.

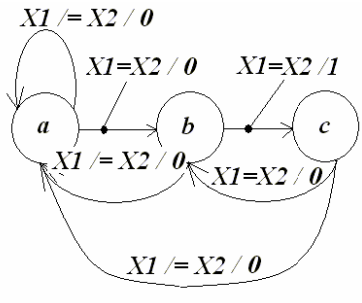
Mivel a továbbiakban az állapottáblával dolgozunk tovább, az állapotgráf felvétele kihagyható, ha azonnal fel tudjuk venni az állapottáblát.

A feladat állapotgráfja és előzetes, szimbolikus állapottáblája

Feladatunk megoldásához ezúttal Mealy-típusú gráfot választottunk, később megmutatjuk e feladat megoldását Moore-típusú hálózattal is. Úgy képzeljük, hogy hálózatunk bekapcsolás után egy *a* jelű kezdeti állapotba kerül. Az ezután jelentkező első órajel-ciklus eredménye attól függ, milyen bemeneti kombináció érkezik. Ha X_1 nem egyenlő X_2-vel (01 vagy 10), ez nem kedvező számunkra, hiszen hálózatunknak a második egyforma szint-állást kell detektálnia, ehhez pedig az első egyforma szint-állásnak előbb meg kell érkeznie. Tehát, mindaddig, amíg 01 vagy 10 érkezik a bemenetre, maradunk a kezdeti állapotban, és a kimenet, Z alacsony szinten marad. Azt viszont, hogy megérkezik az első 00 vagy 11, a hálózatnak meg kell jegyeznie, hogy a második együttállást jelezni tudja. Tehát az *a* állapotból a 00 vagy az 11 bemeneti kombinációkra a *b* állapotba kerül a hálózat, és a *b* szimbólum jelentése, hogy megjött az első, számunkra kedvező bemeneti kombináció. Persze az *a*-ból *b*-be tartó átmenet alatt Z=0 marad, hiszen ez még csak az első kedvező bemeneti kombináció.

Ha hálózatunk a *b* állapotban van, és a következő órajel-ciklust 01 vagy 10 bemeneti kombináció fogadja, akkor hálózatunkat reményt vesztve vissza kell irányítani a kezdeti *a* állapotba, a Z nullán tartása mellett. Ezzel szemben, ha a 00 vagy az 10 érkezik, hálózatunk a *c* állapotba kerül, és a Z = 1 értékkel jelzi, hogy megjött a második együttállás. A *c* állapotból való elágazásnál, ha 01 vagy 10 érkezik, a kezdőállapotba kell mennünk, hiszen újra az első együttállásra kell várni. Jöhet azonban 00 vagy 11 is, és akkor máris a *b* állapotba kell mennünk. Mindkét esetben persze a Z = 0.

Az előzetes, szimbolikus állapottáblába csak azokat az információkat rögzítjük, amelyeket az állapotgráffal megadtunk.



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

2.33. ábra. A specifikáció állapotgráffal és szimbolikus állapottáblával

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

A bemeneti egyszerűsítési lehetőségek kihasználása

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

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

A feltétlenül szükséges számú állapot megállapítása

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

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

Állapot-összevonás az adott feladatban

Az állapottábla tüzetes vizsgálatából kiderül, hogy a fenti feltétel az *a* és a *c* állapotokra teljesül. Mindkettőből kiinduló bemeneti kombinációkhoz ugyanazok a kimeneti értékek tartoznak, hiszen E = 0-nál *a*-ból Z = 0, E = 1 esetben ugyancsak, és egyformák a Z értékek az E = 1 bemenetnél is. A következő állapotok az E = 0-nál mindkettőre *a*, és E = 1-re mindkettőre *b*.

Az *a* és *c* állapot megkülönböztetése tehát felesleges, azokat össze lehet vonni. Jelöljük az új, összevont állapotot *ac*-vel.

Az összevont állapottábla

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

A kódolt állapottábla

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

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

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

A vezérlési tábla

A tervezési folyamatnak ezen a pontján általában döntenünk kell, milyen flip-flopokkal valósítjuk meg a hálózatot. A D-MS tárolókkal való megvalósítás kevesebb tervezői munkával jár, hiszen minden flip-flopnak csak egyetlenegy bemenete van, ugyanakkor a kiadódó kombinációs hálózat általában bonyolultabb, mint J-K MS tárolók alkalmazásakor. Ez az előny illetve hátrány persze erősen feladatfüggő. Ezért javasolható mindkét flip-flop típus kipróbálása, és egy jól megválasztott költség-függvény szerinti összehasonlítás. A vezérlési táblák megalkotásához szükségünk van a flip-flopok vezérlési tábláira, nevezetesen arra, hogy adott kimeneti változáshoz milyen szinteket kell kapcsolnunk a bemenetekre. A 2.34. ábra mutatja a D-MS tároló vezérlési tábláját a már ismert összetett igazságtáblával együtt, abból levezetve. A vastag határvonalakkal feltüntetett táblázat baloldala mutatja a flip-flop kimenetének lehetséges változásait, a jobboldali oszlop pedig azt, hogy az adott változás végbemeneteléhez milyen logikai értéket kell a D bemenetre kapcsolni. Nyilván ez igen egyszerű, hiszen mindig a megkívánt új értékkel azonos értéket kell a D-re kapcsolnunk.

Kicsit bonyolultabb a J-K MS tároló vezérlési táblájának megszerkesztése. A 2.35. ábra mutatja ezt, ugyancsak az összetett igazságtáblából levezetve. Láthatjuk, hogy minden lehetséges változáshoz van egy olyan bemenet a kettő közül, amelynek szintje lehet 0 és lehet 1 is, azaz közömbös.

Ezek a don't care bejegyzések teszik népszerűvé a J-K flip-flopot, hiszen a következő állapot kombinációt generáló kombinációs hálózat egyszerűsítéséhez ezek jótékonyan járulnak hozzá.

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

2.34. ábra. A D-MS tároló vezérlési táblája

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

2.35. ábra. A J-K MS tároló vezérlési táblája

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

A 2.36. ábra mutatja a felsorolt táblázatokat, feladatunk megoldásának soron következő lépéseként. A baloldali összevont szimbolikus állapottábla alapján két állapothoz (*ac, b*) kell állapotkódot választani. Ez egyetlen tárolóval lehetséges. Mindegy, melyik állapothoz rendeljük a Q = 0, és melyik-

hez a Q = 1 értéket. Ezután illesztjük a kódolt állapottáblához a J-K tárolóval való megvalósításhoz szükséges vezérlési táblát. A kódolt állapottáblából kiolvassuk a Q előírt megváltozását, majd a tároló vezérlési táblája alapján a J és K oszlopokba beírjuk az ehhez szükséges értékeket.

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

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

2.36. ábra. A feladat megoldásának három fontos táblázata

Az f_y és f_z hálózatok tervezése K-táblák segítségével

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

A feladat megoldására szolgáló hálózat K táblái és lefedésük

A kódolt állapottáblából már látható, hogy esetünkben igen egyszerű, kétváltozós K-táblákkal megadhatjuk a két kombinációs hálózat logikai függvényeit (2.37. ábra).

A K-táblák kiértékelése igen egyszerű eredményekhez vezet:

$$J = E, \quad K = 1, \quad Z = Q\,E$$

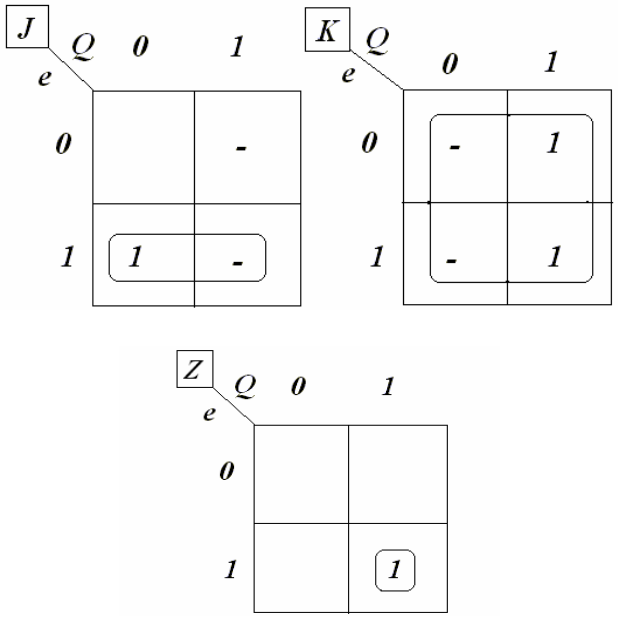
A realizációról általában

A realizáció során – amellett, hogy a flip-flop szimbólumokkal és az ismert kapu-szimbólumokkal felrajzoljuk a hálózat struktúráját – gondoskodnunk kell a kezdeti (bekapcsolás utáni) állapot beállításáról is. Ha PRESET és CLEAR bemenetekkel is rendelkező flip-flopokat választunk, akkor egyszerű dolgunk van: Ha a kezdeti állapotban egy adott tároló kimenete 0, a

CLEAR bemenetre adunk egy kezdeti állapotba állító impulzust, és a PRESET bemenetet állandó 0-ba állítjuk, ha pedig egy adott tárolót kezdeti állapotban 1-be kell állítani, akkor a PRESET kimenetre adunk impulzust, és a CLEAR bemenetre konstans 0-t.

Mindaddig, amíg a kezdeti beállítás más módszereit meg nem ismerjük, csak PRESET és CLEAR bemenetekkel is rendelkező flip-flopokat használunk.

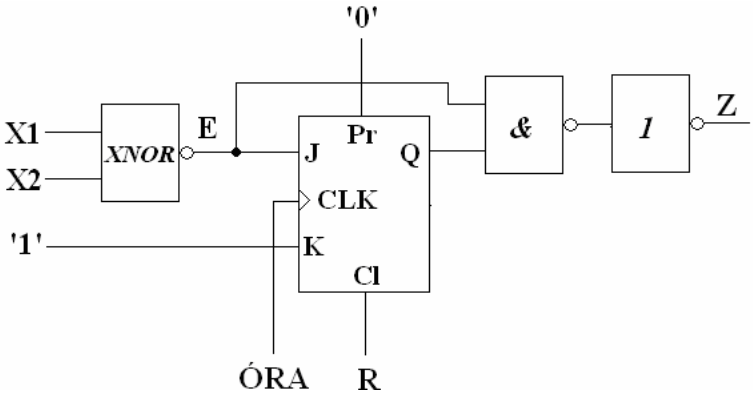
A kezdeti állapot beállítását eredményező speciális bemenő jelet R (RESET) szimbólummal jelöljük.



2.37. ábra. A feladat K táblái

A feladat megoldásának realizációja

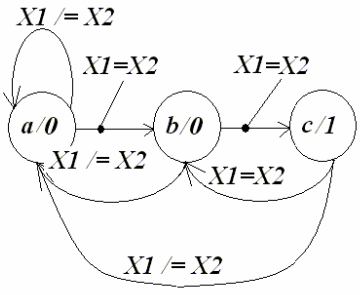
Meglepő lehet számunkra, hogy a 2.38. ábrán bemutatott realizáció tárolójának kimenete nincs visszacsatolva a következő állapotot generáló hálózat bemenetére. Nincs visszacsatolás, annak ellenére, hogy a szinkron sorrendi hálózat általános modelljében ezt hangsúlyoztuk. Könyveljük el, hogy egy adott speciális funkció megvalósítása vezethet arra, hogy egyik vagy másik szekunder változó értékétől nem függenek egyik vagy másik flip-flop állapot-változásai. Realizált hálózatunk sajátossága, hogy a flip-flop aktuális kimenetétől nem függ annak a következő állapota.



2.38. ábra. A feladat realizációja

2.4.2. Az első szinkron feladat megoldása Moore-típusú hálózattal

Ebben a pontban bemutatjuk a feladat Moore-típusú hálózattal történő realizációját, elsősorban a Mealy-típussal való realizációtól való eltérések hangsúlyozásával. Már az állapot-gráfon az állapotok jelölése mutatja a Moore típusnak azt a jellegzetességét, hogy a kimeneti kombinációk az csak az állapotok függvényei. Ugyancsak látható az eltérés a szimbolikus állapot-táblán is. (2.39. ábra) Az előzetes szimbolikus állapottábla alapján elvégezzük az állapot-összevonási lehetőségek vizsgálatát. A szabály hasonló: két állapot összevonható, ha a hozzájuk tartozó kimeneti kombinációk és a következő állapotok bemeneti kombinációnként azonosak. Könnyen megállapíthatjuk, hogy ilyen párok nincsenek, tehát a Moore-típusú hálózatot három állapottal, azaz két szekunder változóval kell megterveznünk.



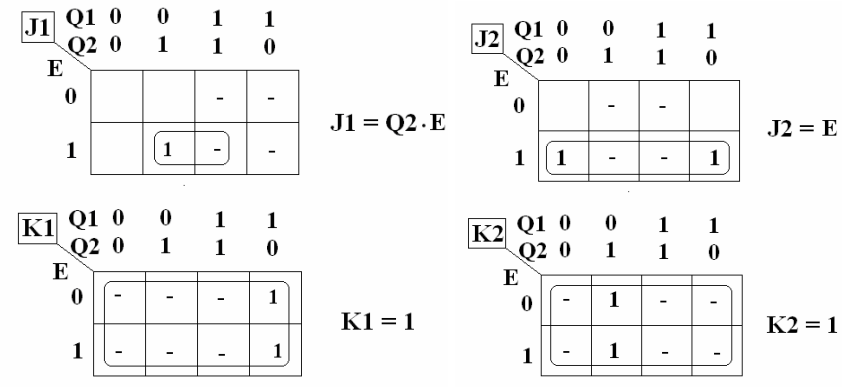
akt. áll./ /kim.	köv.állapot	
	X1/=X2	X1=X2
a/0	a	b
b/0	a	c
c/1	a	b

2.39. ábra. A feladat Moore-típusú realizációjának állapotgráfja és előzetes szimbolikus állapottáblája

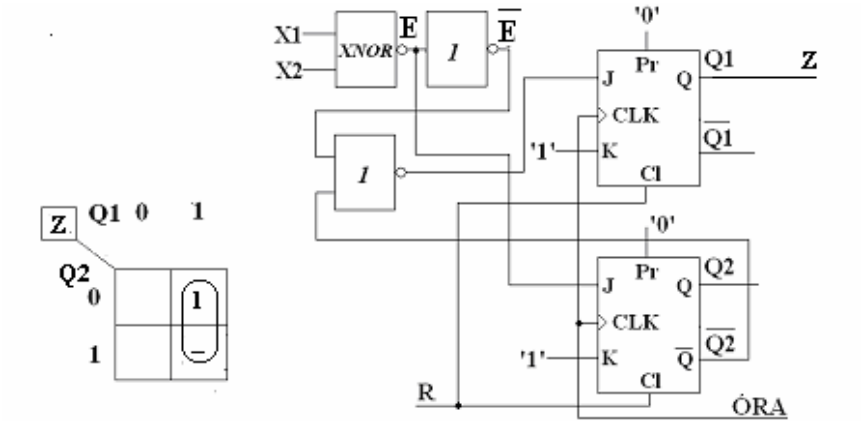
A 2.40. ábra együtt mutatja a szimbolikus és a kódolt állapottáblát, illetve a vezérlési táblát. Ennek kiemelendő sajátossága, hogy a két szekunder változó egyik kombinációja, nevezetesen az 11 itt kihasználatlan, tehát a belőle származó következő állapotokhoz, és azok vezérléseihez közömbös bejegyzéseket tehetünk. A K-táblákon történő függvénylefedések és a kapu szintű realizáció már rutinmunka. Az állapotkombinációk és a kimeneti kombinációk közötti egyértelmű függés szerint a Z kimenet csak a *c* állapotban, azaz az 10 állapotkombináció fennállásakor lesz magas szinten. Ezt K-tábla nélkül is könnyen realizáljuk. (2.41., 2.42. ábrák)

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

2.40. ábra. Állapottáblák és vezérlési tábla a feladat Moore-féle realizációjához



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



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

2.5. Aszinkron sorrendi hálózatok tervezési folyamata mintapéldákon bemutatva

2.5.1. Az első aszinkron hálózat tervezési mintafeladat

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

Időzítési diagram és előzetes szimbolikus állapottábla

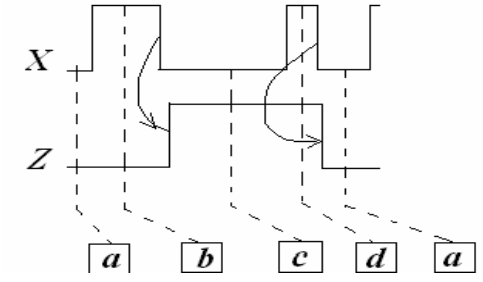
Ahogyan a szinkron hálózattervezés kezdeti lépéseként az állapot-gráf felvételét ajánlottuk, úgy ajánlható aszinkron hálózat tervezésének első lépéséül egy idődiagram felvétele. Az idődiagram felvételekor figyelembe kell venni, hogy az aszinkron hálózat minden új stabil állapotba való elindulása egy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

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

A feladat időzítési diagramja és előzetes szimbolikus állapottáblája

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

Ennek az idődiagramnak alapján szerkeszthető az előzetes szimbolikus állapottábla. Az állapottáblán körbefoglalással jelöljük a stabil állapotokat. Tudjuk, hogy aszinkron állapottáblán stabil következő állapot az, amely-



2.43. ábra. Az első aszinkron tervezési feladat idődiagramja

nek szimbóluma azonos az aktuális állapot szimbólumával. A működést az állapottáblán is követhetjük. A kezdeti a aktuális állapotban az X=0 bemenet az *a* állapotot stabilizálja. Az állapot mellett „/" jellel elválasztva látjuk a kezdeti kimeneti kombinációt.

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

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

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

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

Állapot-összevonás

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

Állapot-összevonás a feladat állapottábláján

A 2.44. ábrán látható állapottáblán nem találunk összevonható állapotokat.

Állapot-kódolás, a kódolt állapottábla felvétele

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

A feladat állapotainak kódolása és kódolt állapottáblája

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

$a \rightarrow 0\ 0$
$b \rightarrow 0\ 1$
$c \rightarrow 1\ 0$
$d \rightarrow 1\ 1$

Az ennek megfelelő kódolt állapottáblát a 2.45. ábrán láthatjuk.

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

2.45. ábra. Az első aszinkron feladat kódolt állapottáblája

Analízis a kritikus versenyhelyzetek felderítésére

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

Kritikus versenyhelyzetek a feladat kódolt állapottáblájának vizsgálatával

Tegyük fel, hogy az X = 1-hez tartozó 0 1 (*b*) állapotban vagyunk. Ha most X bemenetet 0-ra kapcsoljuk, akkor tranziens állapotként az 1 0 (*c*) állapot beállását várjuk, amely a bemenetre visszajutva stabilizálódik, és ezzel megtörténik az elvárt 0 1 → 1 0 átmenet. Ennek szemléltetését látjuk a 2.46. ábrán. Sajnos ha a hálózatot ezzel az állapotkóddal megvalósítjuk, hibás lehet a valóságos működés. A 2.47. ábrán szemléltetjük, mi lesz annak a következménye, ha a késleltetési idők különbözősége miatt egy másik tranziens állapot, a 0 0 áll be.

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

2.46. ábra. Egy ideális állapotátmenet szemléltetése a kódolt állapottáblán

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

2.47. ábra. Egy lehetséges valóságos állapotátmenet szemléltetése: kritikus a versenyhelyzet Y1 és Y2 szekunder változók között a 0 1 → 1 0 átmenetnél

Kritikus versenyhelyzetek kiküszöbölése átkódolással

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

A feladat állapotkódjának megváltoztatása a kritikus versenyhelyzetek kiküszöbölésére

A következő kódválasztás a 2.48. ábra tanúsága szerint megfelel e kritikus hazárdmentesség követelményének. Javasoljuk az olvasónak, analizálja valamennyi stabil állapotátmenet mentességét a kritikus versenyhelyzetektől.

$a \rightarrow 0\ 0$
$b \rightarrow 0\ 1$
$c \rightarrow 1\ 1$
$d \rightarrow 1\ 0$

bem. / akt. áll.	köv.áll/kim		k. akt. áll.	k.köv.áll/kim	
	X=0	X=1	$Y1''Y2''$	X=0 Y1Y2/Z	X=1 Y1Y2/Z
a	(a)/0	b /0	0 0	(0 0)/0	0 1 /0
b	c /1	(b)/0	0 1	1 1 /1	(0 1)/0
c	(c)/1	d /1	1 1	(1 1)/1	1 0 /1
d	a /0	(d)/1	1 0	0 0 /0	(1 0)/1

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

A realizáció K-táblái és lefedésük

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

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

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

Az első aszinkron feladat realizációja

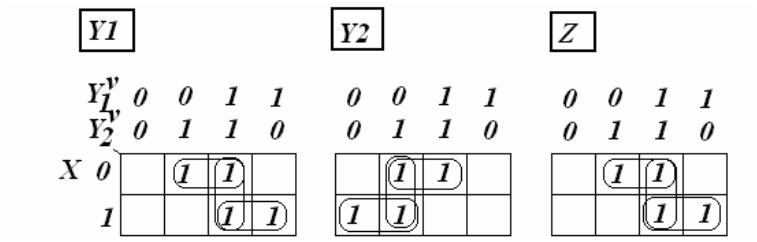
Az első aszinkron feladat K-táblái és lefedésük rutin-feladat, de vigyáznunk kell arra, hogy mindhárom függvény realizációját statikus hazárdoktól mentesen kell megoldani. Az f_y hálózat kimenetein jelentkező statikus hazárd ugyancsak a specifikációtól eltérő, hibás működéshez vezethet (2.49. ábra). A lefedés a következő eredményeket szolgáltatja:

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

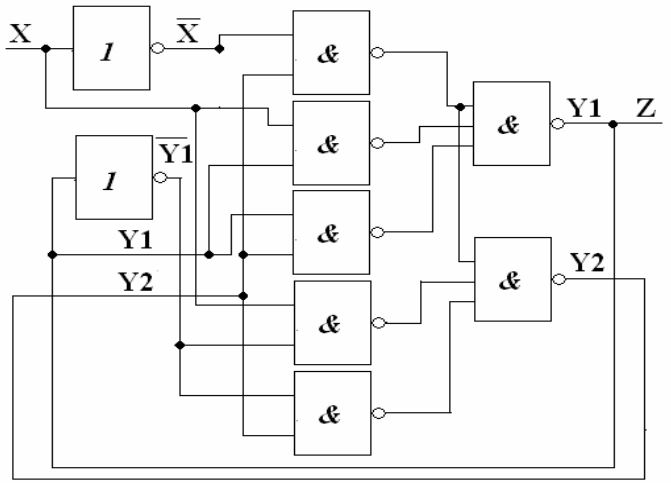
Értelmezzük a Z-re kapott érdekes eredményt. Z nem függ a bemenettől, csakis egyetlen szekunder-változótól. Mealy-típusú hálózat tervezésébe fogtunk, mégis, annak speciális eseteként, Moore-típusú hálózatot kaptunk. A realizációt a 2.50. ábra mutatja. Az ábra szerint hálózat azonban még nem használható. Ha bekapcsoljuk, azaz ráadjuk a tápfeszültséget, hiába adjuk rá az alapállapotnak megfelelő X = 1 kombinációt, nemcsak az *a,* a *c* állapot is beállhat. Márpedig bizonyosnak kell abban lennünk, hogy a bekapcsolás után, az *a* állapot áll be, sőt a hálózatot ebbe az alapállapotba bármikor be szeretnénk állítani.

Aszinkron hálózatok beállítása kezdeti állapotba

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



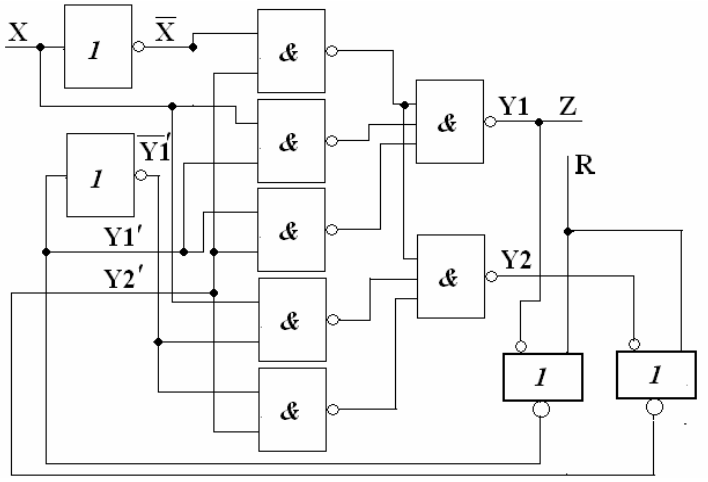
2.49. ábra. Az első aszinkron feladat K táblái



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

Az első aszinkron minta-feladat realizációjának kiegészítése kezdeti állapotba kényszerítő R-logikával

Az előző pont szerint eljárva, hasítsuk fel a visszacsatolást, és illesszünk be a visszacsatoló körbe két két-bemenetű logikát. Az egyik bemenetük közös, a kezdeti állapotba kényszerítő R (RESET) jel, a másik bemenetük a visszacsatolandó szekunder változókra kapcsolandó. A kimeneteket kapcsoljuk az f_y hálózat bemeneteire. Mindkét R-logika az R = 1 esetben 0-t ad tovább, ez pedig a kezdeti állapot kódja. Ha R = 0, a logikák kimenetére a megfelelő szekunder változó kerül, tehát él a visszacsatolás. A 2.51. ábra mutatja az R-logikákkal kiegészített realizációt.



2.51. ábra. A realizáció R (RESET) kezdeti állapotba állító logikákkal

2.5.2. A második aszinkron hálózat tervezési mintafeladat

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

Az előzetes, szimbolikus állapottábla

Ennél a feladatnál mellőzhetjük az idődiagramot, az előzetes szimbolikus állapottábla (2.52. ábra) anélkül is megszerkeszthető. A feladat specifikációjából egyértelmű, hogy a Z magas értékei csak az 1 1 oszlopban lesznek, de csak azoknál az állapotoknál. Amelyek az 1 0-ban levő állapotokat követik. Az kezdeti állapotot (*a*) a 0 0 bemeneti kombinációhoz vesszük fel. Ebből az állapotban az 1 1 bemeneti kombinációra való áttérés nem megengedett, hiszen ebben az esetben két bemeneti változó is értéket váltana, ezt pedig megtiltjuk. Így az 1 1-heztartozó következő állapot és a kimenet értéke közömbös. A 0 1 és az 1 0 azonban megengedettek, mindkettőre új állapotot vettünk fel, és a hozzájuk tartozó kimenetek természetesen 0-k. A *b* és *c* állapot sorainak kitöltésekor ismét célszerű először a tiltott bemeneti kombinációkhoz tartozó bejegyzésekről gondoskodni. Ha a *b* állapotban 0 0 jelentkezik, az *a* állapotba mehetünk vissza. Ha 1 1 jön, akkor egy új, a *d* állapotot vesszük fel, és Z-t továbbra is alacsonyan tartjuk. A *c* állapotból 0 0-ra a-ba mehetünk a Z = 0-val, 1 1-re viszont az új *e* állapot beálltához a Z = 1 tartozik, hiszen teljesült a speciális ÉS feltétel, X2 az

X1-et követően emelkedett magasra. Most a két legutóbb felvett állapotról, *d*-ről és *e*-ről kell gondoskodni. Egyikből sem kapcsolhatunk 0 0-ra, de a 0 1-re a *b* / 0, 1 0-ra a *c* / 0 jó választás, hiszen az előbbi esetben X1 lefut, így a speciális ÉS feltétel teljesülésének lehetősége távolabbra kerül, a második esetben viszont fennmarad.

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

2.52. ábra. A második aszinkron mintafeladat előzetes szimbolikus állapottáblája

Az összevont, szimbolikus állapottábla

A közömbös bejegyzések miatt finomítjuk a két állapot összevonhatóságáról kimondott kritériumunkat. Két állapot összevonható, ha bemenőkombinációnként megegyeznek a specifikált kimeneti kombinációk, és a specifikált következő állapotok. Ennek alapján a következő párok vonhatók össze: *ab* , *ad* , *bd* , *ce*. Az összevont állapotok tehát: (*abd*), (*ce*). Jelöljük az (*abd*) összevont állapotot *s1*-vel, a (*ce*)-t *s2*-vel. Az összevont állapottábla (2.53. ábra) sorainak kitöltésénél az eredeti tábla közömbös bejegyzései okoznak gondot. Könnyen belátjuk azonban, hogy mindig azt az összevont állapotot kell beírnunk, amelyhez tartozó állapot az adott oszlopban,

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

2.53. ábra. A második aszinkron mintafeladat összevont állapottáblája

az összevont állapot eredeti állapotainak sorában szerepelt. Például: *s1* sorában az 1 1 oszlopban *s1*-et írunk, mivel az *s1*-hez tartozó állapotok specifikált következő állapota a *d*, ami az *s1*-ben szerepel.

A második aszinkron feladat kódolt állapottáblája (2-54.ábra)

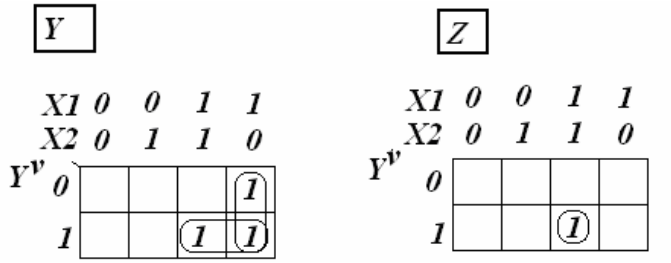
Mivel két állapot van, egyetlen szekunder változó elég a kódoláshoz. Nyilvánvaló, hogy egy szekunder változó esetén a kritikus versenyhelyzet problémája fel sem merül.

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

2.54. ábra. A második aszinkron minta-feladat kódolt állapottáblája

A második aszinkron feladat függvényeinek lefedése

A 2.55. ábrán mutatjuk be a második aszinkron feladat megoldásának K-tábláit.



2.55. ábra. K-táblák a második aszinkron feladathoz

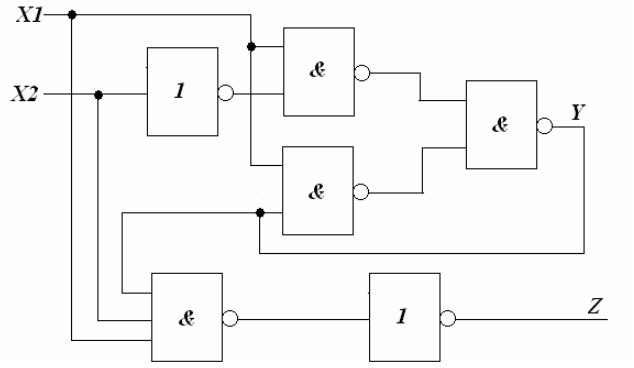
A lefedés eredménye:

$$Y = X_1\overline{X_2} + X_1Y^v$$

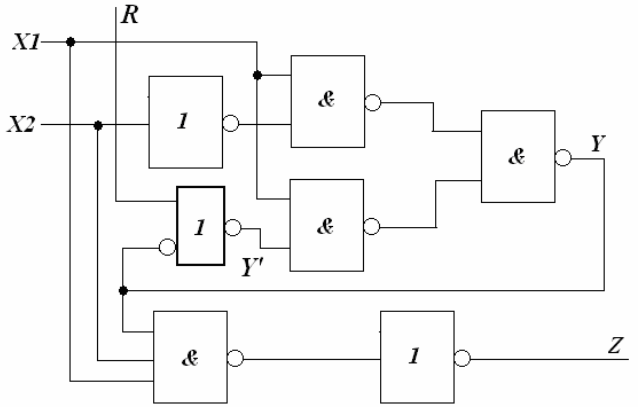
$$Z = X_1X_2Y^v$$

A sorrendi ÉS kapu realizációja R-logika nélkül

A 2.56. ábra mutatja a kezdeti beállítás nélküli realizációt, míg a 2.57. ábrán az R-logikával való kiegészítést is láthatjuk.



2.56. ábra. A sorrendi ÉS kapu NÉS-NÉS realizációja



2.57. ábra. A sorrendi ÉS áramkör R-logikával kiegészítve

A sorrendi ÉS kapu realizációja S-R tárolóval

A 2.58. ábra mutatja az S-R tároló vezérlési tábláját. Ennek segítségével kapjuk meg a sorrendi ÉS áramkör S-R tárolós realizációjának vezérlési tábláját, ami a 2.59. ábrán szerepel. A 2.60. ábra a K-táblákat, a 2.61. ábra a realizációt mutatja. Ez kezdeti állapot beállítás nélküli realizáció.

S	R	Y^v	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	–
1	1	1	–

$Y^v \rightarrow Y$		S	R
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0

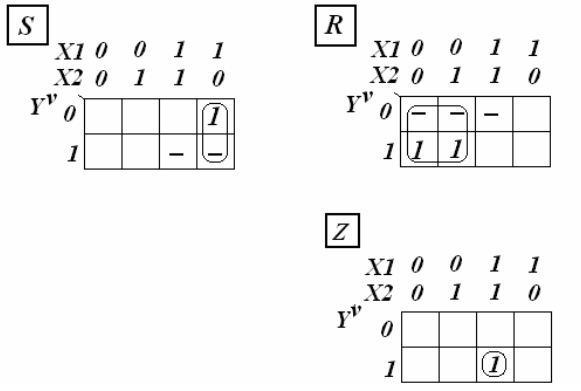
2.58. ábra. Az S-R tároló vezérlési táblája

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

vezérlési tábla:

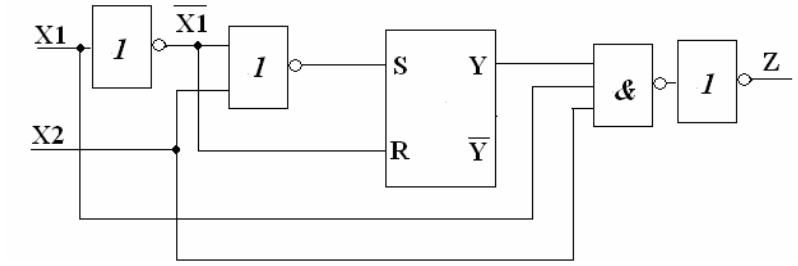
X1 X2 / bem. k.akt.áll.	0 0		0 1		1 0		1 1	
	S	R	S	R	S	R	S	R
0	0	–	0	–	1	0	0	–
1	0	1	0	1	–	0	–	0

2.59. ábra. A sorrendi ÉS kódolt állapottáblája és vezérlési táblája



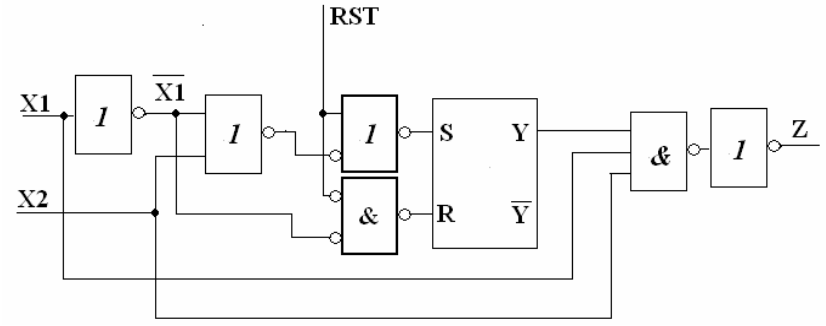
2.60. ábra. A sorrendi és S-R tárolós megvalósításának K táblái

$$S = X_1 \overline{X_2}$$
$$R = \overline{X_1}$$
$$Z = X_1 X_2 Y^v$$



2.61. ábra. A sorrendi ÉS S-R tárolós megvalósítása kezdeti állapotbeállítás nélkül

A kezdeti állapot beállítását könnyű megoldani az S és az R bemeneteken. Ha az RST (RESET) jelet felemeljük, akkor az S bemenetre 0-t, az R bemenetre 1-et kényszerítünk, így állítjuk be az Y = 0 kezdeti állapotot (2.62. ábra).



2.62. ábra. A sorrendi „ÉS” S-R tárolós megvalósítása kezdeti állapotbeállítással

2.5.3. Lényeges hazárdok aszinkron hálózatokban

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

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

2.6. Sorrendi hálózatok tervezési folyamatainak összegzése

Ebben a pontban összefoglaljuk a sorrendi hálózatok tervezésének fő lépéseit. Mivel a tervezési folyamatok a szinkron és az aszinkron hálózatok esetében lényeges különbségeket mutatnak, ezért külön alpontokban mutatjuk be a folyamatok lépéseit.

2.6.1. Szinkron sorrendi hálózatok tervezésének lépései

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

2.6.2. Aszinkron sorrendi hálózatok tervezésének lépései

1. lépés: A szimbolikus előzetes állapottábla felvétele
2. lépés: Állapot-összevonás
3. lépés: Állapotkódolás, a kritikus versenyhelyzetekre figyelemmel.
4. lépés: Kódolt állapottábla felvétele
5. lépés: Döntés arról, hogy közvetlenül visszacsatolt kombinációs hálózat, vagy S-R tárolós legyen a hálózat
6. lépés: Szekunder változók lefedése, vagy a vezérlési tábla felvétele és a tárolók vezérlő jeleinek lefedése a statikus hazárdok kiküszöbölésével
7. lépés: A hálózat elemzése a lényeges hazárdok kiküszöbölésére
8. lépés: Késleltetések beiktatása
9. lépés: Kimeneti hálózat logikai függvényének lefedése

2.7. A kezdeti állapot beállítása

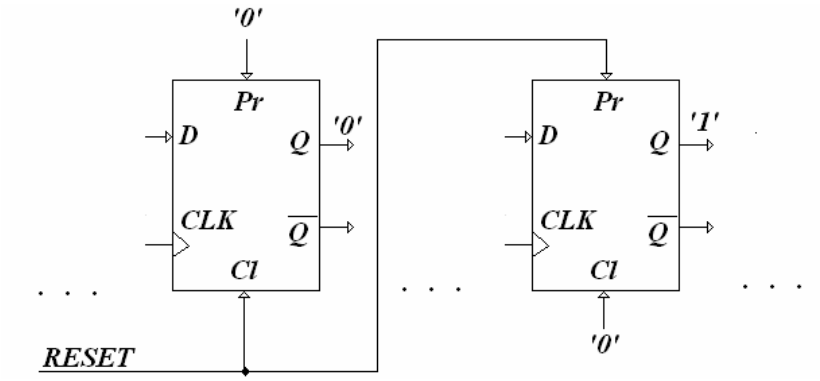
A kezdeti állapotkódok kezdeti beállítását tulajdonképpen nemcsak utólag, hanem a tervezéssel párhuzamosan is elvégezhetnénk, ha a bemenő jelek listájára felvennénk az R jelet, és már a szimbolikus összevont állapottáblán is figyelembe vennénk a lehetséges bemeneti kombinációk között. Ennek hátránya,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

2.7.1. Szinkron sorrendi hálózatok kezdeti állapotának beállítása

Mivel a szinkron hálózatokat mindig MESTER-SZOLGA tárolókkal valósítjuk meg, a kezdeti állapot beállítása a flip-flopok kezdeti állapotainak beállítását jelenti. A következő alpontokban ennek módszereit tekintjük át.

Beállítás a PRESET (Pr) és a CLEAR (Cl) bemenetek kihasználásával

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



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

Ebben a pontban ezt a módszert általánosítjuk tetszőleges kezdeti állapot kódra. A 2.63. ábra egy elképzelt flip-flop sorban (állapotregiszterben) két különböző kezdeti értékű flip-flopot mutat. Nyilvánvaló, hogy a 0 kezdeti értékűek CLEAR, míg az 1 kezdeti értékűek PRESET bemenetét aktivizáljuk az R jellel.

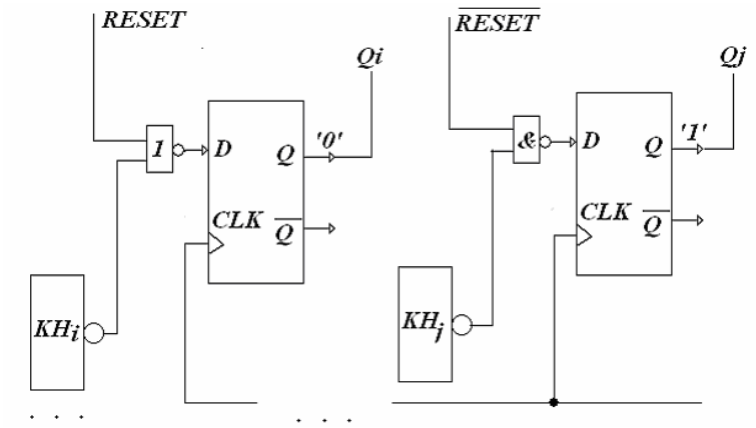
Beállítás az f_y hálózat kiegészítésével

A D flip-flopok esetén alkalmazandó kiegészítő hálózatok igazságtáblái láthatók. A D'_i olyan flip-flop bemenete, amelyet 0-ba, a D'_j olyan flip-flop bemenet, amelyet 1-be kell állítani kezdetben. Az igazságtáblák láthatók a 2.64. ábrán. A 2.65. ábra mutatja a segédhálózatok beillesztését.

D_i	RESET	D_i'
0	0	0
0	1	0
1	0	1
1	1	0

D_j	RESET	D_j'
0	0	0
0	1	1
1	0	1
1	1	1

2.64. ábra. A D-flip-flopoknál alkalmazott kiegészítő hálózatok igazságtáblái



2.65. ábra. A kiegészítő hálózatok beillesztése D flip-flopok kezdeti állapotainak beállítására. A már kiszámított KH_i és KH_j hálózatok végére invertereket kell tennünk, ezeket a körök szimbolizálják

Az ábra alapján kiszámíthatók a kiegészítő hálózatok realizációinak kifejezései:

$$D_i' = D_i\ \overline{RESET} = \overline{\overline{D_i} + RESET}$$

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

A realizációk láthatók a 2.65. ábrán.

A 2.66. ábra a J-K tárolók esetén alkalmazandó kiegészítő hálózatok tervezését, míg a 2.67. ábra a realizált hálózatokat mutatja.

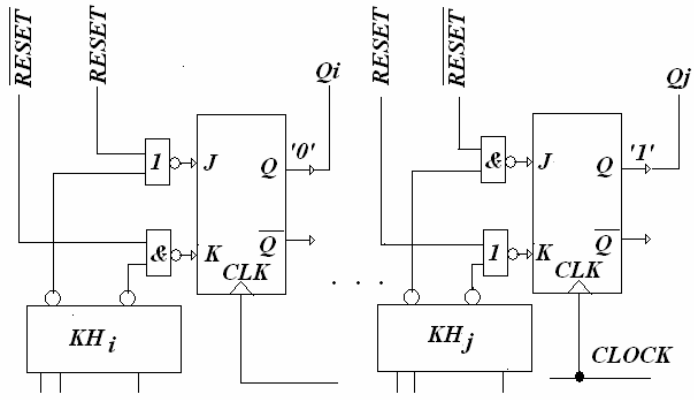
J_i	RESET	J_i'
0	0	0
0	1	0
1	0	1
1	1	0

K_i	RESET	K_i'
0	0	0
0	1	1
1	0	1
1	1	1

J_j	RESET	J_j'
0	0	0
0	1	1
1	0	1
1	1	1

K_j	RESET	K_j'
0	0	0
0	1	0
1	0	1
1	1	0

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



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

2.7.2. Aszinkron sorrendi hálózatok kezdeti állapotának beállítása

Közvetlenül visszacsatolt kombinációs hálózattal megvalósított aszinkron hálózat kezdeti állapotának beállítása

Szekunder változónként felhasítjuk a visszacsatolást, és beillesztünk egy-egy olyan kiegészítő hálózatot, amely RESET = 0 esetén a kiszámolt szekunder változó értéket, RESET = 1 esetén a kívánt kezdő értéket (0-t vagy 1-et) helyezi a megfelelő kimenetre. Jelöljük ezt a kimenetet Y'-val. A kezdeti állapotot beállító segéd hálózatok igazságtáblái és logikai megoldásai láthatók 2.68. ábrán.

Y_i	RESET	Y_i'
0	0	0
0	1	0
1	0	1
1	1	0

Y_j	RESET	Y_j'
0	0	0
0	1	1
1	0	1
1	1	1

2.68. ábra. Közvetlenül visszacsatolt aszinkron hálózat kezdeti állapotának beállítása

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

Az olvasóra bízzuk, hogy az eddigi tapasztalatok alapján szerkesszen általános sémát a kezdeti állapot-beállítás kapuszintű realizációjára.

S-R tárolókkal visszacsatolt aszinkron hálózatok kezdeti állapotának beállítása

A 2.69. ábrán az igazságtáblák, a 2.70. ábrán a realizáció látható. Az i indexű S és R bemenetű tárolót 0-ba, a j indexű S és R bemenetű tárolót 1-be kell állítani.

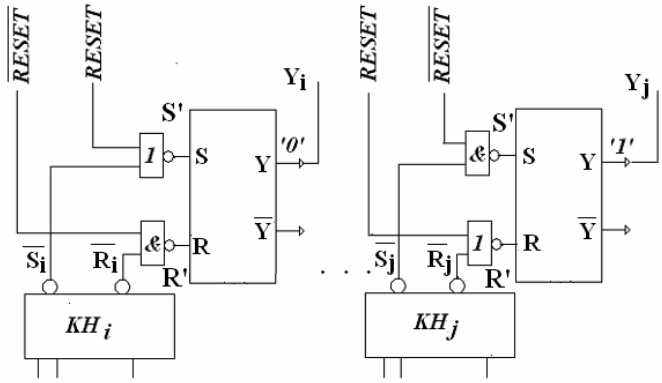
S_i	RESET	S_i'
0	0	0
0	1	0
1	0	1
1	1	0

R_i	RESET	R_i'
0	0	0
0	1	1
1	0	1
1	1	1

S_j	RESET	S_j'
0	0	0
0	1	1
1	0	1
1	1	1

R_j	RESET	R_j'
0	0	0
0	1	0
1	0	1
1	1	0

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



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

2.8. Állapot-összevonási módszerek

2.8.1. Állapot-összevonás teljesen specifikált előzetes szimbolikus állapottáblán

Az összevonhatóság feltétele

Általános megfogalmazást adunk arra, mikor tekinthetünk két szimbolikus állapotot összevonhatónak egy teljesen specifikált előzetes, szimbolikus állapottáblán. Teljesen specifikált az állapot-tábla (TSH) akkor, ha nem tartalmaz egyetlen „közömbös” bejegyzést sem. Két állapot a TSH állapot-

tábláján nem megkülönböztethető (NMK), ha a két állapotból elindulva bármely bemeneti sorozatra ugyanazt a kimeneti sorozatot látjuk.

Ebből a magától értetődő definícióból kiindulva bizonyítható, hogy két állapot összevonható, ha a két állapotból bármely bemeneti kombinációra adott kimeneti kombinációk megegyeznek, és NMK állapotokra vezetnek.

A nem-megkülönböztethetőség, mint reláció

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

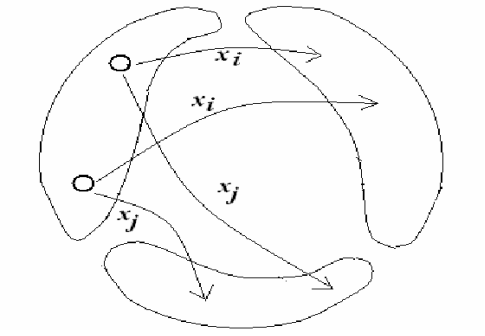
- reflexív
- szimmetrikus
- tranzitív

A reflexivitás jelentése az a trivialitás, hogy egy szimbolikus állapot saját-magától nem különböztethető meg, azaz a≡ a.

Szimmetrikusak azok a bináris relációk, amelyekre igaz, hogy amenynyiben a≡ b akkor bizonyosan fennáll a b≡ a reláció is.

Tranzitív relációk esetén igaz, hogy amennyiben: a≡ b és b≡ c, akkor teljesül az a≡ c is.

Magától értetődő, hogy a teljesen specifikált állapottáblákra definiált nem megkülönböztethetőség reflexív, szimmetrikus és tranzitív. Az ilyen relációkat ekvivalencia-típusú relációknak nevezzük. Szokás a TSH-n ezt a tulajdonságot állapot-ekvivalenciának is nevezni, azaz az NMK állapot-párok tagjait ekvivalenseknek mondjuk.



2.71. ábra. Elágazások ekvivalens állapotokból

A nem-NMK (MK), azaz a megkülönböztethető állapotpárok tagjait antivalens állapotoknak nevezzük. Az állapot-összevonás szempontjából fontos tétel, hogy egy adott halmaz elemein értelmezett ekvivalencia-típusú reláció a halmazt diszjunkt részhalmazokra bontja fel. Így az állapotekvivalencia az TSH állapothalmazát olyan, közös elemeket nem tartalmazó részhalmazokra bontja fel, amelyek az összevont állapothalmazt alkotják. Ezt szemléletesen mutatja a 2.71. ábra, amelyen a diszjunkt részhalmazok egyikében két tetszőleges állapotból láthatjuk az x_i és x_j bemeneti kombinációkra történő elágazásokat. A lényeg, hogy az ekvivalens állapotok bemenő-kombinációnként azonos részhalmazba ágaznak el.

Visszatérve az összevonhatóság általános megfogalmazására, az összevont állapotok alkotta új állapotokat megvalósító hálózat és az eredeti között nem ugyanolyan bemeneti sorozat alkalmazásával a kimeneti sorozatok között nem észlelhetünk különbséget. Úgy is fogalmazhatnánk, hogy az előzetes állapottáblával megfogalmazott hálózat és az összevont állapottáblával megfogalmazott hálózat egymással ekvivalens.

Az ekvivalencia kimutatására állapottábla alapján páronként kell vizsgálnunk az ekvivalencia vagy antivalencia tényét. Az alkalmazott jelölések a következők:

- a ≡ b : a és b ekvivalensek
- a < > b : a és b állapotok antivalensek

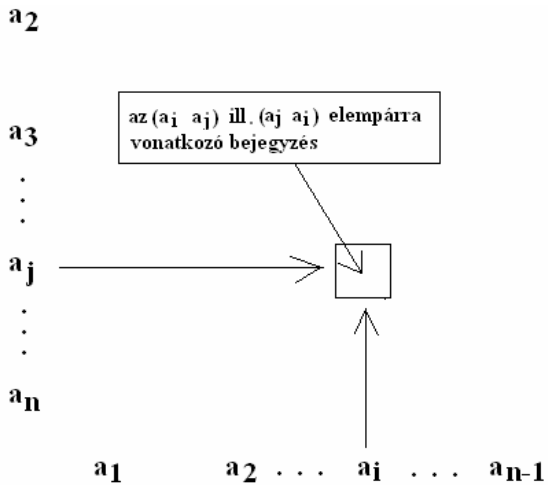
Az állapottábla analízise során sokszor nem lehet eldönteni azonnal az ekvivalencia vagy az antivalencia fennállását. Ezért, ha nem látjuk két állapotról azonnal, hogy antivalensek, akkor feljegyezzük azokat a feltételeket, amelyek fennállása esetén a két állapot ekvivalens.

A feltételes ekvivalenciát magával a feltétellel jelöljük. Például, ha a jelölés a következő felsorolás: (ab, cd …) akkor az a két állapot, amelyekre ez vonatkozik, feltételesen ekvivalensek, azaz csak akkor ekvivalensek, ha a ≡ b és c ≡ d.

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

Lépcsős tábla az összevonhatóság páronkénti vizsgálatára

Ha egy halmaz *n* elemből áll, akkor egy *n*×*n* méretű négyzetrács négyzeteibe bejegyezhetjük egy bináris reláció teljesülését, nem teljesülését, vagy a teljesülés feltételeit. Ha a reláció szimmetrikus, elég ehhez az átló mentén felezett négyzetrács tábla, amit jellegzetes alakjáról lépcsős táblának nevezünk. A lépcsős tábla formája egy *a1, a2, …. an* elemhalmaz esetén a 2.72. ábrán látható. A lépcsős táblás állapot-összevonás természetesen nem áll meg a páronként értelmezett ekvivalencia megállapításánál. Ha a tranzitivitást a megállapított állapot-párok között érvényesítjük, akkor kialakulnak a maximális ekvivalencia osztályok, azaz azok a diszjunkt állapothalmazok, amelyeknél nagyobbakat már nem lehet találni. Így valamennyi állapot bekerül egy ekvivalencia-osztályba, és nincs egyetlen olyan állapot sem, amely egynél több osztályba bekerülne.

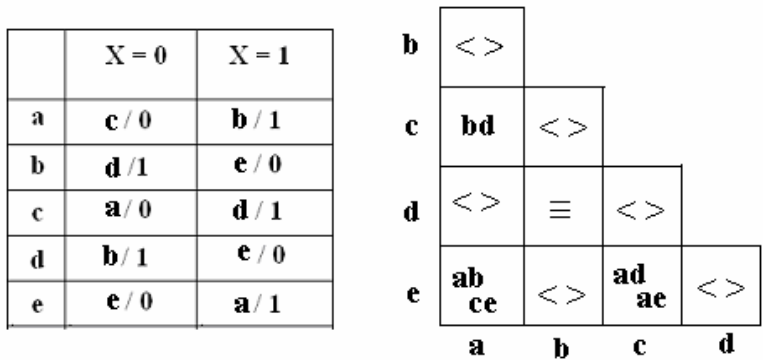


2.72. ábra. A lépcsős tábla struktúrája

Mintapélda megoldása lépcsős táblán

Tekintsük a 2.73. ábra szerinti előzetes szimbolikus állapottáblát. Ennek alapján készül el az első lépcsős tábla, amely az ábra jobb oldalán látható. Kitöltésekor a következő módszert alkalmazzuk:

Ha a kimenetek is és a következő állapotok is bemeneti kombinációnként azonosak, akkor a keresztezési cellába beírjuk az ekvivalencia szimbólumát (≡).



	X = 0	X = 1
a	c / 0	b / 1
b	d / 1	e / 0
c	a / 0	d / 1
d	b / 1	e / 0
e	e / 0	a / 1

b	<>			
c	bd	<>		
d	<>	≡	<>	
e	ab ce	<>	ad ae	<>
	a	b	c	d

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

Ha a kimeneti értékek legalább egy bemeneti kombinációra különbözők, akkor a keresztezési cellába beírjuk az antivalencia szimbólumát (<>).

Ha a kimeneti értékek bemeneti kombinációnként megegyeznek, de a következő állapotok legalább egy bemeneti kombinációnál eltérnek, akkor feltétel bejegyzés kerül a keresztezési cellába. Egynél több eltérés esetén a rész-feltételek és-kapcsolatba kerülnek.

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

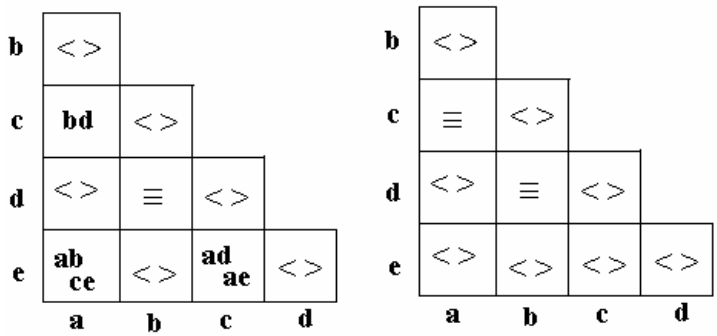
Ugyanakkor az (*a c*) pár vizsgálatakor a kimenetekkel nincs baj, de a következő állapotok (*a c*) és (*d b*) ugyan nem azonosak, de ekvivalensek még lehetnek! Sőt az (*a c*) ekvivalenciájához feltételként az (*a c*) feltételt beírni tautológia, tehát csak a (*b d*) párost írjuk be. A (*b d*) ekvivalenciája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

A második lépcsős táblát az elsőből kiindulva alkotjuk meg a következő szabályok alapján:

A 2.74. ábra bal oldala a kiindulási 1. lépcsős tábla, a jobb oldali az ebből előállítható 2. lépcsős tábla.

Minden egyes antivalencia bejegyzés következményeit érvényesítjük a táblán. Azaz, ha két állapot antivalens, és kettejük ekvivalenciája két másik állapot ekvivalencia-feltételeként szerepel valahol, oda antivalens bejegyzést kell tennünk.

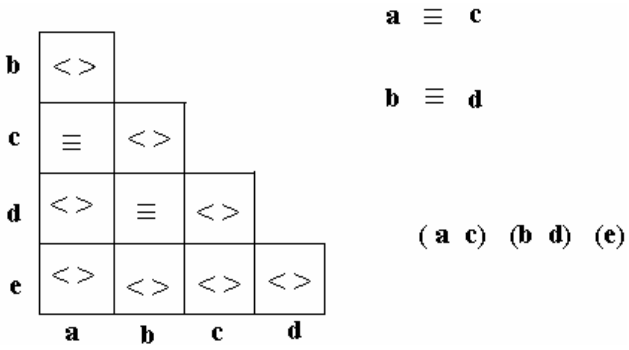
Ezután ezeket a második generációs antivalencia bejegyzéseket is érvényesíteni kell, és mindezt addig kell ismételni, amíg érvényesítetlen antivalencia bejegyzés van a táblán.



2.74. ábra. Az első és a második lépcsős tábla

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

Példánkban a 2. tábla már kizárólag ekvivalencia és antivalencia bejegyzéseket tartalmaz, így az ekvivalens párok és a tranzitivitás figyelembevételével a maximális ekvivalencia-osztályok könnyen kialakíthatók (2.75. ábra).



2.75. ábra. A csak ekvivalenciákat és antivalenciákat tartalmazó lépcsős tábla

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

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

A mintapélda összevont állapottáblájának szerkesztése

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

(a c) → *s1*
(b d) → *s2*
(e) → *s3*

	X = 0	X = 1
a	c / 0	b / 1
b	d /1	e / 0
c	a / 0	d / 1
d	b / 1	e / 0
e	e / 0	a / 1

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

2.76. ábra. Az összevont állapottábla szerkesztése az összevonás és az előzetes alapján

2.8.2. Állapot-összevonás nem teljesen specifikált előzetes szimbolikus állapottáblán

Az állapotkompatibilitás

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

Két szimbolikus állapot az NTSH állapottábláján csak akkor megkülönböztethető, (MK), ha létezik legalább egy olyan specifikált bemeneti sorozat, amely mindkét állapotra érvényes, és amelynek legalább egy elemére más kimeneti kombináció adódik.

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

Ha a kiválasztott két állapotra létezik olyan bemeneti kombináció, amelyre vagy a kimenetek, vagy a következő állapotok, vagy mindkettő

specifikálva vannak, a két állapot akkor nem-megkülönböztethető, ha a specifikált kimeneti kombinációk bemeneti kombinációnként megegyeznek, a specifikált következő állapotok pedig nem-megkülönböztethetők.

Az NTSH állapotpárjaira érvényes NMK bináris reláció a következő tulajdonságokat mutatja:

- reflexív
- szimmetrikus
- nem tranzitív

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

A kompatibilitás elégséges feltételei:

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

Az előző fejezetben megismert lépcsős táblának természetesen a kompatibilitás vizsgálatakor is fontos szerep jut. A következő jelöléseket fogjuk alkalmazni a táblázat celláiban:

$a \sim b$: a és b állapotok kompatibilisek
$a \,/\, \sim b$: a és b állapotok nem kompatibilisek

Feltételes kompatibilitás: *ab, cd* … az a két állapot, melyekre ez a bejegyzés vonatkozik, feltételesen kompatibilis, azaz csak akkor kompatibilis, ha $a \sim b$ és $c \sim d$…

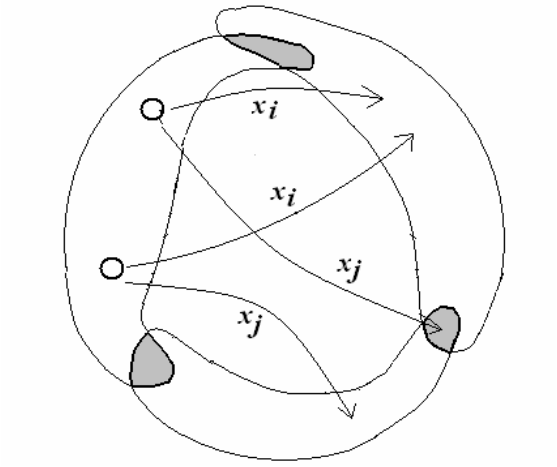
A kompatibilitási osztályok zárt halmaza

A fenti kompatibilitási reláció az állapothalmazt nem-diszjunkt osztályokra bontja, azaz lehetnek az osztályoknak közös elemeik is. Egy ilyen osztály valamennyi lehetséges állapotpárjára fennáll a kompatibilitás. Ezek az osztályok akkor maximálisak, ha további elemek egyetlen osztályba sem vonhatók be.

A maximális kompatibilitási osztályok halmazának két igen fontos tulajdonsága van. Az egyik a teljes lefedettség, azaz valamennyi állapotnak leg-

alább egy osztályban szerepelnie kell. A másik tulajdonság a zártság. Belátható, hogy a maximális kompatibilitási osztályok zárt halmazt alkotnak.

A kompatibilitási osztályok egy adott halmaza zárt, ha a halmazban szereplő bármelyik osztály tetszőleges két állapotából kiindulva minden olyan bemeneti kombinációra, amely mindkét állapotból specifikált következő állapotot ír elő, a következő állapotok is együtt szerepelnek a halmaznak legalább egy osztályában. A 2.77. ábra a zártságot szemlélteti. A bal felső osztályból két állapotot ragadtunk ki. Ezek közül az egyik utódállapota az *xj* bemeneti kombinációra két másik osztály közös részében van, de ezek közül az egyik osztály azonos azzal az osztállyal, amelyben a másik állapot *xj*-re adott utódja helyezkedik el.



2.77. ábra. A kompatibilitási halmazok zártságának szemléltetése

Kevesebb vagy kisebb állapotszámú osztályból álló zárt kompatibilitási osztályhalmaz keresése

Visszatérve a nem teljesen specifikált előzetes állapottáblára kimondott általános összevonhatósági kritériumra, megállapítható, hogy a maximális kompatibilitási osztályok megtalálása után a nem teljes specifikáció lehetőséget teremt arra, hogy az állapotok teljes lefedése és a zártság megőrzése mellett egyszerűbb kompatibilitási halmazrendszert válasszunk. Ez azt jelenti, hogy úgy döntünk a közömbös bejegyzésekről, hogy döntésünk vagy kevesebb kompatibilitási osztályból álló, vagy az egyes osztályokban kevesebb állapotból álló osztály-halmazt eredményezzen.

Ennek érdekében először megvizsgáljuk, van-e olyan kompatibilitási osztály, amelynek valamennyi állapota szerepel valamely más osztályban is. Ha így van, megkísérelhetjük elhagyni ezt az osztályt. Ez akkor lehetséges, ha az osztály elhagyása után is zárt marad a kompatibilitási osztályok halmaza. Ha a zártság nem tartható fenn, akkor visszatesszük az elhagyni kívánt osztályt, és a többszörösen szereplő állapotok egyes osztályokból való elhagyásával próbálkozunk. Ha találunk a teljes lefedettség és a zártság fenntartásával elhagyható állapotokat, akkor egyszerűbb összevont állapottáblát kapunk. Láthatjuk, hogy több megoldás is kínálkozhat, ezek közül kell választanunk a megvalósítandó összevont állapottáblát.

Úgy is fogalmazhatunk, hogy az összevont állapottábla szerinti hálózat realizálja az előzetes állapottábla szerinti hálózatot abban az értelemben, hogy a specifikált bemeneti sorozatokra adott kimeneti sorozatok nem különböznek egymástól.

Az összevont állapottábla szerkesztése

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

Példa NTSH állapottáblázaton történő állapot-összevonásra

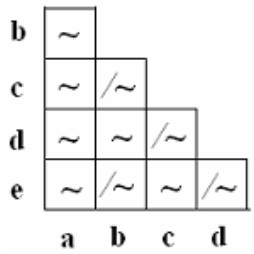
A 2.78. ábra alapján a 2.79. ábra lépcsős táblájából kapott maximális kompatibilitási osztályokból elindítva az egyszerűsítésre irányuló vizsgálatokat, azonnal belátható, hogy amennyiben bármelyik osztályt elhagyjuk, állapotok maradnak lefedetlenül, tehát marad a közös állapotok elhagyásával való kísérletezés. Két zárt osztályhalmazt kaphatunk így, az (*a, b, d*), (*c, e*), és a (*a, c, e*), (*b, d*) osztályhalmazokat. Az első osztályhalmaz zártságáról az állapottábla alapján meggyőződhetünk, és beláthatjuk, hogy az (*a, b, d*) minden eleme bemeneti kombinációnként ugyanabba az osztályba képződik le, illetve ez a (*c, e*) osztály elemeire is igaz. Hasonlóan bizonyítható a második osztályhalmaz zártsága is. Ebből az következik, hogy a példának kétféle állapot-összevonása is jó megoldáshoz vezet.

Mindezek alapján két összevont állapottáblát szerkeszthetünk, (2.80. ábra) és nekiláthatunk a kódolt állapottábla megszerkesztéséhez és a realizációhoz.

Példánk egyik megoldásában egyébként felismerhetjük a már megvalósított speciális sorrendi ÉS áramkörünket. Ezúttal egy másik, a korábban bemutatottól eltérő állapot-összevonási lehetőséget is találtunk.

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

2.78. ábra. NTSH állapottábla, az állapot-összevonás bemutatására



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

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

2.79. ábra. A maximális kompatibilitási osztályok megkeresése lépcsős tábla segítségével

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

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

2.80. ábra. Sorrendi ÉS két lehetséges állapot-összevonással

2.8.3. Összefoglalás az állapot-összevonási módszerekről

a. Állapot-összevonás teljesen specifikált előzetes állapottáblából:
   a.1. Az ekvivalens és antivalens állapotpárok megkeresése lépcsős tábla segítségével
   a.2. A maximális ekvivalencia-osztályok meghatározása
   a.3. A maximális ekvivalencia-osztályoknak megfelelő állapotokkal az összevont állapottábla elkészítése.

b. Állapot-összevonás nem teljesen specifikált előzetes állapottáblából:
   b.1. Valamennyi kompatibilis és inkompatibilis állapotpár megkeresése a lépcsős tábla segítségével
   b.2. A lépcsős tábla alapján a maximális kompatibilitási osztályok megkeresése
   b.3. A kompatibilitási osztályok legkedvezőbb zárt halmazainak megkeresése
   b.4. A legkedvezőbb zárt halmaz osztályainak egy-egy állapotot rendelve az összevont állapottábla szerkesztése

2.9. Állapotkódolási módszerek

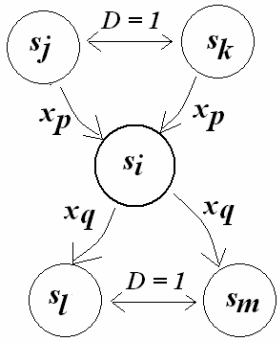
Az állapot-kódolási módszerek bemutatásakor élesen szét kell választani a szinkron és aszinkron eseteket. Más a cél az egyiknél, más a másiknál. Szinkron hálózatnál nincs versenyhelyzet veszély, így az állapotkódolás arra irányul, hogy a legegyszerűbb struktúrát alakítsuk ki. Aszinkron hálózatok esetében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

2.9.1. Szinkron hálózatok állapot-kódolási módszerei

A szomszédos állapotkódok választásának elvei

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

Egyszerűbb lesz a hálózat, ha egy adott állapot következő állapotainak kódjai bemenő-kombinációnként szomszédosak, csak egy szekunder változóban különböznek egymástól, azaz a kódok közötti távolság egységnyi (D=1). Ugyancsak előnyös, ha azok az állapotok, amelyek valamely adott állapotnak az elődjei, bemenő-kombinációnként szomszédos kódúak.



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

2.81. ábra. Szomszédos állapotkódok és egy előzetes állapottábla

A 2.81. ábra bemutatja a szomszédos kódolást kifejező állapotgráfon a leírt előnyös kódolási helyzeteket. Az ábra jobb oldalán egy előzetes állapottábla látható, amelyen megkísérlünk szomszédos állapotkódokat találni.

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

A példa szerinti állapottáblát a 2.82. ábrán látható formába dolgozzuk át, azaz elkészítjük az állapotok utódjait és az állapotok elődeit bemutató táblázatokat. A 2.83. ábra mutatja az állapotkódok optimális elhelyezését.

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

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

2.82. ábra. Utódok és elődök a 2.81. ábra állapottáblája alapján

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

Közös utódja van X = 1-re:
b,d
c,e

Közös elődje van X = 0-ra:
c,d

Közös elődje van X = 1-re

Szomszédos kódok:

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

Q1 Q2 / Q3	0 0	0 1	1 1	1 0
0	e	b		
1	c	d	a	

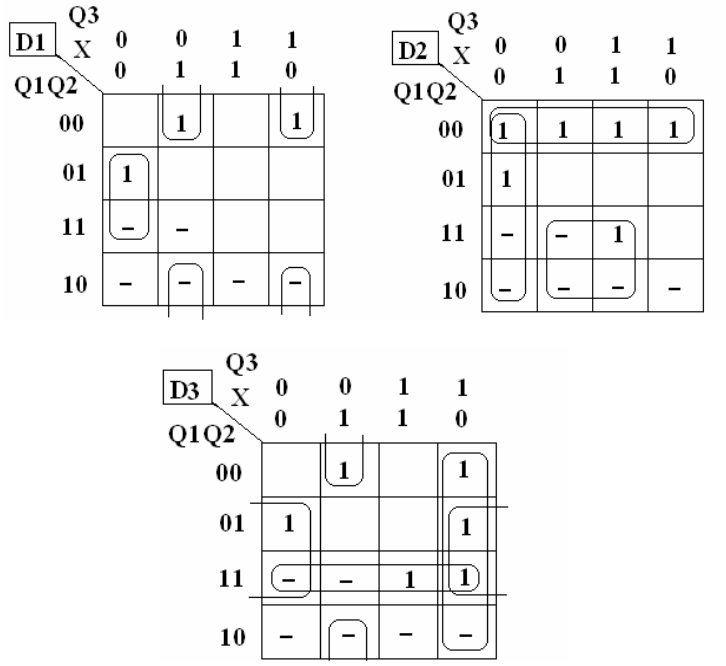
2.83. ábra. Az állapotok elhelyezése K-táblán,
az előnyös szomszédosságok legnagyobb arányú biztosítására

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

áll.	$Q_1Q_2Q_3$	X = 0	X = 1
		$D_1\ D_2\ D_3$	$D_1\ D_2\ D_3$
a	1 1 1	0 0 1	0 1 1
b	0 1 0	1 1 1	0 0 0
c	0 0 1	1 1 1	0 1 0
d	0 1 1	0 0 1	0 0 0
e	0 0 0	0 1 0	1 1 1
	1 0 0	– – –	– – –
	1 1 0	– – –	– – –
	1 0 1	– – –	– – –

2.84. ábra. Az állapotkód és a kódolt állapottábla

A 2.84. ábra a kódolt állapottáblát, a 2.85. ábra pedig a realizáció K-tábláit mutatja.



2.85. ábra. A realizáció K-táblái

Önfüggő szekunder-változó csoport keresése: egy bevezető példa

Ez a módszer az állapothalmazon értelmezett partíció-párok elméletén alapul. E helyen nem lehet célunk az elmélet teljes kifejtése, de egy egyszerű bevezető példán keresztül megvilágítjuk a lényeget, így segítve a konkrét módszer elsajátítását.

Tekintsük a 2.86. ábrát az ott található kódolt állapot-átmeneti táblázatokkal. A táblák által megadott sorrendi hálózatok közös sajátossága, hogy az X=0 vezérlés hatására állapotuk nem változik, az X=1 hatására azonban a-ból *b*, *b*-ből *c*, *c*-ből *d*, végül *d*-ből ismét *a* lesz. Ugyanakkor a az egyik kódolási változatban a *c* állapothoz a 11, a *d*-hez az 10 kódot választottuk, míg a másik kódolási változatban fordítva.

			X=0		X=1	
	Q_1^n	Q_2^n	Q_1^{n+1}	Q_2^{n+1}	Q_1^{n+1}	Q_2^{n+1}
a	0	0	0	0	0	1
b	0	1	0	1	1	1
c	1	1	1	1	1	0
d	1	0	1	0	0	0

			X=0		X=1	
	Q_1^n	Q_2^n	Q_1^{n+1}	Q_2^{n+1}	Q_1^{n+1}	Q_2^{n+1}
a	0	0	0	0	0	1
b	0	1	0	1	1	0
c	1	0	1	0	1	1
d	1	1	1	1	0	0

2.86. ábra. A példafeladat kódolási változatai

A két kódolási változathoz tartozó tároló-vezérlési kifejezések a következők kifejezésekkel adjuk meg.
Az első kódolási változat esetén:

$$D_1 = Q_1^n \overline{X} + Q_2^n X$$
$$D_2 = Q_2^n \overline{X} + \overline{Q_1^n} X$$

A második kódolási változat esetén:

$$D_1 = Q_1^n \overline{X} + Q_1^n \overline{Q_2^n} + \overline{Q_1^n} Q_2^n X$$
$$D_2 = Q_2^n \overline{X} + \overline{Q_2^n} X$$

Most alkossunk néhány partíciót (diszjunkt részhalmazra bontást) az állapothalmazon. Tekintsük most az első kódolási változatot. Az első partíció ($\mathbf{\Pi}_1$) azokat az állapotokat sorolja egy osztályba, amelyeket a baloldali flip-

flop egyformán kódol, azaz egy osztályba kerül az *a* állapot a *b*-vel, és a *c* pedig *d*-vel.

$$\Pi_1 = (a\ \ b),(c\ \ d)$$

A második, a $\mathbf{\Pi_2}$ partíció egy osztályba sorolja azokat az állapotokat, amelyeket a jobboldali flip-flop egyformán kódol, (*a d*),(*b c*).

$$\Pi_2 = (a\ \ d),(b\ \ c)$$

Ez a két partíció az első kódolási változat két flip-flopját abban az értelemben írja le, hogy osztályaik a flip-flopok állapotait azonosítják, azaz azok pontosan annyi osztályból állnak, ahány állapota a flip-flopoknak van, azaz kettőből.

Alkossuk meg ezeket a partíciókat a másik kódolási változatra is!

$$\Pi_2^K = (a\ \ d),\ (b\ \ c) = \Pi_2$$

$$\Pi_2 = (a\ \ c),(b\ \ d)$$

Figyeljünk fel arra, hogy mindkét kódolási változatnál a két partíciónak egyik osztályában sincs két olyan állapot, amely a másikban is egy osztályban szerepelne. Ezt úgy is kifejezhetjük, hogy a két partíció metszete a legfinomabb partíció, ($\mathbf{\Pi_0}$) azaz az a partíció, amely egy-állapotos osztályokból áll.

$$\Pi_1 \cap \Pi_2 = \Pi_0$$

Most foglalkozzunk mindkét kódolási változatra olyan partíciók felírásával amelyek a flip-flopok környezetének állapotait azonosítják, azaz ezek a partíciók azokat az állapotokat sorolják egy osztályba, amelyeket a flip-flopok környezete kódol azonos módon. Az első kódolási változatra fenn áll, hogy mindkét flip-flop környezetéhez a legfinomabb partíció tartozik, hiszen a D bemenetekre mindkét tároló állapotváltozója rákapcsolódik.

$$\Pi_1^K = (a),\ (b),\ (c),\ (d) = \Pi_0$$

$$\Pi_2^K = (a),\ (b),\ (c),\ (d) = \Pi_0$$

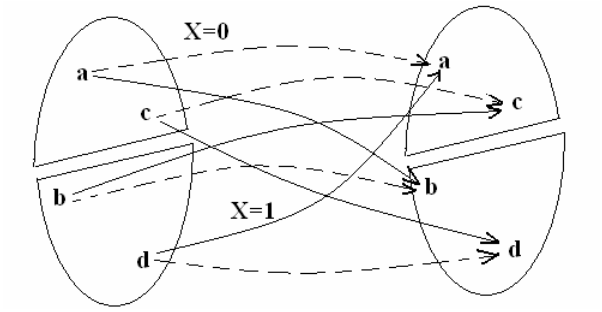
A második kódolási változat esetében érdekes dolgot veszünk észre: Mivel a jobboldali flip-flop D_2 bemeneti hálózatára a Q_1 nem kapcsolódik, a jobboldali flip-flop környezete ugyanazokat az állapotokat különbözteti meg, amelyeket maga a flip-flop is megkülönböztet. Mivel ebben a kódo-

lási változatban a jobboldali flip-flop partíciója és környezetének partíciója azonos, azt helyettesítési tulajdonságú partíciónak nevezzük.

$$\Pi_1^K = (a),\ (b),\ (c),\ (d) = \Pi_0$$

$$\Pi_2^K = (a\ c),\ (b\ d) = \Pi_2$$

A környezet és a hozzá tartozó flip-flop, azaz komponens partíciók között még egy fontos tulajdonságra kell felhívni a figyelmet. Ez pedig az, hogy amennyiben két állapot a környezeti partíció ugyanazon osztályában van, akkor bemeneti kombinációnként a következő állapotok a komponens-partíció ugyanazon osztályában szerepelnek. Ez a tulajdonság ttriviális módon áll fenn azokban az esetekben, amikor a környezeti partíció a $\mathbf{\Pi_0}$. Mivel a második változat jobboldali komponensének partíciója megegyezik saját környezetének partíciójával, a fenti szabályt ugyanazon partícióval szemléltetjük a 2.87. ábrán.



2.87. ábra. A partíciópár szemléltetése a második kódolás változat D2-Q2 tárolójához rendelt partíciópárral, amelynek tagjai megegyeznek

A környezeti partíciók a komponens partíciókkal partíció-párokat alkotnak. A második kódolási variánsnál tehát a $\mathbf{\Pi_2}$ önmagával alkot partíciópárt.

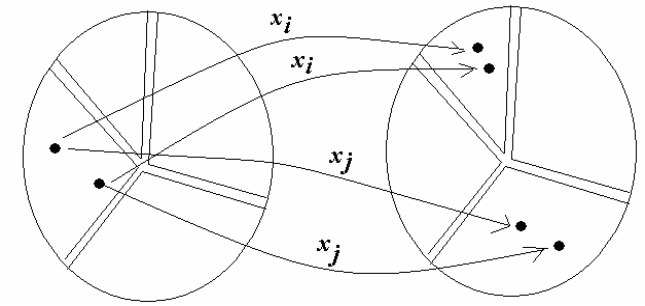
Önfüggő szekunder változó-csoport keresése: a módszer általánosítása

A példában bemutatott partíciópárok tulajdonságai, és azok összefüggése az állapotkódolással általánosíthatók. Az általánosítás eredményeképpen a következő állítások képezik a módszer elvi alapját:

Ha egy szinkron sorrendi hálózat állapotváltozóit csoportokra bontjuk, akkor a csoportokhoz komponenseket rendelhetünk. Minden egyes komponenshez két partíciót tartozik. Az egyik partíció azon állapotokat sorolja

egy osztályba, amelyeket a komponens egyformán kódol. A másik partíció azokat az állapotokat sorolja egy osztályba, amelyeket a komponensre kapcsolódó környezet egyformán kódol. A környezeti partíció és a komponens partíció úgynevezett partíció párt alkot.

Mivel a komponens a következő állapotának kialakításakor egy adott bemeneti kombináció esetén nem tesz különbséget két, a környezete által azonosan kódolt állapot között, a környezeti partíció ugyanazon osztályába tartozó állapotok következő állapotai bemeneti kombinációnként a komponens partíciónak is ugyanazon blokkjában vannak. Ezért nevezzük ezt a partíció kettőst partíció-párnak. A komponens partíciók metszete a legfinomabb partíciót eredményezi. Ha egy komponens bemeneteire nem kapcsolódnak más komponensek állapotváltozói, akkor a komponens partíció önmagával alkot partíció-párt. Az ilyen partíciót helyettesítési tulajdonságú (HT) partíciónak nevezik. A HT partíció összefüggését az állapot-átmeneti táblával, most már általános esetben, a 2.88. ábra szemlélteti.



2.88. ábra. A HT partíció önmagával partíciópárt alkot

Az HT partíciós komponenshez tartozó hálózat egyszerűbb, hiszen a többi komponens állapotváltozói nem tartoznak a bemenetei közé.

Célszerű tehát olyan komponensekre bontani a hálózatot, amelyek közül minél több független a többitől, azaz a szekunder változók önfüggő csoportjait reprezentálják. A tervezési módszer lényege, hogy az előzetes, szimbolikus állapottábla alapján HT partíciókat keresünk. Ha találunk nem triviális (nem a legfinomabb és nem a legdurvább) HT partíciót, akkor azzal önfüggő hálózat-komponenst valósíthatunk meg.

Néhány egyszerű állapotkódolási lehetőség:

Ha az állapothalmazon egy nem triviális HT partíciót találunk, akkor azzal egyetlen önfüggő hálózatrészt különíthetünk el a többitől.

Ha két nem triviális HT partíciót találunk, akkor két önfüggő hálózatrész különíthető el a többitől.

Ha két olyan nem triviális HT partíciót találunk, amelyek metszete a legfinomabb partíció, akkor a hálózat megvalósítható két önfüggő csoporttal.

Önfüggő szekunder változó-csoport keresése: egy HT-partíciós állapotkódolási feladat

Tekintsük a 2.89. ábrán látható állapot-átmeneti táblázatot. Kódoljuk az állapotokat egy önfüggő csoport elkülönítésével.

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

2.89. ábra. Állapottábla HT partíciós állapotkódoláshoz

A HT partíciók keresését lépcsős táblán végezzük. Először kitöltjük a lépcsős tábla celláit, milyen párosítási feltételek következnek a cellához tartozó két állapot egy osztályban való megjelenéséből. Ezután feltételezzük két állapotról, hogy egy osztályban vannak. (jelölés: y, karikában) Ezután minden olyan cellát „y”- al jelölünk, amelyekhez tartozó állapotok összetartozását a kiindulási párosítás implikálja. A jelöléskor figyelembe kell vennünk a tranzitivitást is.

Ha már nincs újabb implikált pár, az üresen maradt cellákba X-et teszünk.

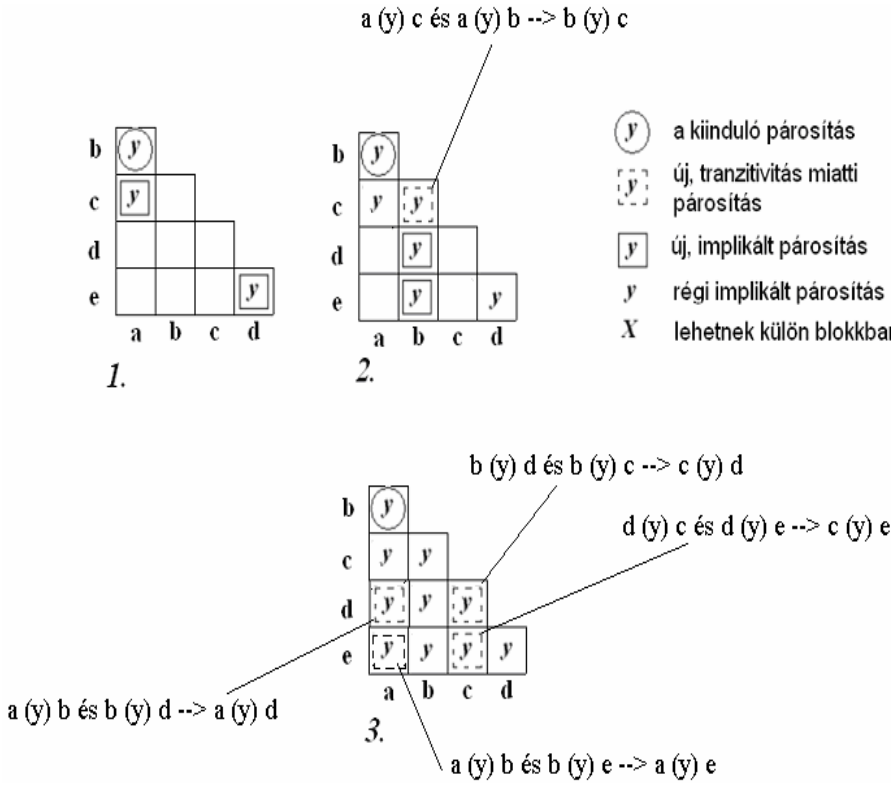
Ezután az ismert eljárással felvesszük a nem-diszjunkt osztályokat, majd a tranzitivitás alapján egyesítjük azokat, amelyeknek van közös elemük.

Ha az összevonás után a teljes állapothalmazt kapjuk, ez triviális (egyblokkos) HT partíció, és nem használható. Ilyenkor új kiindulási párt kell választani, és az eljárást erre az új párra kell megismételni. Ha valamennyi lehetséges nem triviális HT partíciót előállítottuk, válogatunk.

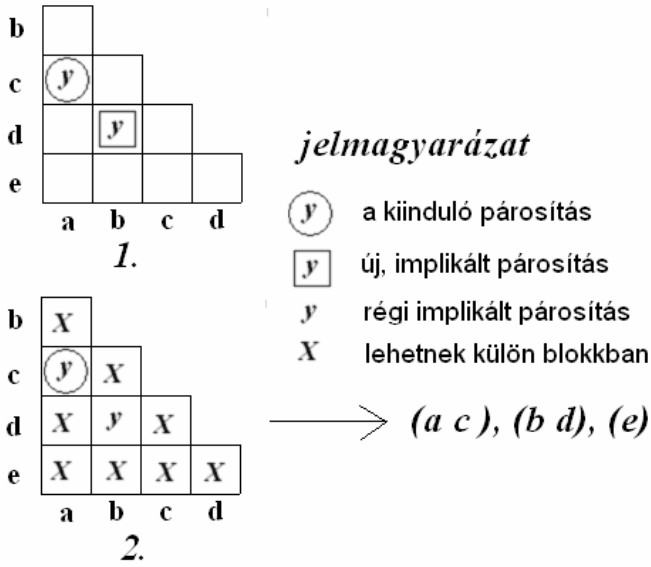
Világos, hogy egy adott HT partíciót kiválasztva annak blokkszáma (B) adja az önfüggő csoport állapotainak számát, a blokkokban előforduló maximális állapotszám (A) pedig a maradék szekunder változó-csoport által képviselt állapotszámot adja. Így a HT partícióhoz szükséges szekunder változók száma,

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

A 2.90. ábra lépcsős első tábláit először azzal a feltételezéssel töltjük ki, hogy az *a* és a *b* állapotok egy osztályban vannak. A következményeket látjuk az 1-es jelű táblán, majd tovább lépve a 2-es és 3-as táblákra, látjuk, hogy triviális partíciót kaptunk, hiszen az adódott, hogy minden állapot ugyanabba az osztályba tartozik. A 2.91. ábra szerinti feltételezés, miszerint az *a* és a *c* legyenek egy osztályban, nem triviális HT partícióra vezet: (*a c*), (*b d*), (*e*)



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



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

A 2.92. ábra szerinti állapotkód önfüggő csoportjában a szekunder változók száma 2, hiszen három osztályt kaptunk. Ezeken az osztályokon belüli kód finomításához már csak egyetlen változó kell, hiszen a legnépesebb osztály állapotszáma 2. Az állapotok kódjainak megválasztásakor egyetlen szabálya, hogy az egy osztályban szereplőket az önfüggő csoport állapotváltozói egyformán kódolják.

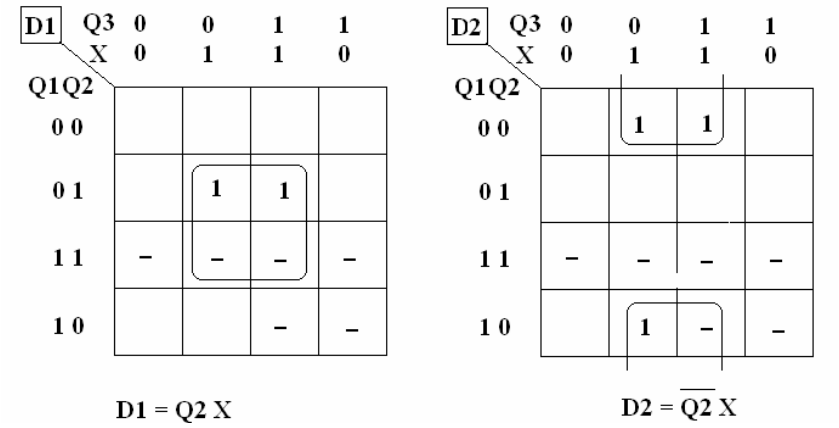
kód / áll.	Q1	Q2	Q3
a	0	0	0
c	0	0	1
b	0	1	0
d	0	1	1
e	1	0	0

2.92. ábra. Állapotkód, önfüggő változókkal

A 2.93. és a 2.94. ábrák a vezérlési táblával kiegészített kódolt állapottáblát és a realizáció K-tábláit mutatják. Látható, hogy sem a D1, sem a D2 nem függ Q3-tól. Az „önfüggés” tehát igazolódott.

áll.	X: $Q_1Q_2Q_3$	$D_1D_2D_3$: 0	1
a	0 0 0	0 0 1	0 1 1
b	0 1 0	0 0 0	1 0 0
c	0 0 1	0 0 0	0 1 0
d	0 1 1	0 0 1	1 0 0
e	1 0 0	0 1 0	0 0 0
	1 0 1	– – –	– – –
	1 1 0	– – –	– – –
	1 1 1	– – –	– – –

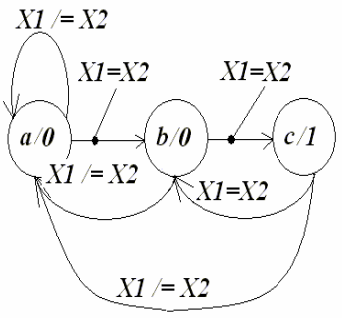
2.93. ábra. Kódolt állapot- és vezérlési tábla



2.94. ábra. Az önfüggés igazolása K-táblákkal

Szinkron hálózatok 1-es súlyú állapotkódolással

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



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

2.95. ábra. Egy korábbi feladat Moore-típusú realizációjának állapotgráfja és előzetes szimbolikus állapottáblája, 1-es súlyú, D-flip-flopos implementációra

A 2.95. ábrán látható állapotgráf és tábla alapján így megvalósítandó 1-es súlyú állapotkód három D-MS flip-flopot igényel. Az 1-es súlyú kódolás egy lehetséges változata a következő:

	Qa	Qb	Qc
a	1	0	0
b	0	1	0
c	0	0	1

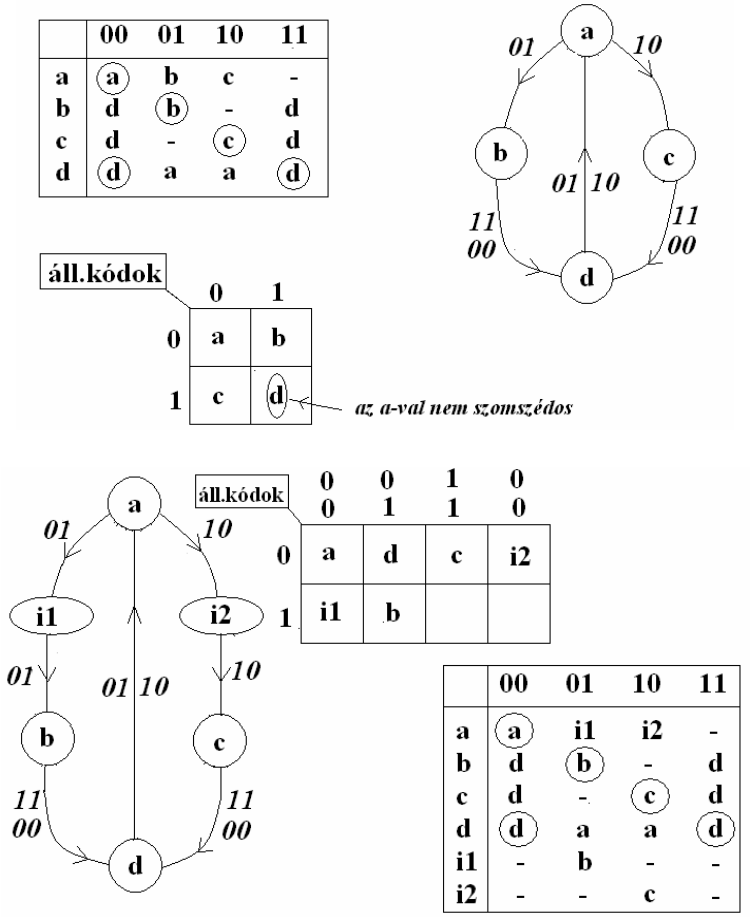
A D bemenetek vezérlésének megvalósítása ilyenkor rendkívül egyszerű, és az állapotgráf alapján elvégezhető.

Minden egyes D bemenetre akkor és csak akkor kell 1-et kapcsolni, amikor az általa reprezentált tároló kimenetnek 0-ról 1-re kell változnia, azaz amikor az adott flip-flop által reprezentált állapotnak be kell állnia. Ez pedig az állapotgráfból kiolvasható. További előny, hogy az alapállapotot beállító R jelet is azonnal „bedolgozhatjuk". Feltesszük, hogy E a két bemenet EXNOR függvénye.

$$D_a = R + \overline{R}(Q_a\overline{E} + Q_c\overline{E}) = R + \overline{E}(Q_a + Q_c)$$
$$D_b = \overline{R}\,E(Q_a + Q_c)$$
$$D_c = \overline{R}\,Q_b\,E$$

2.9.2. Aszinkron hálózatok állapotkódolása

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



2.96. ábra. Bővítés szomszédos kódú instabil állapotokkal

Instabil állapotok beillesztése a kritikus versenyhelyzetek kiküszöbölésére

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

Az eljárást egy igen egyszerű példán szemléltetjük (2.96 ábra).

Az előzetes állapottábla alapján felvett állapotgráf mutatja, hogy az *a* állapottal a *d* állapot kódját nem lehet szomszédossá tenni, ha ragaszkodunk a két szekunder változóhoz. Be kell szúrnunk két instabil állapotot (*i1* és *i2*), és ezzel egy új szekunder változót is! Az így kibővített állapothalmaz elemeinek kódjait már meg lehet úgy választani, hogy az egymás után következő tranziens állapotok kódjai szomszédosak legyenek. Hangsúlyozzuk, hogy sem az *i1*, sem az *i2* állapot soha sem stabilizálódik, de áthidaló tranziens szerepet töltenek be. Ez jól követhető az új állapotátmeneti táblázat segítségével is.

Tracey és Unger módszere a kritikus versenyhelyzetek kiküszöbölésére

Kritikus versenyhelyzet akkor áll elő, ha egy stabil állapotból kiindulva megváltoztatjuk a bemeneti kombinációt, és ennek hatására olyan átmeneti állapotkód áll elő, amelynek sorában és az adott bemeneti kombináció oszlopában ez az állapotkód szerepel.

A nem kívánt átmeneti állapotkódot HAZÁRD-KÓDNAK nevezzük.

A TRACEY-UNGER módszer lényege, hogy a normális (tervezett) állapotátmenethez tartozó kiinduló és cél állapotok kódjai legalább egy adott szekunder változóban mindketten különbözzenek a hazárd kódtól.
Ilyenkor ugyanis ez a szekunder változó az átmenet során állandó marad, és így soha sem áll elő a hazárd állapot kódja.

A kódolási eljárást az összevont szimbolikus állapottábla analízisével kezdjük. Egy listára felvesszük a stabil-stabil állapot-átmeneteket, és hozzájuk írjuk azoknak a „leselkedő" hazárd állapotoknak a nevét, amelyek az átmenet során felléphetnek, azaz azokat, amelyek a stabil célállapot oszlopában szerepelnek. Egy ilyen átmenet és a leselkedők listája egy kódolási szabályt definiál. Ez úgy hangzik, hogy a kiindulási és a cél állapot legalább egy szekunder változó értékében egyezzen, de ebben a változóban mindketten különbözzenek a „leselkedőktől".

Miután az összes állapotátmenetet ilyen módon listáztuk, akkor M számú kódolási szabályunk lesz.

Rendeljünk minden szabályhoz egy szekunder változót, és a szekunder változókkal oszlopokat, a kódolandó szimbolikus állapotokkal sorokat alkotva írjunk fel olyan kódot, amely a megállapított kódolási szabályoknak eleget tesz. A szabályok csak a kötelező különbségtételt írják elő, azt hogy melyik szekunder változó legyen 1 és melyik legyen 0, azt nem. Ügyeljünk azonban arra, hogy ahol az adott szabály nem kényszerít az adott állapotváltozóra értéket, oda közömbös bejegyzést tegyünk.

Belátható, hogy a közömbös bejegyzések tetszőleges konkretizálásával máris kritikus versenyhelyzettől mentes állapotkódot kaptunk, de túl sok szekunder változó bevezetése volt az ár. Ugyanakkor felismerhetjük, hogy a közömbös bejegyzések kihasználásával az oszlopokat összevonhatjuk. Ha az összevonás nehézségekkel jár, cseréljük fel szabadon az egyes oszlopokban az 1 és a 0 bejegyzéseket, és próbálkozzunk újra az összevonással. Tekintsük most a 2.97. ábra szerint előzetes állapottáblát.

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

2.97. ábra. Szimbolikus előzetes állapottábla a Tracey-Unger módszer bemutatására

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

ANALÍZIS:

Bemenő kombináció változásonként vizsgáljuk az állapot-átmeneteket, és a „leselkedő” hazárd állapotokat. Az analízis eredményét a 2.98. ábra *a* táblázata mutatja. Láthatjuk például, hogy a 00 → 01 megengedett bemeneti váltáshoz egyetlen stabil-stabil állapot-átmenet olvasható ki, nevezetesen az *a* állapotból a *c* állapotba való átmenet. Ezt egyedül a *d* állapot tranziens kódja veszélyezteti, tehát itt a leselkedő állapot a *d*. Az ábra *b* táblázatán az ennek megfelelő kódolási szabályt az Y1 szekunder változó képviseli, mégpedig azzal, hogy az oszlopában feltüntetett állapot-kódban mind az *a*, mind a *c* 0-val jelenik meg, a leselkedő *d* viszont 1-gyel! A *b* állapot Y1 pozícióbeli kód-része közömbös. Így szerkesztjük végig a táblázatot, ismétlésekbe nem bocsátkozunk, és a fordított irányú de azonos leselkedőt mutató átmeneteket is értelem-szerűen csak egyszer tüntetjük fel (lásd az áthúzott táblázatelemek).

A *c* táblázat az összevont oszlopokat mutatja. Az összevont állapotváltozók indexei mutatják, mely oszlopokat sikerült összevonni. Ezzel kialakult, hogy a versenyhelyzet-mentes kód végül is négy szekunder változóval biztosítható.

00→01		00→10		01→00		01→11		10→00		10→11		11→01		11→10
a→c	*d*	d→a	*b* *c*	c→d	*a*	d→b	*c*	b→d c→d	*a* *a*	a→b	*c*	b→d	*c*	

a.

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

b.

sz.v. áll.	Y145	Y36	Y2
a	0	0	0
b	1	0	1
c	0	1	1
d	1	1	0

c.

2.98. ábra. A Tracey-Unger módszer végrehajtása során keletkező táblázatok

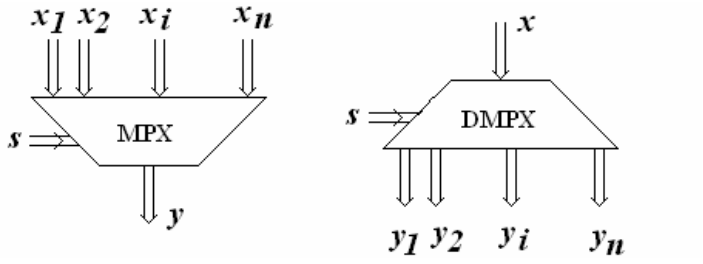
3. Összetett digitális egységek

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

- a multiplexerek, demultiplexerek, amelyek adatút szakaszokat jelölnek ki, a
- a regiszterek, amelyek adatokat tárolnak, és ezek elérését is biztosítják, és a
- funkciós egységek, amelyek adatok közötti műveleteket végeznek.

3.1. Multiplexerek, demultiplexerek

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

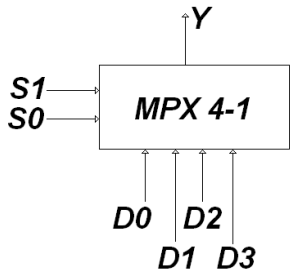


3.1. ábra. A multiplexerek és a demultiplexerek RT-szintű ábrákon használatos szimbólumai

3.1.1. Négybemenetű, egykimenetű multiplexer

A közepes integráltságú logikai áramkör családok kifejlesztői leginkább egy négy bit adatbemenettel és egy bit adatkimenettel, valamint ennek

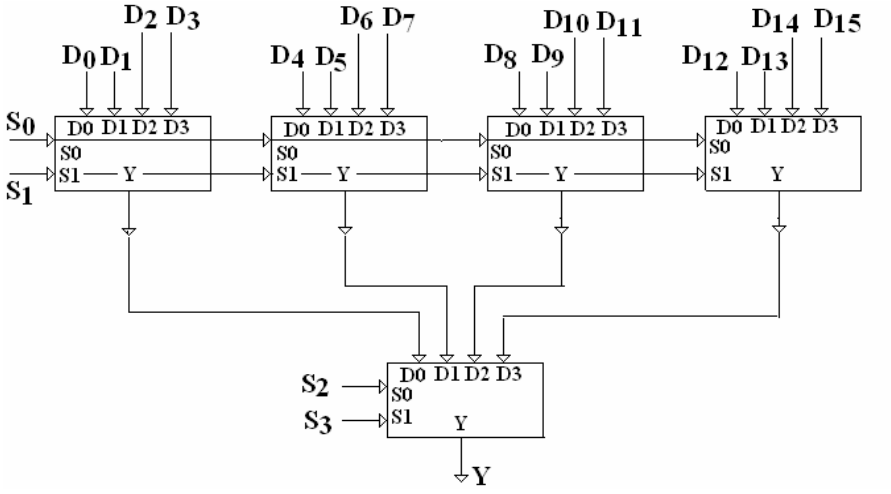
megfelelően két-bit címző bemenettel rendelkező alaptípust gyártottak és gyártanak, ez pedig többféle módon bővíthető. (3.2. ábra.)



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

3.1.2. Bővítés a bemenetek számának növelésére

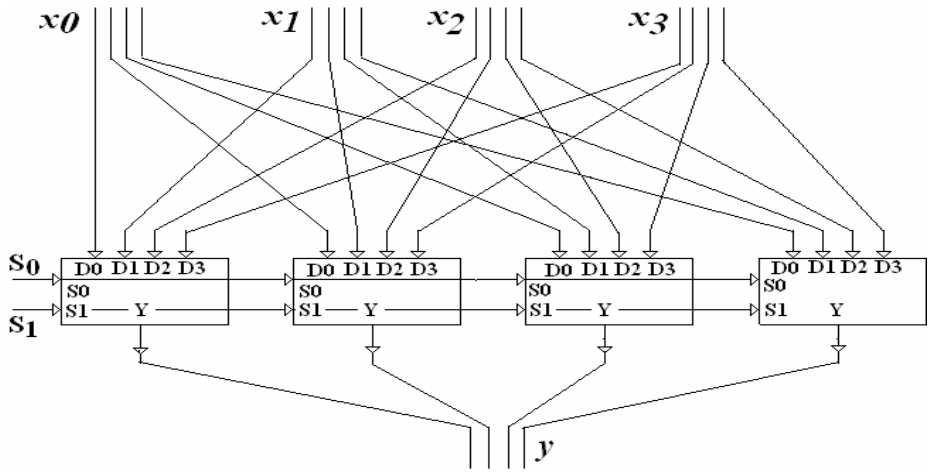
Ha a bemeneti adatunk négynél több bitből áll, úgy a 3.3. ábra szerinti bővítési sémát alkalmazzuk. Láthatjuk, itt 16-1 multiplexert alkottunk 4-1 egységekből. Látható, hogy nemcsak vízszintes irányban kellett bővíteni az egységet, hanem mélységében is. Felhívjuk a figyelmet az első sor összekötött címző-bemeneteire,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



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

3.1.3. Bővítés sínek közötti választás céljából

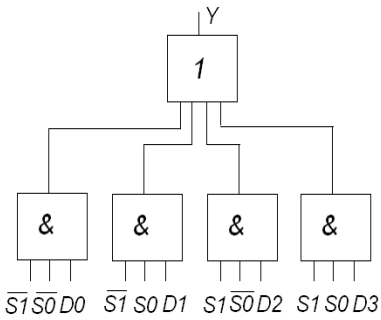
A négy, négybites sínből egyet kiválasztó multiplexer összeállítása vízszintes irányú bővítéssel oldható meg (3.4 ábra). Természetesen gyakran kell a kétféle bővítési módszer kombinációját alkalmaznunk.



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

3.1.4. A multiplexerek felépítése

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

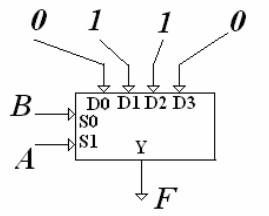


3.5. ábra. A 4-1 multiplexer egység kapuszintű struktúrája

3.1.5. A multiplexer, mint programozható logikai hálózat

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

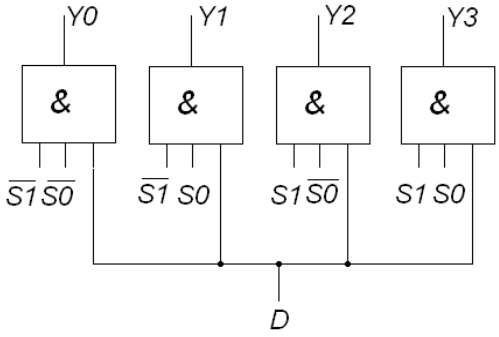
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



3.6. ábra. EXOR függvény 4-1 multiplexerből

3.1.6. Demultiplexerek

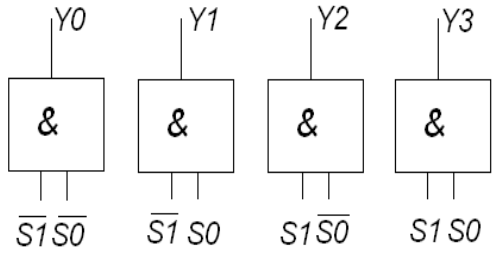
A demultiplexerek funkciója egy adat több lehetséges irány egyikébe történő továbbítása. Az 1-4 (egyet a négyből az egyikre) méretű demultiplexerhez négy, 3-bemenetű ÉS kapu szükséges. (3.7. ábra)



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

3.1.7. Dekóderek

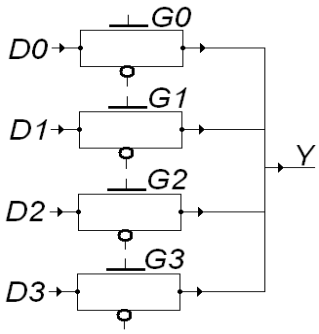
Ha egy 1-4 demultiplexer adatbemenetét állandó, logikai 1 szintre kapcsolunk, akkor ez egyenértékű azzal, hogy az ÉS kapuk bemenetei közül elhagyjuk az adatbemenetet. Az ilyen áramkör sajátossága, hogy a kiválasztó bemenetekre kapcsolt kombinációk csak egyetlen, a kiválasztó kódnak megfelelő kimeneten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



3.8. ábra. 2-bemenetű dekóder, ÉS kapukkal

3.1.8. Multiplexerek és demultiplexerek CMOS átvivő-kapukkal

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



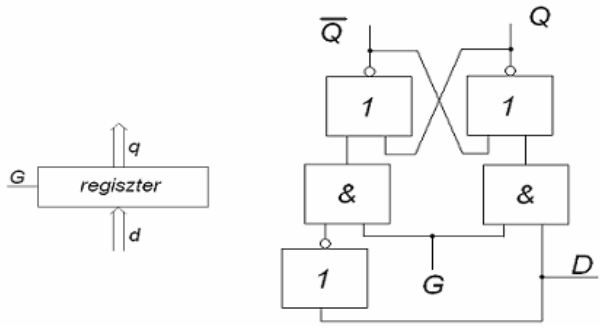
3.9. ábra. C-MOS átvivő-kapus multiplexer

A harmadik logikai állapotot is lehetővé tevő kimenetek szerepe a modern logikai hálózatokban és a mikroprocesszoros rendszerekben meghatározó jelentőségű. Ha ugyanis garantálni tudjuk, hogy egyetlen közös pontra a kimenetével kapcsolódó több logikai elem közül legfeljebb csak egy kimenete legyen nem harmadik állapotú, akkor olyasmit tehetünk, amely kétállapotú kimenettel rendelkező elemek esetén szigorúan tilos; kimeneteket kapcsolhatunk össze.

3.2. Regiszterek, párhuzamos elérésű tárolók

3.2.1. Szintvezérelt, statikus regiszter

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



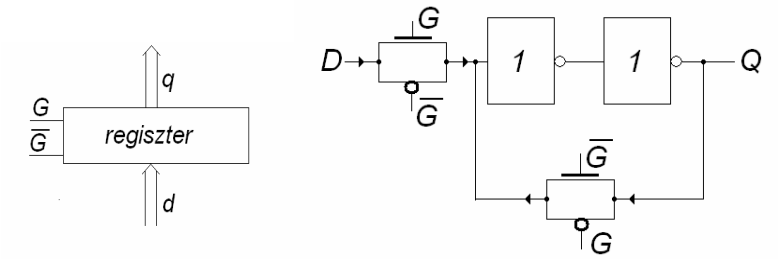
3.10. ábra. Szintvezérelt regiszter szimbóluma és egy bitjének belső felépítése

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

3.2.2. Szintvezérelt regiszter ponált és negált beírójelekkel

A CMOS technikában, különösen a VLSI áramkörökben előszeretettel alkalmazzák az átvivő-kapus tárolókból álló statikus regisztert. Az egy bitnyi tároló (latch) a két stabil állapotú (bistabil) invertergyűrű beírásának egy más módszerét alkalmazza, mint a NOR-bistabilok. A beírás (G=1) alatt a visszacsatolás meg van szakítva, hiszen a visszacsatoló átvivő-kapu a G=0-

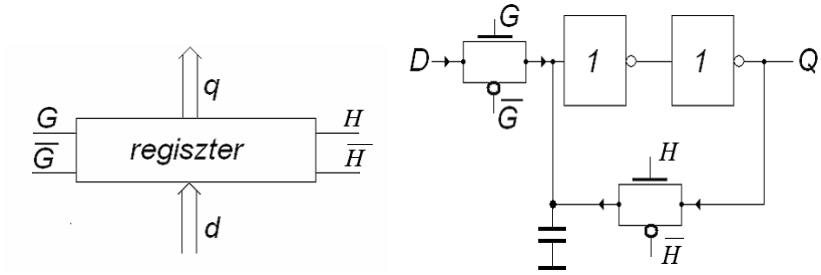
nál van bekapcsolva. Ez a beírás után azonnal bekövetkezik, és megvalósul a tárolás. Ennek megfelelően a regiszter bemenetei között a G vezérlővezetéknek mind a ponált, mind a negált változata megjelenik. (3.11. ábra)



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

3.2.3. Kvázistatikus regiszter

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

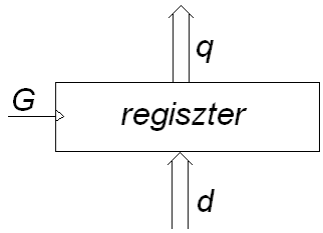


3.12. ábra. Kvázistatikus regiszter szimbóluma és egy-bitnyi struktúrája.

3.2.4. Élvezérelt regiszter

Az élvezérelt regiszterekben az átlátszóság a beíró-jel valamelyik éléhez kötődik. Ez lehet a beíró jel felfutó éle, de lehet a lefutó él is. Az élvezérelt regiszterekből felépített digitális rendszer kevésbé érzékeny az órajelek időbeli elcsúszásából adódó aszinkronitásokra. Az élvezérlést a beíró jel

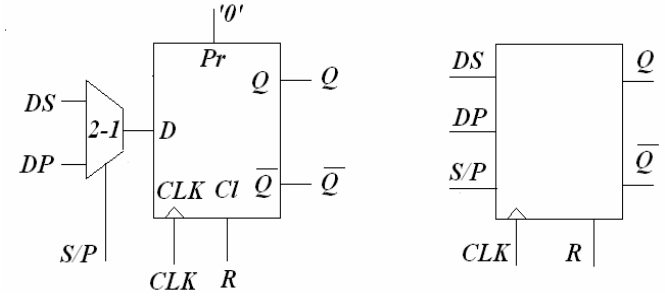
bemenetre elhelyezett speciális szimbólum jelzi. Elfogadott, hogy a felfutó-élre való átlátszóságot a beíró-jel ponált formája jelöli. A cella kapuszintű bemutatásától annak bonyolultsága miatt itt eltekintünk.



3.13. ábra. Felfutó élre beíró regiszter szimbóluma

3.3. Soros elérésű tárolók

A soros elérésű memóriák alapeleme az él-vezérlésű vagy kétfázisú D-MESTER-SZOLGA tároló. Ebből a tárolóból egy 2-1 multiplexer alkalmazásával olyan egységet kapunk, amelynek két adat-bemenete közül (DS, DP) közül az S/P vezérlőjel szintjének egyike választ. A tároló PRESET bemeneteit konstans logikai alacsony szintre kötjük, a CLEAR bemeneteket ezzel szemben a kezdeti „0” állapot beállítására használni fogjuk. A 3.14. ábrán látható egységet soros memóriák építőelemeként fogjuk felhasználni. Az ábra jobboldalán a komponensekből álló séma, a jobboldalon a kapott egység szimbóluma látható.

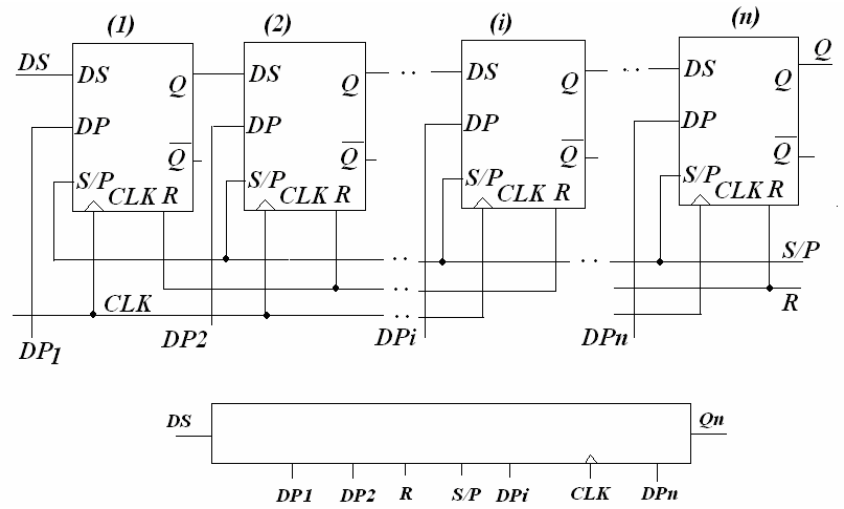


3.14. ábra. Soros memória építő elemének sémája és szimbóluma

3.3.1. Párhuzamosan is betölthető soros memóriák

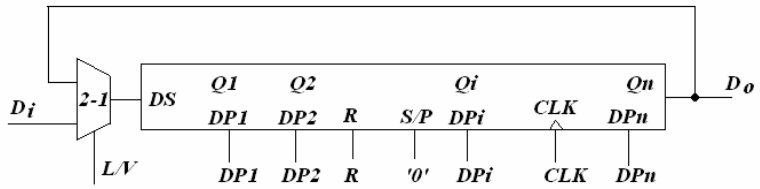
A 3.15. ábra a 3.14. ábra szerinti egységekből felépített nyitott, soros elérésű memória sémája. Az S/P vezérlő-bemenet állapotától függően az óra-

jel-ciklusra vagy balról jobbra léptetés, vagy párhuzamos betöltés történik. Az egység az R jellel alapállapotba hozható.



3.15. ábra. A soros memória sémája és szimbóluma

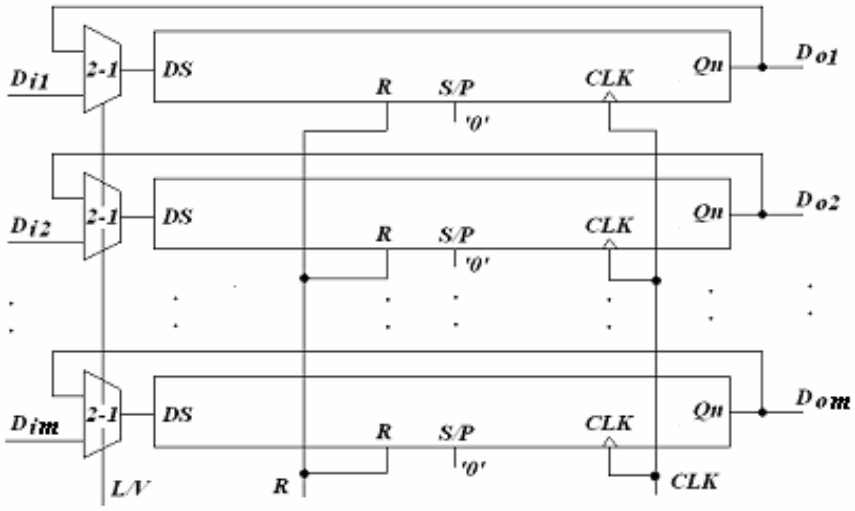
A 3.16. ábra ennek az egységnek az a változata, amikor a memória tartalmát körbe forgatva bármely beírt adat elérhető a kimeneten, de egy adat elvesztése árán új adatot is betölthetünk a D_i bemenetről, az L/V vezérlő bemenet segítségével.



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

3.3.2. Szószervezésű soros memóriák

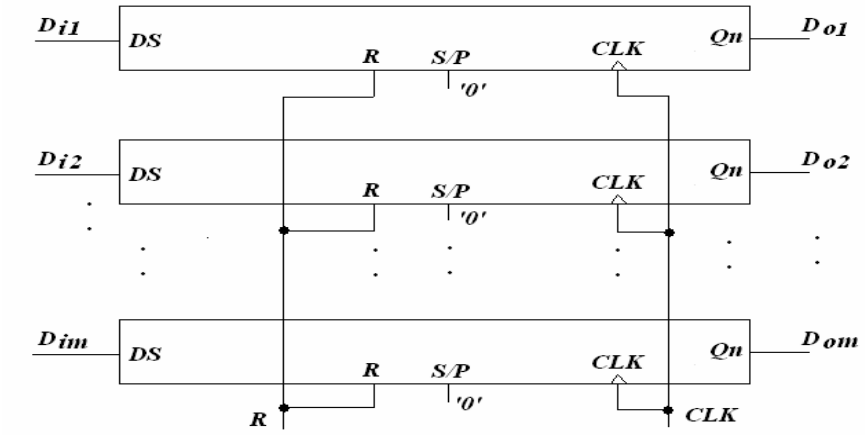
A megismert építőelemből 1-nél nagyobb, például *m* szószélességű soros memóriát építünk (3.17. ábra), elhagyjuk a párhuzamos beírás lehetőségét, azaz az *m* számú gyűrű S/P bemeneteit soros üzemmódra állítjuk be. A memória L/V vezérlő-vezetékével beállíthatjuk, hogy a memóriában lévő adatokat forgatjuk, vagy új adatot szúrunk be a régiek közé.



3.17. ábra. Egy m-bit szószélességű soros memória-egység

3.3.3. FIFO memóriák

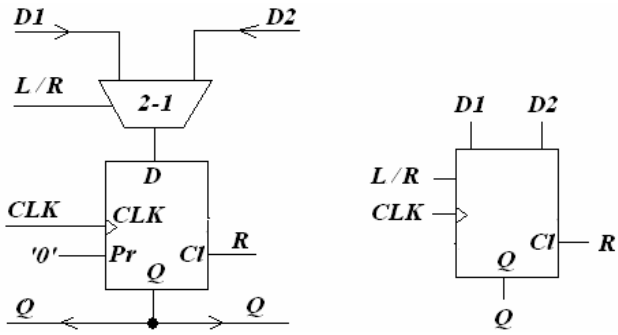
A FIFO olyan soros memória, amelyből az az adat olvasható ki először, amelyet elsőként töltöttünk be. (First In First Out). A fent bemutatott 1-bit szélességű párhuzamosan betölthető soros memória egységekből elvesszük a párhuzamos betöltés lehetőségét és *m*-számú ilyen egységből *m*-bites szószélességű FIFO-t csinálunk, ahogyan azt a 3.18. ábra is mutatja.



3.18. ábra. Egy m-bit szószélességű FIFO memória

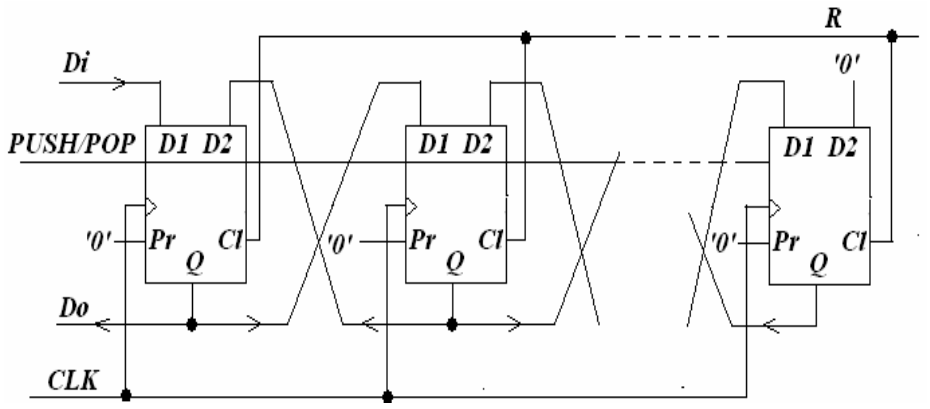
3.3.4. LIFO memóriák

Nézzük a 3.19. ábra szerinti alapelemet. Itt a D-MS flip-flop kimenetén az L/R vezérlőjeltől függően vagy a baloldali D1, vagy a jobboldali D2 bemenet szintje jelenik meg. Ez a LIFO tárolók alapeleme. A LIFO olyan soros memória, amelyből az az adat olvasható ki először, amelyet utoljára töltöttünk be. (Last In First Out). Két vezérlő bemenete van. Betöltésre a BETÖLT (PUSH), kiolvasásra a KIUGRAT (POP) vezérlő-bemeneteket használjuk, természetesen egymás kizárásával. A LIFO egyetlen adatcsatlakozása tehát bemenet és kimenet szerepét is betölti.



3.19. ábra. LIFO memóriaelem

A LIFO elemekből alkotott 1-bit szélességű LIFO memória a 3.20. ábrán látható. Ebből könnyen alkothatunk *m*-szószélességű LIFO memóriát.

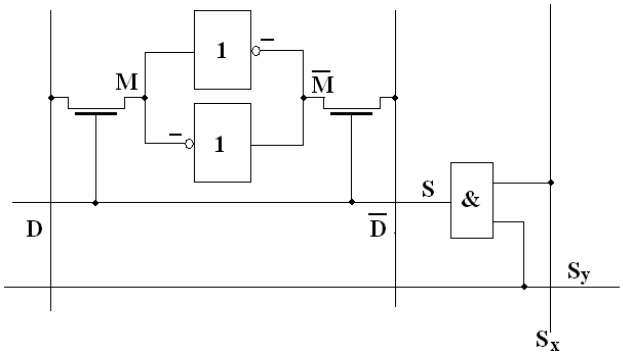


3.20. ábra. LIFO-sor

3.4. Párhuzamos hozzáférésű memóriák

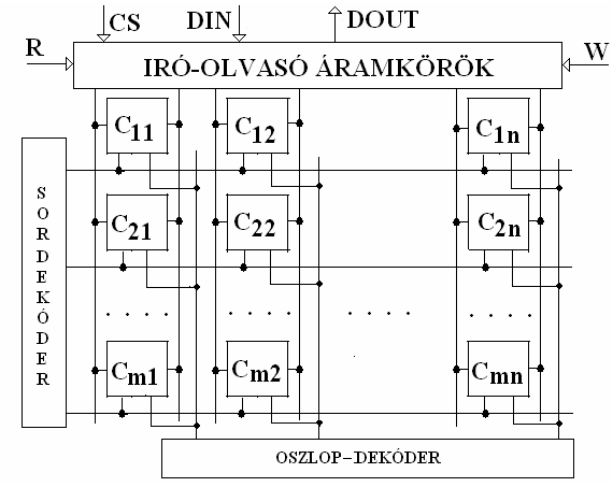
A párhuzamos hozzáférés azt jelenti, hogy a memória minden egyes bitjéhez, vagy minden egyes szavához annak helyétől független elérési idővel férünk hozzá akár átírás, akár kiolvasás szándékával. Ezeket az írható olvasható memóriákat szokás RAM (Random Acccess Memory) egységeknek hívni. A RAM memóriák cellákból állnak. A RAM cellákban nemcsak a korábban említett harmadik állapot lehetőségét használjuk ki, de az azonos logikai szintek erőssége közötti különbségek lehetőségét is. Tekintsük a 3-21. ábrát. Az egymást ölelő inverterek bistabilt alkotnak. Ha a két elektron-vezetésű MOSFET kapcsoló zárva van, azaz az S bemenet alacsony szintű, akkor az inveterek őrzik az utoljára beírt állapotot. Ha S magas szintű lesz, akkor két eset van. (Az S vezérlő-jelet egy Sx és egy Sy kiválasztó-jel ÉS kapcsolatával állítjuk elő).

- Ha a D és a negált-D vonalakat kívülről lebegtetjük, akkor az S felemelkedésekor A D vonalon az M, a negált D vonalon a negált M jelenik meg. Ez tekinthető a tárolt adat kiolvasásának.
- Ha a D és a negált-D vonalakat kívülről ellentétesen meghajtjuk, akkor a logikai szintek egymáshoz viszonyított erőssége határozza meg a lezajló folyamatot. Az inverterek kimenetét „-” jellel jelöltük meg, ezzel kifejezve, hogy azok gyengébbek, mint a D és negált-D értékeket az M és negált M pontra kényszerítő MOSFET kapcsolók. Ha D 1 és M pedig 0, illetve a negált D 0 és a negált M pedig 1, akkor a memória-cella átbillen a másik, az előzővel ellentétes állapotba. Az erősebb illetve gyengébb logikai meghajtó-képességet az invetereket és a kapcsolókat alkotó MOSFET eszközök megfelelő méretezésével lehet elérni.



3.21. ábra. RAM cella

A 3.22. ábrán egy bit-szervezésű RAM látható. A kapacitásától függetlenül csak egy adatkimenete és egy adat-bemenete van, címzése pedig dekóderekkel történik. A megcímzett bit cellájának állapota olvasáskor (R) a „DOUT" bit-kimenetre kerül, míg iráskor (W) a „DIN" bitvektorra kapcsolt érték a memóriának a kijelölt cellájába kerül. Mindkét művelet végrehajtásának a feltétele az is, hogy a CS (Chip Select) vezérlő bemenet magas szintű legyen.



3.22. ábra. Bit-szervezésű RAM hálózat

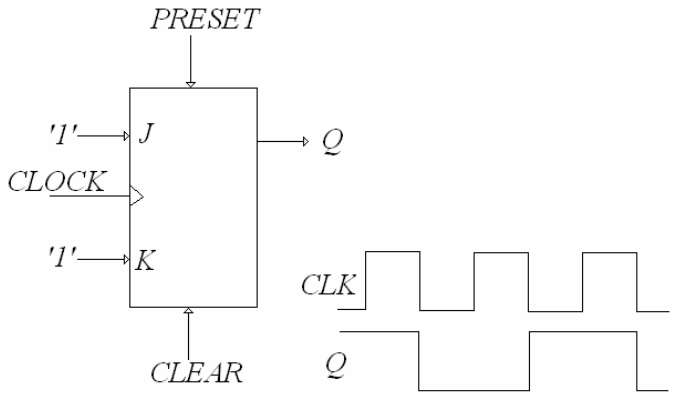
3.5. Számlálók, állapotregiszterek

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

3.5.1. A MESTER-SZOLGA J-K flip-flop mint a számlálók alapeleme

Az aszinkron PRESET és CLEAR bemenetekkel ellátott J-K MESTER-SZOLGA tárolót korábbi tanulmányainkból már jól ismerjük. Az élvezérelt flip-flop funkciót szinkron számlálókban használjuk ki, míg a kettes-

osztó funkciót az aszinkronnak nevezett számlálókban (számláncok) alkalmazzuk. A 3.23. ábra mutatja a kettes osztót, idő-diagrammal. Belátható, hogy amennyiben a MESTER tároló beírása az órajel felfutó, a SZOLGA beírása a lefutó élre történik, az órajel frekvenciája megfeleződik, azaz az így kapcsolt J-K flip-flop az órajel frekvenciát 2-vel osztja. Megjegyezzük, hogy az aszinkron számláncokban az órajel logikai szerepet kap, ezért olyan tárolót kell választani, amelyben a CLEAR bemenet közvetlenül és azonnal hat a kimenetre, az órajel közreműködése nélkül.



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

A számlálókat két csoportra osztjuk, szinkron és aszinkron számlálókra. Mindkét típusú számlálót a *modulo* értékkel jellemzünk. A *modulo-m* (*m*-modulusú) számláló számlálási tartománya a természetes számok m-mel való osztása után kapott lehetséges maradékok tartományán halad át. (0, 1, 2, … $m-1$).

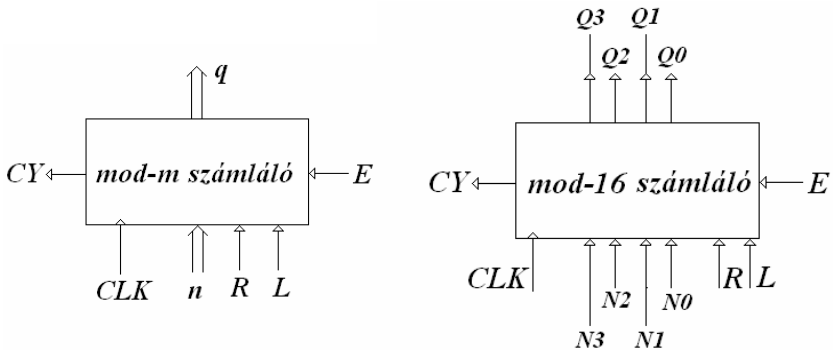
Egy alapvető fogalmat kell még bevezetnünk. A CARRY-LOGIKA olyan kombinációs hálózat, amely a kimenetén 1-et ad, ha a számláló kimenetein az ($m-1$) kódja jelenik meg.

3.5.2. A szinkron számlálók modellje

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

- Az R (RESET) magas szintje a rákövetkező órajel lefutására nullázza a számlálót. Az R jelnek a többi vezérlőjellel összehasonlítva abszolút prioritása van.
- Az L (LOAD) jel hatására, amennyiben nincs R, a számláló tartalma a következő órajel lefutó élére az n bemenetre kapcsolt bit-vektor lesz.
- Az E (ENABLE) jel, amennyiben az R és az L bemenet is logikai 0 szinten van, engedélyezi, hogy a következő órajel lefutó élére a számláló tartalma 1-gyel növekedjék (inkrementálás).

A 3.24. ábrán bemutatjuk a fenti szinkron számláló szimbólumát, és az egyik leggyakoribb modulo-16-os változatot. Megjegyezzük, hogy ennél több funkciót is megvalósítanak, például a felfelé történő számlálás mellett lefelé való számlálást is. (UP-DOWN COUNTERS).



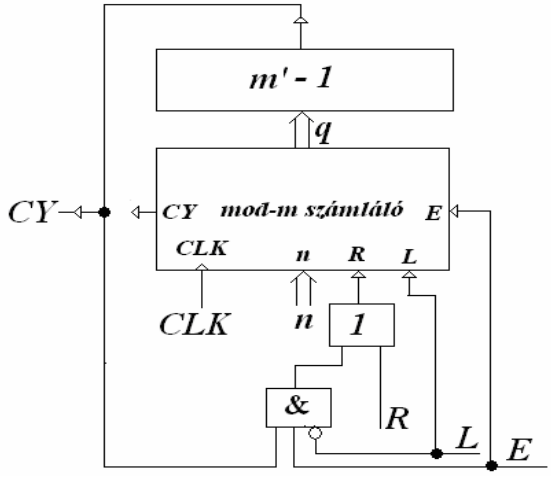
3.24. ábra. Betölthető, engedélyezhető, törölhető szinkron modulo-m számláló általános szimbóluma és a gyakori 4-bites mod.-16-os alaptípus

Ha a bemutatott modulo-16-os számlálót 15-ig (1 1 1 1) felszámoltattuk, akkor a CY (CARRY) kimeneten magas szint jelenik meg. A következő órajel lefutó élére a számláló ismét 0-ra áll (0 0 0 0), és a CY kimenet is alacsony szintre kerül. Ez CY arra alkalmas, hogy egy magasabb helyiértékre helyezett számláló E bemenetét magas szintre állítsa, így lehetővé téve nagyobb modulusú számlálók kialakítását.

3.5.3. Adott modulusú számláló átalakítása más modulusúvá

Ha egy m-modulusú szinkron számlálót át kívánunk alakítani $m' < m$ modulusúvá, akkor új CY hálózatot kell kialakítani. Az új CY detektálja az új $m' - 1$ értéket, és a soron következő órajel a számlálót a RESET bemenet segítségével törli (3.25. ábra) A vezérlő jelek közötti prioritási viszony csak

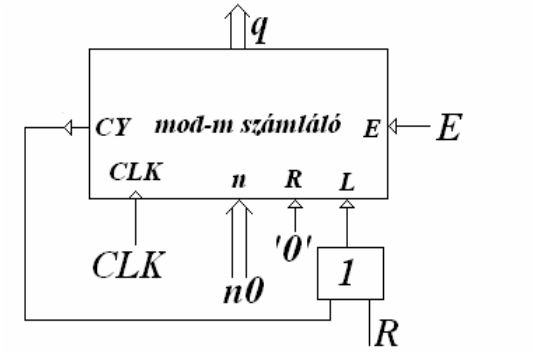
akkor örökíthető át az átalakított számlálóra, ha beiktatjuk a negált-L, E és CY bemenetű ÉS kaput. Az L jel E feletti prioritásának ugyanis akkor is érvényesülnie kell, ha a CY = 1.



3.25. ábra. Egy m-modulusú szinkron számláló átalakítása m' < m modulusú szinkron számlálóvá

3.5.4. Számláló nullától különböző kezdő értékének beállítása

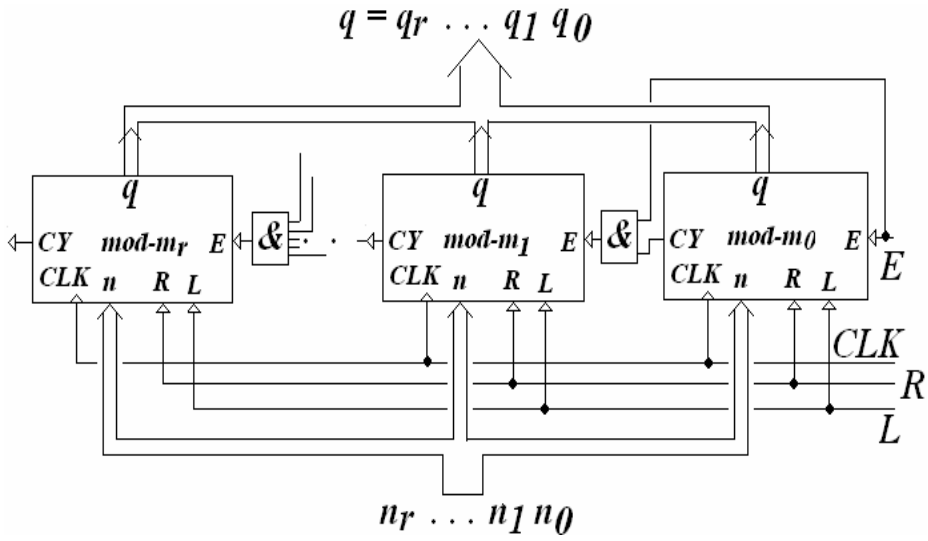
A nullától különböző, de a modulusnál kisebb kezdőértéket az eredeti számláló L jelének felemelésével, a CY =1 feltétellel írjuk a számlálóba. Az új R bemenet ugyancsak az L bemenetre hat. Az eredeti R bemenetet nem használjuk, azt konstans logikai 0 értékre kötjük.



3.26. ábra. 0-tól különböző kezdeti érték beállítása

3.5.5. Szinkron számlálók kaszkádosítása

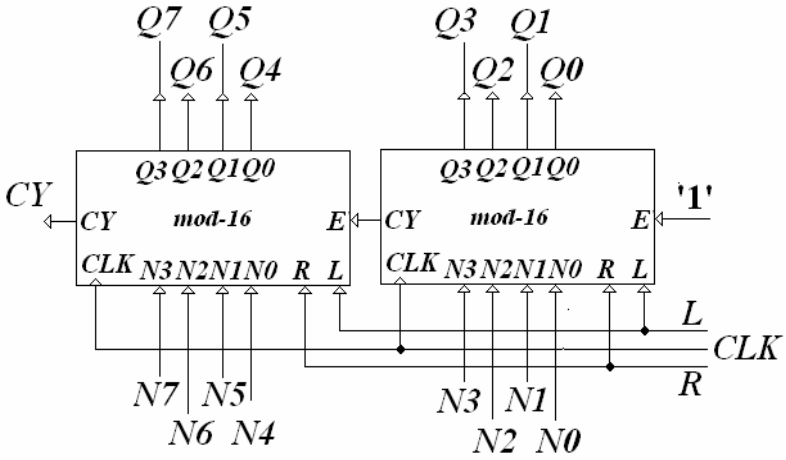
A kaszkádosítás célja, hogy a szinkron számláló modulusát több, kisebb modulusú, leggyakrabban a sorozatban gyártott számlálókból, mint komponensekből állítjuk össze. Minden modul csak akkor lép tovább a soron következő órajelre, ha a tőle jobbra állók mindegyike m−1 értéket mutat. A CY kimenetek tehát ÉS kapcsolatban állnak egymással. Mind a kompozit-számláló tartalmát, mind a betöltendő számot ilyenkor célszerű olyan számláncolattal jelölni, ahol a lánc minden „szeme" az adott komponensre jellemző, az adott modulusnál kisebb, vagy azzal egyenlő szám, $(q_r \ldots q_1\ q_0)$, $(n_r \ldots n_1\ n_0)$.



3.27. ábra. Kompozit szinkron számláló, kaszkádosítással

3.5.6. Példa: Modulo-256-os számláló mod-16 számlálókból

Vizsgáljuk azokat a számláló struktúrákat, amelyekben ENABLE bemenetekre CY logikák csatlakoznak. Például, 4-bites, modulo-16-os számlálókból kialakított struktúra (3.28. ábra).

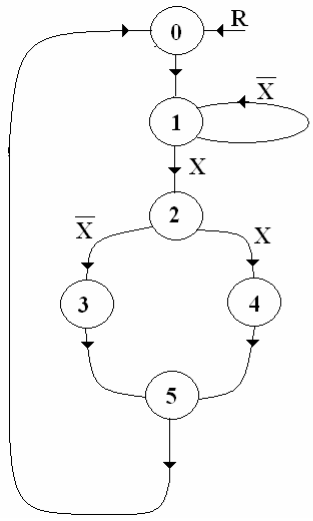


3.28. ábra. Mod-256-os számláló kialakítása mod-16 modulokból

3.5.7. Szinkron számlálók alkalmazása szinkron sorrendi hálózatok tervezésére

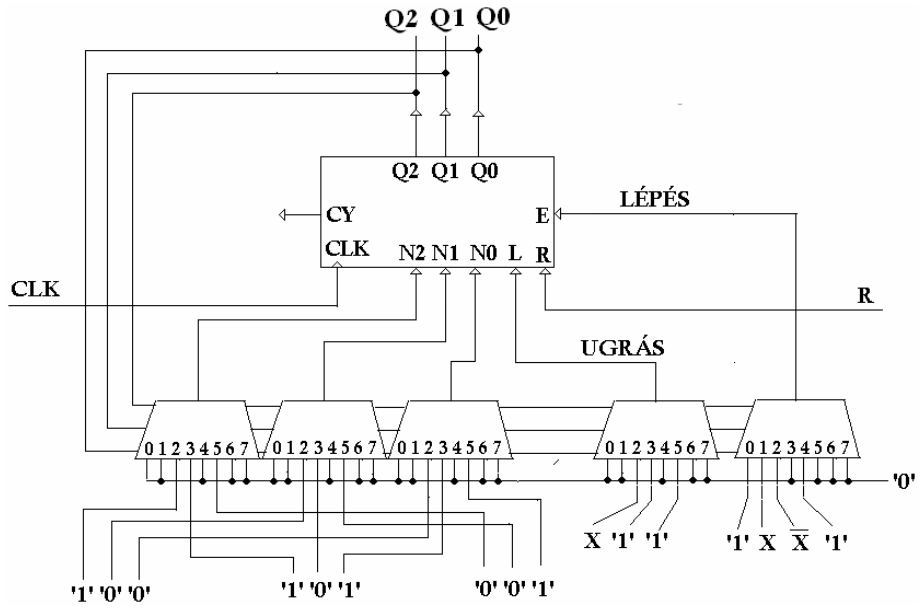
A törölhető, engedélyezhető, betölthető szinkron számlálókat szinkron sorrendi hálózatok állapotregiszteréként alkalmazhatjuk. Ez azt jelenti, hogy a hálózat állapotait a számláló kimenetei kódolják. Ha egy állapot-gráf lehetséges átmeneteit számba vesszük, beláthatjuk, hogy a fenti számláló maradéktalanul képes azokat megvalósítani. Ha egy állapot következő állapota önmaga, akkor ezt a számlálóval úgy realizáljuk, hogy valamennyi vezérlőjelét 0 szinten hagyjuk. Ha az állapot-átmenet a bináris kód inkrementálása, akkor egyedül az E jelet emeljük fel. Ha az átmenet egy nem önmagára és nem az inkrementált kódú következő állapothoz vezet, akkor az L jel felemelésével betöltjük a számlálóba a következő állapot kódját. Az R jel funkciója akkor használható ki a legegyszerűbben, ha a szinkron hálózat kezdő állapotához a 0 bináris kódját rendeljük. Példánkból látni fogjuk, hogy a számlálók vezérlő bemeneteire multiplexereket csatlakoztatunk.

Ezek után tekintsük a 3.29. ábra szerinti állapot-kimenetű szinkron sorrendi hálózat kódolt állapot-gráfját, és az annak alapján felvett két táblázatot. Az egyik táblázat a lépések, azaz az inkrementálások logikai feltételeit, a másik az ugrások logikai feltételeit és következő állapotait tartalmazza. A 3.30. ábra mutatja, hogyan kell az aktuális állapot kódja által megcímzett multiplexer bemenetekre a táblázatok szerinti feltételeket, illetve a betöltendő következő állapotkódokat rákapcsolni.



LÉPÉS (E=1)		UGRÁS (L=1)		
akt. áll.	feltétel	akt.áll.	feltétel	köv.áll.
0	'1'	2	X	4
1	X	3	'1'	5
2	$\overline{X}$	5	'1'	1
4	'1'			

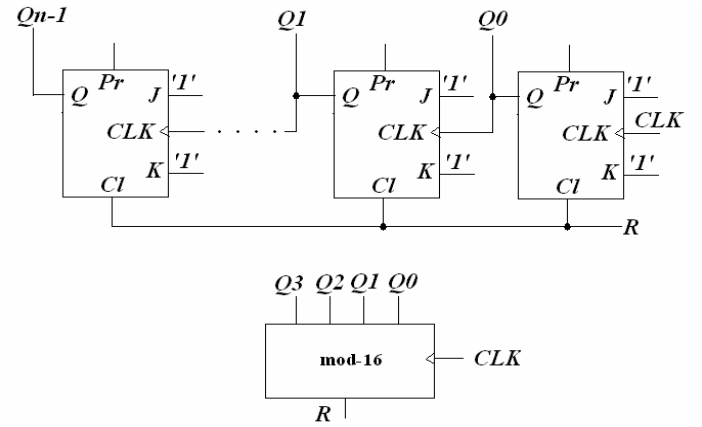
3.29. ábra. Állapot-kimenetű szinkron hálózat tervezéséhez szükséges táblázatok



3.30. ábra. Realizáció mod-8-as számlálóval és 8-1 multiplexerekkel

3.5.8. Aszinkron számlálók

Az aszinkron számlálók alapeleme a kettes osztó. *Modulo-2^n* ($m = 2^n$) számlálót kapunk, ha *n*-számú kettes osztót a 3.31. ábrán látható módon kapcsolunk össze:



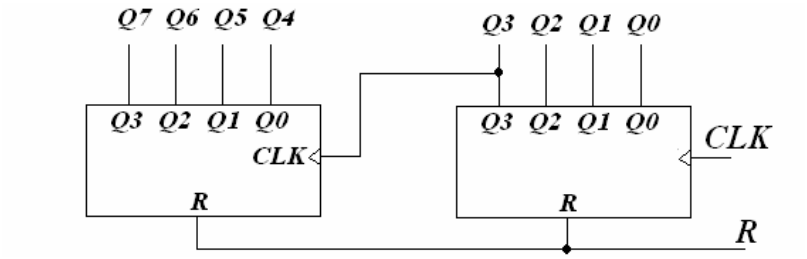
3.31. ábra. Aszinkron számlánc kettes osztókból, és a szimbólum n = 4 esetén

A számlálási sorrend (Q_{n-1} … Q_0) R vezérlés (RESET) után:

000. .00 (0)
000. .01 (1)
000. .10 (2)
. . . . . .
. . . . . .
100. .00 ($2^{(n-1)}$)
. . . . . .
111. .11 (2^n-1) = m−1
000. .00 (0)

3.5.9. Aszinkron számlálók kaszkádosítása

Nem okoz időzítési (hazárd-működésből adódó) gondokat, ha a kaszkád komponensek modulusai kettő hatványai. A 3.32. ábra egy aszinkron *mod-255* számlálót mutat, *mod-16*-os modulokból.



3.32. ábra. Mod-256 számláló 4-bites aszinkron számláncok kaszkádosításával

3.6. Funkciós egységek

A funkciós egységek egy vagy két binárisan ábrázolt adattal valamilyen műveletet végeznek el. A funkcionális egységek kimerítő tárgyalása meghaladná ennek a kurzusnak a kereteit, csupán felvillantjuk a legfontosabb alapelveket. A funkciós egységek opearandusai és eredményei bináris kódban ábrázolt számok. Kettő hatványaival súlyozott bináris kódolás elvét, az egész számok bináris alakjának felírását, a bináris számok decimálisokká való alakításának szabályait már ismerjük. Meg kell azonban ismernünk a negatív számok (egészek és törtek) ábrázolásának azt a módját is, amely könnyűvé teszi az előjeles bináris számokkal való aritmetikai műveleteket. Ezért ebben a bevezető részben megismerjük a komplemens kódok fogalmát és a velük való számolás fő szabályait.

Minden hatványkitevős súlyokkal ábrázolt pozitív számból, függetlenül a számrendszer r alapjától, képezhető egy inverz szám. Az inverz szám minden egyes számjegye helyébe azt a számjegyet írjuk, amelyet az eredeti számjegyhez hozzáadva a rendszer maximális számjegyét kapjuk. Például 3-jegyű decimális számok esetén: Ha N = 642, akkor N_I = 357, hiszen a maximális értékű számjegy 9. Belátható, hogy minden pozitív számra igaz, hogy

$$N + N_I = r^n - 1,$$

Itt *n* a számjegyek száma, azaz az előbbi példa esetében *n* = 3.

Az 3 jegyű pozitív decimális szám és inverzének összege 999, azaz 10^3-1. Ebből következik, hogy az N pozitív szám negatív párja a következőképpen írható fel:

$$-N = -r^n + N_I + 1$$

Tehát a −642 felírható, mint (−1000 + 357 +1).

Ha ezeket a kettes hatványai szerint súlyozott bináris számokra alkalmazzuk, akkor az inverz előállítása azt jelenti, hogy minden egyes számjegyet, mint logikai értéket (0 vagy 1) negálni kell, majd ebből megkapjuk a szám negatív párját, ha a legkisebb helyi-értéken 1-et hozzáadunk az inverzhez. Példa: legyen egy bináris tört a 0. 1 0 1, ahol a bináris pontot a legnagyobb helyi-értékű pozíció után képzeljük. Ilyenkor a szám decimális tört alakban 1.(1/2) + 0.(1/4) + 1.(1/8) = + 5/8

Ennek a számnak az inverze: 1.0 1 0, komplemense pedig

$$\begin{array}{r} 1.\,0\,1\,0 \\ +\,0.\,0\,0\,1 \\ \hline 1.\,0\,1\,1 \end{array}$$

Ez tehát a −5/8 tört bináris, komplemens kódja.

A legfontosabb tulajdonság, hogy a +5/8 és a −5/8 összege bináris összeadással bináris zérust ad, azaz az összevonás műveleti eredményeit az algebra szabályainak megfelelően kapjuk meg.

$$\begin{array}{r} 0.\,1\,0\,1 \\ +\,1.\,0\,1\,1 \\ \hline 1\,0.\,0\,0\,0 \end{array}$$

A (2^1) súlyú pozícióban keletkezett túlcsordulást nem vesszük figyelembe. Ha a negatív számokat abszolút értékük komplemensével jelenítjük meg, akkor a következő aritmetikai szabályok adódnak:

- A számok összeadása az ábrázolásnak megfelelő eredményt szolgáltat
- Egy szám kivonása azonos eredményt ad a komplemensének hozzáadásával.

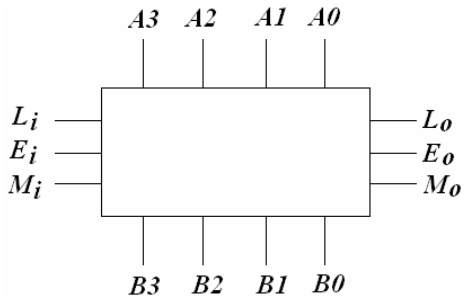
Ebből az következik, hogy az aritmetikai egységünk a kivonást is összeadással végezheti el.

3.6.1. Komparátorok

A komparátorok binárisan ábrázolt adatok összehasonlítását végzik el. A következőkben csak pozitív bináris számok összehasonlításával foglalkozunk, de megjegyezzük, hogy léteznek a komplemens kódra kiterjesztett komparátor változatok is. Az összehasonlításnak általában háromféle eredménye lehet: az egyik adat nagyobb a másiknál, (M_o), azonos értékű a másikkal, (E_o), vagy kisebb a másiknál. (L_o) A logikai áramkörcsaládok legtöbbjénél a komparátorok mindhárom lehetőséget egy-egy kimenettel

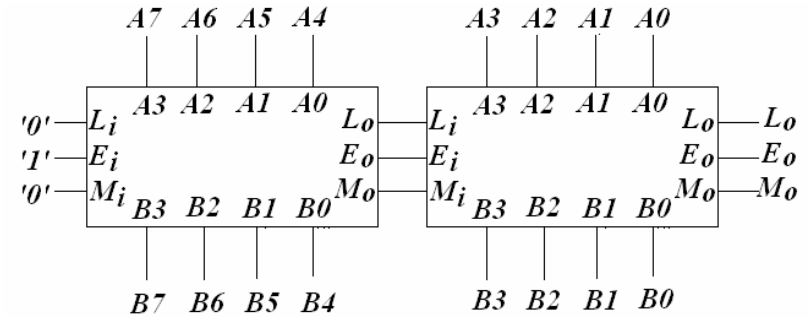
reprezentálják, de a rugalmas felhasználáshoz az kell, hogy az adott méretű adatok összehasonlítását végző egységek összekapcsolhatók legyenek az operandus-méretek növelésének céljából. A bővítést szolgálja az összehasonlítás elve is, nevezetesen, hogy a komparátor az MSB-től az LSB felé haladva mindaddig nem dönt, amíg valamelyik bit-pozícióban eltérést nem talál. Az eltérés döntést eredményez, és feleslegessé teszi a további alacsonyabb helyi értékű bitek összehasonlítását.

Ha egy adott méretű, például egy 4-bites komparátort három bemenettel is ellátunk (M_i, E_i, L_i) és a magasabb helyi értékű 4-bitre elhelyezett komparátor kimeneteivel vezéreljük, valamint biztosítjuk, hogy a már feljebb meghozott $M_o = 1$ vagy $L_o = 1$ döntés egyenesen kijusson a megfelelő kimenetre, akkor egy bővíthető (kaszkádosítható) komparátor egységet kapunk. Egy ilyen komparátor egységet látunk a 3.33. ábrán.



3.33. ábra. 4-bites komparátor egység

A 3.34. ábra mutatja, hogyan kapcsoljuk össze a két 4-bites egységet 8-bitessé. Ábránkon a baloldali modul balszélső, így bemeneteire a megfelelő logikai konstans értékeket kell kapcsolni.

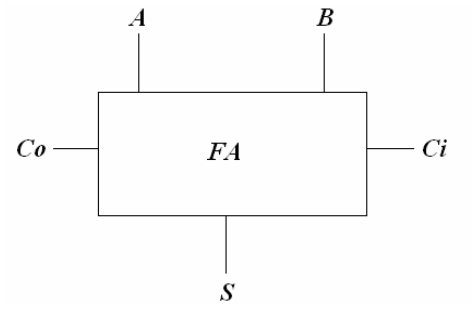


3.34. ábra. 8-bites komparátor két 4-bites egységből

3.6.2. Összeadók

A teljes összeadó (1-bites összeadó)

A legtöbb aritmetikai logikai egység alapeleme az 1-bites összeadó. Az egység igazság-táblázata könnyen megalkotható, ha elképzeljük a bináris összeadás műveletét egy adott helyi-értéken, ahová áthozhatunk értéket a megelőző, kisebb helyi-értékről, (C_i), hozzáadjuk az adott helyi-értéken megjelenő operandus bitek (A, B) összegét (S), és képezzük a nagyobb helyi-értékre való átvitel értékét, (C_o). A teljes összeadó szimbóluma és igazság-táblája látható a 3.35. ábrán.



3.35. ábra. A teljes összeadó szimbóluma

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A tervezés K-táblán való elvégzése utáni logikai kifejezések:

$$S = (A \oplus B) \oplus C_i$$

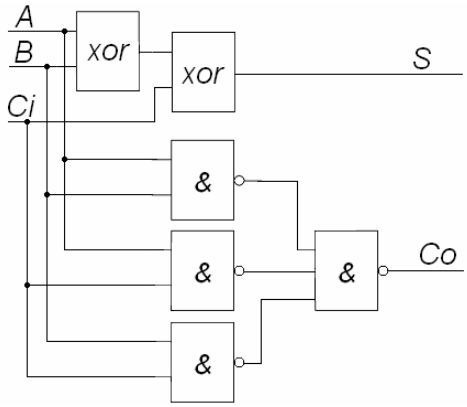
$$C_o = AB + BC_i + AC_i \qquad (1)$$

$$C_o = (A \oplus B)C_i + AB \qquad (2)$$

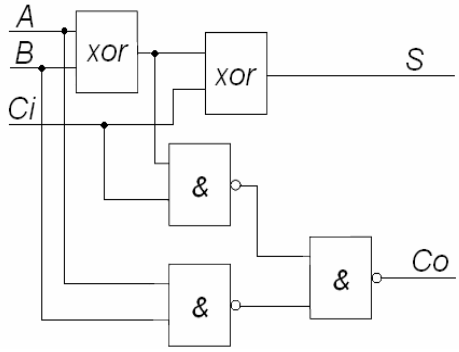
Itt a „⊕” szimbólum a „kizáró-vagy” (EXOR) jele.

Láthatjuk, hogy az átvitel (C_o) kifejezése kétféleképpen is felírható. A második kihasználja az összegben már szereplő EXOR eredményt.

Mindkét megoldás kapu-szintű struktúrája látható a 3.36. és 3.37. ábrákon. Az első megoldás hátránya, hogy több kapuból áll, mint a második, ugyanakkor előnye, hogy mind az összeg, mind az átvitel két kapunyi úton halad át, szemben a második megoldással, ahol az átvitel három kapun keresztül alakul ki.



3.36. ábra. A teljes összeadó gyorsabb változata

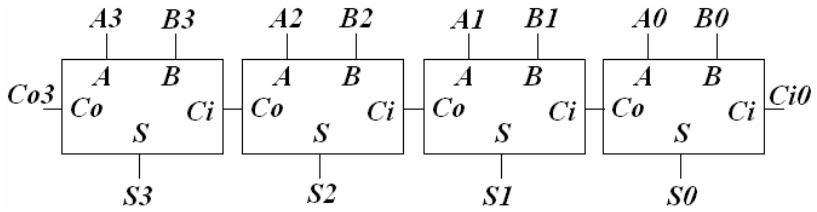


3.37. ábra. A teljes összeadó egyszerűbb változat.

Soros átvitelképzésű bit-vektor összeadó

A teljes összeadók kaszkádosításával soros átvitelképzésű bit-vektor összeadót kapunk. (3.38. ábra). A soros átvitelképzés óriási hátránya, hogy a legnagyobb helyi-értéken az eredmény az összes fokozaton által késleltetve jelenik meg. Minél szélesebb az összeadó, annál nagyobb lesz a műveleti

idő. A műveleti idő csökkentését szolgálják a párhuzamos átvitel-képzésű, illetve a vegyes, párhuzamos-soros átvitel-képzésű összeadók.



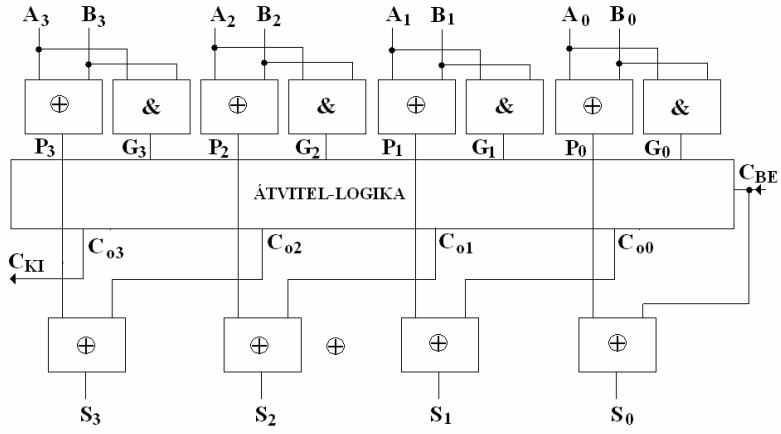
3.38. ábra. Soros átvitelképzésű, 4-bites összeadó egység

Párhuzamos átvitelképzésű bit-vektor összeadó

Minden helyiértéken, így a k. helyi-értéken is, képezzük a következő logikai jeleket:

$$P_k = A_k \oplus B_k$$
$$G_k = A_k\, B_k$$
$$S_k = (A_k \oplus B_k) \oplus C_{ik} = P_k \oplus C_{o(k-1)}$$
$$C_{ok} = (A_k \oplus B_k)\, C_{ik} + A_k\, B_k = P_k C_{o(k-1)} + G_k$$

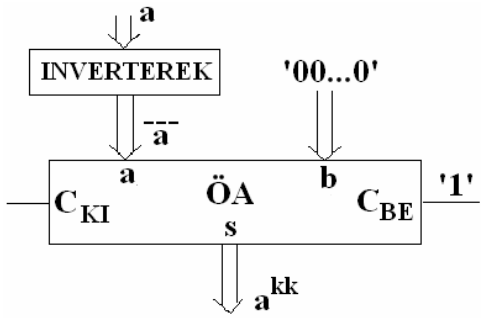
Elvégezve a legkisebb helyiértéktől kezdve a C_{ok} értékek kiszámítását, és minden indexnél behelyettesítve a kisebb helyiértékek bemeneteit, olyan logikai kifejezéseket kapunk, amelynek alapján a 3.39. ábra sémája rajzolható fel.



3.39. ábra. 4-bites, párhuzamos átvitelképzésű összeadó

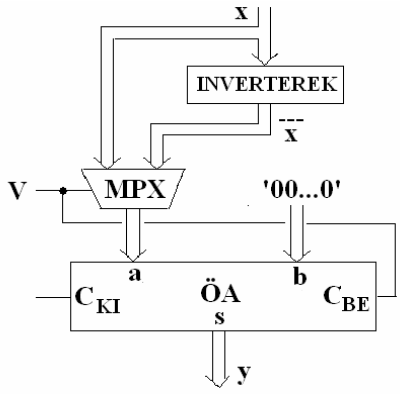
3.6.3. Kivonás

A kivonás műveletét a bináris számok kódolásának megfelelő megválasztásával összeadásra lehet visszavezetni. Azt a bináris kódot, amelynek alkalmazásakor az összeadás fenti módokon való elvégzése a kivonás műveletét is kiszolgálja, a már tárgyalt komplemens kód. A 3.40. ábra szerinti egység az *a* kettes komplemens kódban ábrázolt szám kettes komplemensét állítja elő. Azaz, minden kivonandót rajta átengedve, összeadással hajthatjuk végre a kivonást.



3.40. ábra. Kettes-komplemens-képző egység

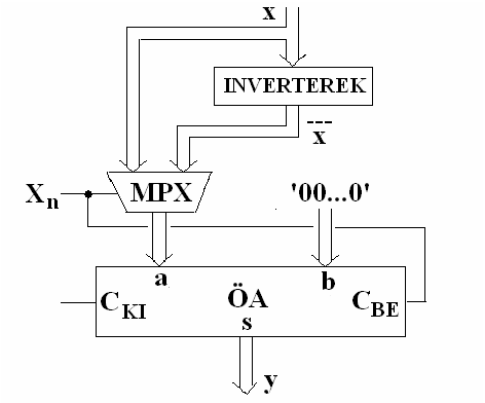
A 3.41. ábra szerinti egység V = 1 esetén az *x* kettes komplemensét állítja elő, míg V = 0 esetén *y* = *x*.



3.41. ábra. Vezérelhető kettes-komplemens képző egység.

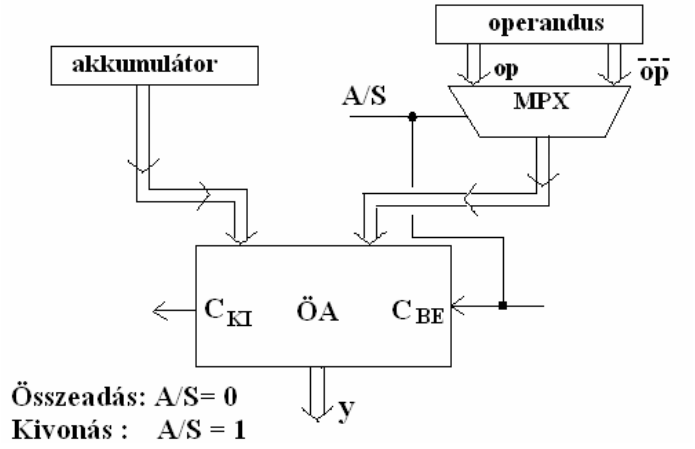
A 3.42. ábrán abszolútérték-képző egységet láthatunk. Bármely kettes komplemens kódban érkező szám abszolút értéke kerül a kimenetre. Az

X_n, az MSB, egyben a szám előjele. Ha „1”, akkor a szám kettes komplemense lesz az abszolút érték.



3.42. ábra. Abszolútérték-képző egység

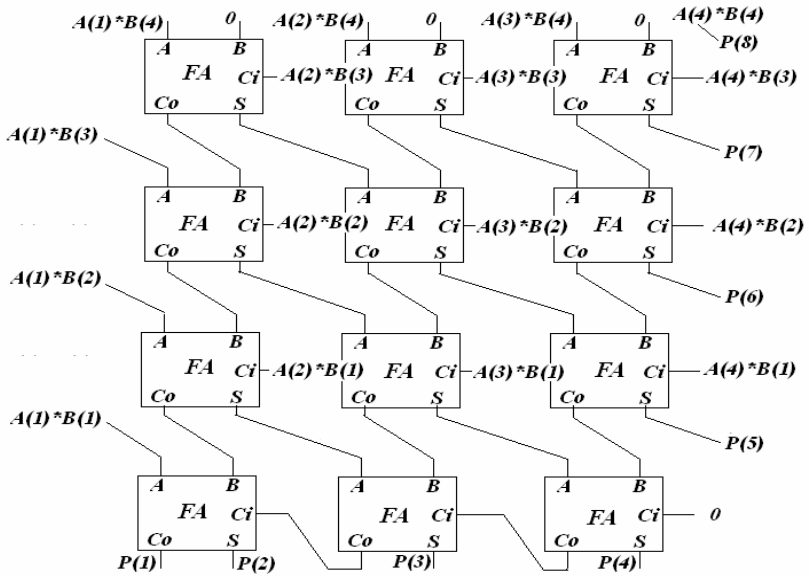
A 3.43. ábrán azt mutatjuk be, hogyan vezetik vissza a mikroprocesszorokban a kivonás műveletét összeadásra. Ha összeadunk, A/S = 0, így az összeadó jobboldali bemenetére maga az operandus kerül az azt tároló regiszterből. Ezzel szemben, ha A/S = 1, azaz kivonni kell, a multiplexer az operandus bitenkénti negáltját kapcsolja az összeadóra, sőt, a C_{BE} bemenet is 1, azaz az operandus kettes-komplemense kerül összeadásra az akkumulátor tartalmával.



3.43. ábra. Kivonás mikroprocesszorok aritmetikai egységében

3.6.4. Szorzók

A szorzókat, hasonlóan az összeadókhoz az jellemzi, hogy a bináris vektorok közötti bitenkénti szorzás-léptetés-összeadás műveletsorát végzik el. A szorzó megoldások igen nagy száma miatt ezeket részletezni nem fogjuk, inkább egyetlen megoldást mutatunk be, egy array-szorzót. (3.44. ábra) A tömb speciálisan összekapcsolt teljes-összeadókból áll. Az A(i)*B(j) jelölések ÉS kapukat szimbolizálnak. Az ábrán két 4-bites bináris tört-vektor szorzását mutatjuk be, az indexek a bináris ponttól jobb felé, azaz a kisebb helyi-értékek felé növekednek. Így a szorzat legnagyobb helyi-értékén P(1), a legkisebb helyi-értékén P(8) szorzat-bit szerepel. Hangsúlyozzuk, hogy az ábra szerinti tömb pozitív számok szorzására alkalmas, de hasonló tömb szorzó megoldásokat fejlesztettek ki kettes komplemens kódban ábrázolt előjeles számok kettes komplemens kódú eredményt szolgáltató szorzására is.



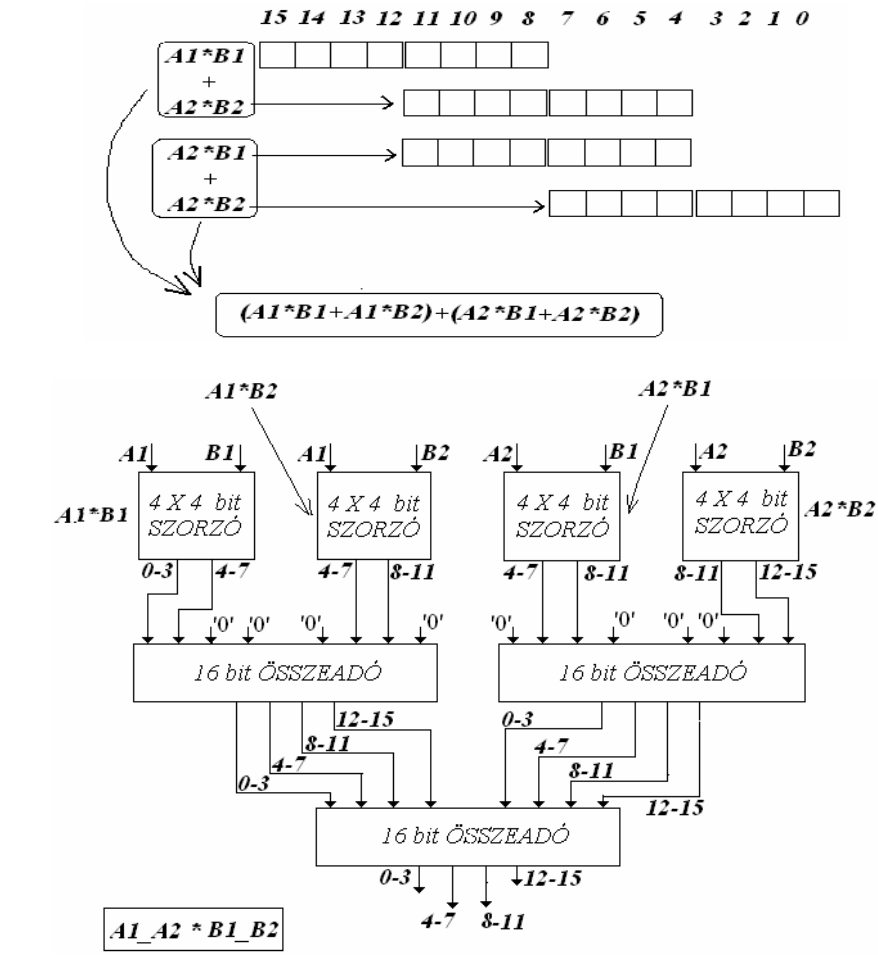
3.44. ábra. 4×4-es tömb-szorzó

Megjegyezzük, hogy a tömb-szorzó késleltetése a bitvektorok dimenzió-számával közel lineárisan nő.

A 3.45. ábra azt mutatja, hogyan lehet viszonylag kisméretű, például 4-bites tömbökből nagyobb méretű, pl. 8×8 bites szorzót kialakítani. Ha az A1_A2 két négybites vektor lánca az egyik operandus, a B1_B2, amely

ugyancsak két négybites vektor lánca a másik operandus, akkor a szorzatokat 4×4 bites szorzókkal, részletekben is elő lehet állítani. Ezután minden részlet-szorzatot a maga helyi-értékének megfelelő helyen kell 16-bites összeadókra vezetni.

Figyelem! A 3.45. ábra szerinti séma minden egyes összeköttetése egy 4-bites vektort jelöl.



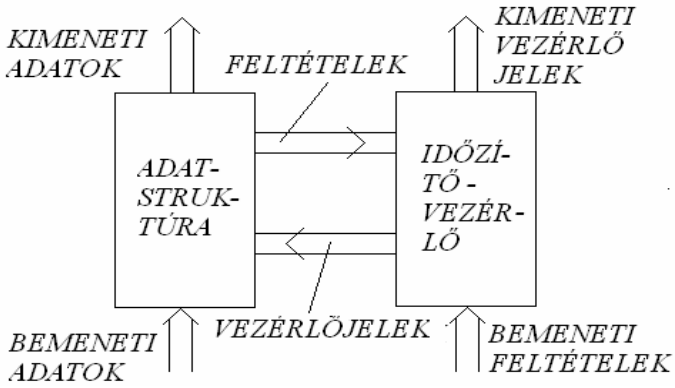
3.45. ábra. 8×8 bites szorzó, 4×4-es komponensekből (A szorzat MSB: 0, LSB: 15).

3.7. Vezérlő egységek

Ebben a fejezetben bemutatjuk a digitális berendezések tervezésének azt a módszerét, amelynek első lépése egy felbontás, azaz dekompozíció. Ez a gondolat elkülöníti az adat-utakat az azok kijelölését és időbeli mozgatását meghatározó időzítő-vezérlő egységtől.

3.7.1. Digitális egység felbontása adat- és vezérlő-alegységre

Az adatstruktúrára és időzítő-vezérlő egységre való felbontás szükségessé teszi a két egység közötti kapcsolódási pontok pontos meghatározását. Az adat-struktúra lényegében regiszterekből, multiplexerekből és funkciós egységekből áll, bemeneti adatokat fogad, és kimeneti adatokat szolgáltat. Ezek az egységek egymással összekapcsolódva különféle adat-pályákat tesznek lehetővé. Az, hogy ezek közül adott helyzetben mely pályák valósulnak meg, illetve e pályáknak mely szakaszai aktivizálódnak egy adott pillanatban, ezt az időzítő-vezérlő határozza meg. Az időzítő-vezérlő címzi meg a multiplexerek és forrásait illetve kimeneteit, állítja be funkciós egységek művelet-vezérlő kódjait, és felemeli, illetve lebocsátja a regiszterek beíró jeleit. Ezeket nevezzük vezérlő-jeleknek. Az időzítő vezérlő működését azonban az adat-struktúrából rávezetett jelek, feltételek befolyásolják. Ugyanakkor a környezet bemeneti-feltételekkel befolyásolja az időzítő-vezérlő működését, és maga az időzítő vezérlő is szolgáltat kimeneti vezérlő jeleket a környezet számára (3.46. ábra). A következőkben az időzítő-vezérlő egységet röviden vezérlő egységnek fogjuk nevezni.



3.46. ábra. Digitális egység (rendszer) felbontása.

3.7.2. Számláló-típusú vezérlők

A számláló típusú vezérlő három fő részből áll. Alapegysége egy törölhető, engedélyezhető, betölthető szinkron számláló, amely a három multiplexerrel együtt egy állapot-kimenetű szinkron sorrendi hálózatot valósít meg. egyet. Nézzük a 3.47. ábrát. A számláló vezérlő-bemenetei között fennálló elsőbbségi viszonyokat figyelembe véve a működés a következőképpen írható le:

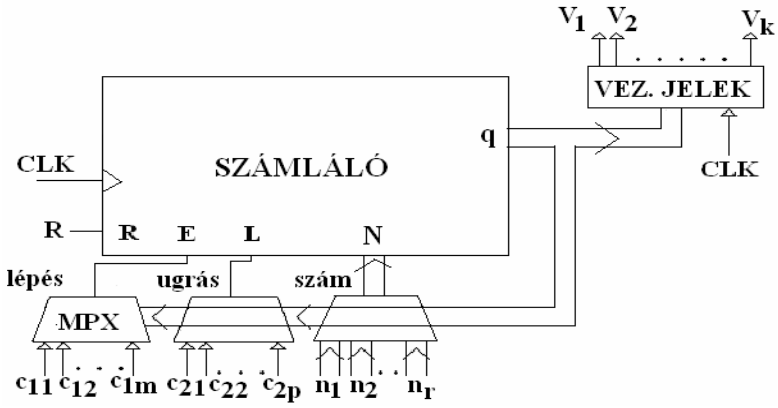
- Ha az R bemenetet felemeljük, a számláló tartalma a fennálló helyzettől függetlenül nullázódik. Ezzel szemben, ha
- az R bemenet szintje alacsony, akkor két eset van:

Egyik, ha a számláló kimenete által megcímzett, a betöltést vezérlő bemenetre kapcsolódó feltétel magas szintű, akkor az általa egyidejűleg megcímzett szám betöltődik a számlálóba. Ezzel szemben áll,

- ha a betöltésre kapcsolódó feltétel szintje alacsony.

Ekkor ismét két eset van:

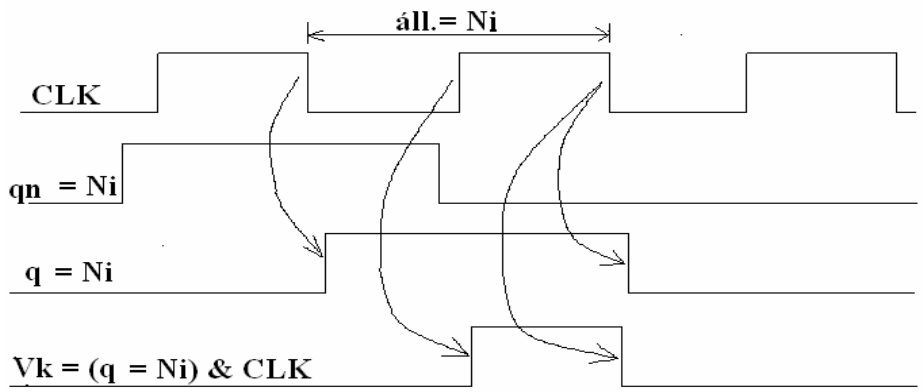
Egyik, ha a kiválasztott, az engedélyező bemenetre kapcsolódó feltétel magas. Ilyenkor a számláló tartalma inkrementálódik, ezzel szemben, ha az engedélyező bemenetre kapcsolódó feltétel szintje alacsony, akkor a számláló értéke nem változik. Mindezeket a 3.5.7. pontban már részleteztük, de a vezérlő harmadik egységét ott nem tárgyaltuk.



3.47. ábra. Számláló típusú vezérlő

A harmadik fő egység a környezetnek és az adat-struktúrának szóló vezérlő-jeleket generálja. A vezérlőjelek generálásának alapproblémája a hazárd-

mentesség. Ez azt jelenti, hogy a vezérlőjelek csakis a tervező által meghatározott időintervallumokban lehetnek aktívak, azokon kívül stabilan passzív logikai szinten kell azokat tartani. Ha valamennyi vezérlő jel aktivitását magával az órajellel szinkronizáljuk, akkor a 3.48. ábra szerinti idődiagram igazolja a hazárdmentességet.



3.48. ábra. A számláló típusú vezérlő időzítése.
qn: a következő számláló állás, q az aktuális számláló állás

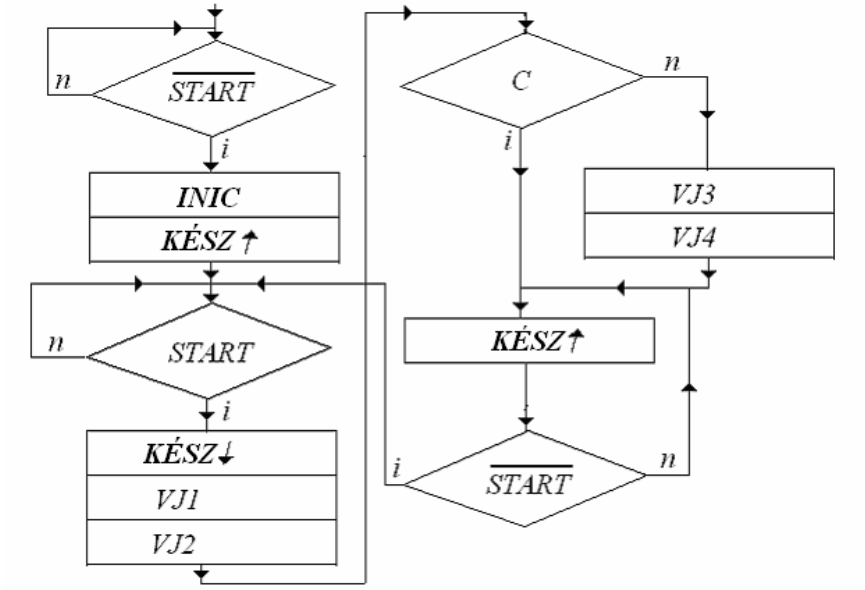
Tegyük fel, hogy a $q = N_i$ számláló-álláskor, és csakis akkor szeretnénk a V_k vezérlő-jel aktivitását kiváltani, az órajellel szinkronban. Belátható, hogy a vezérlőjel bekövetkezése után következő órajel már az időzítési feltétel biztos elmúlása után jelenik meg.

3.7.3. Példa számláló típusú vezérlő egység tervezésére

A megvalósítandó időzítő-vezérlő egységet a 3.49. ábra szerinti folyamatábra definiálja. Eszerint az adat-struktúra számára szolgáltatni kell egy INIC nevű jelet, valamint a VJ1, VJ2, VJ3, VJ4 vezérlő-jeleket, a környezetből fogadni kell egy START jelet, tegyük fel továbbá, hogy az adatstruktúrából ered a C nevű jel, és szolgáltatni kell a környezet számára a KÉSZ jelet.

A folyamatábra szimbólumainak jelentése:

1. rombusz, logikai feltétellel: Ha igaz, az „i” kimeneten távozunk
2. Négyszög, jelnév-bejegyzéssel: órajellel szinkronizált pozitív impulzus a megnevezett jelen.
3. Négyszög jelnév-bejegyzéssel és felfelé v lefelé mutató nyíllal: Az órajel felfutó élére szinkronizált felfutás vagy lefutás a meg nevezett jelen

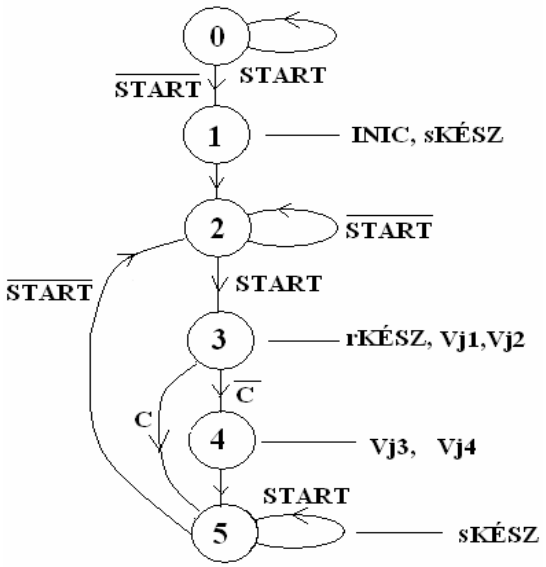


3.49. ábra. Egy vezérlő-egység működésének megadása folyamatábrával

Eszerint a vezérlő-egység, amennyiben a START alacsony, produkál egy INIC impulzust, és a KÉSZ felemelésével jelzi a készenlétet. Ezután vár a START felfutó élére. Ha az megjelenik, visszaejti a KÉSZ jelet, és a VJ1, VJ2 vezérlőjeleken párhuzamosan impulzust produkál. A következő akciók a C bemenettől függenek. Amennyiben C alacsony szintű, akkor a VJ3, VJ4 impulzusai megjelennek, ha magas, akkor azok kimaradnak. Ezután megint fel kell emelni a KÉSZ jelet, és a START szintje szerint visszatérni.

A vezérlési folyamat ütemezése, azaz egy vezérlési szekvencia leírás elkészítése

A folyamatábra akcióit a számláló-állásokkal jelölt állapot-gráfban rendezzük. A számláló-állásokhoz rendelt *akciókat* a gráf jobboldalán soroljuk fel. Az akció rész lehet üres. Ilyen például az első ütem, amikor a START értékétől függően várakozunk, vagy egy ütemmel tovább lépünk. Figyeljük meg, hogy itt a jelemeléseket és -ejtéseket „s”, ill. „r” előtagokkal jelöljük, érzékeltetve ezzel, hogy az ilyen típusú jeleket S-R bistabilokkal állítjuk elő. Így minden ilyen jelhez két impulzus-típusú jelet rendelünk. Ezzel azt is elérjük, hogy az ütemezett vezérlési szekvencia már csak impulzus-típusú jelekre hivatkozik. Lássuk tehát az ütemezett vezérlési-szekvencia leírást, a 3.50. ábrán.



3.50. ábra. A számláló állapotainak gráfja akciókkal

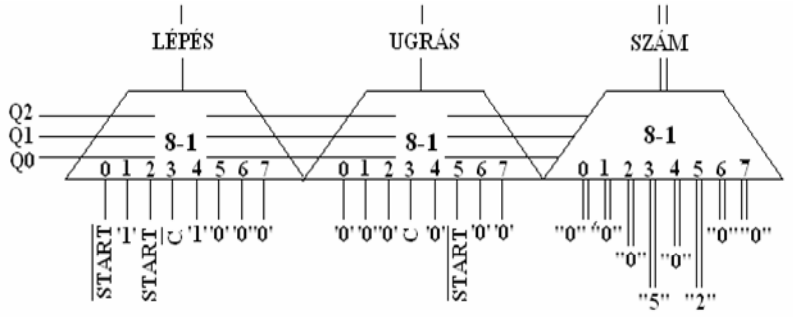
A multiplexerek megtervezése a vezérlési szekvencia-leírásból

A realizáláshoz *mod-8* számlálót fogunk használni, tehát 3-bites számlálót. A „LÉPÉS" multiplexer megtervezésekor azokat az ütemeket gyűjtjük össze, amelyekben inkrementálás található. Az ütemek az inkrementáláshoz tartozó feltételeket címzik. Feltétel nélküli inkrementálási ütemhez nyilvánvalóan konstans logikai 1 tartozik. Hasonlóan gyűjtjük össze az „UGRÁS" multiplexer címeit és bemeneteit is. Ugyanezekhez a címekhez rendeljük a „SZÁM" multiplexer bemeneteit is, amelyekhez az ugrásokhoz beírt ütemszámokat rendeljük.

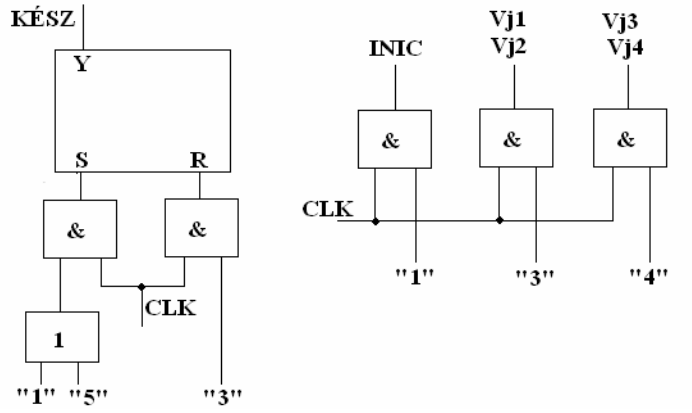
A vezérlőjel-generátorok tervezése

Az INIC valamint a VJ1 … VJ4 vezérlőjelek előállítása NEM-VAGY kapukkal történhet, a számlálóról levett és dekódolt ütemszámok felhasználásával. Mivel a kapuk bemenetére a negáltakat kell kapcsolni, a dekódolt ütemszámok negáltját tüntettük fel a kapuk bemenetein.

A KÉSZ jel előállítása bistabil tárolón keresztül történik. Látható, hogy az alkalmazott NÉS-NÉS bistabilon is az órajellel szinkronban történik meg a változás. A multiplexereket a 3.51. ábra, a vezérlőjel-generátorokat a 3.52. ábra mutatja be.



3.51. ábra. A számlálót vezérlő multiplexerek.
A (Q0 Q1 Q2) vektor elemei a számláló kimenetei



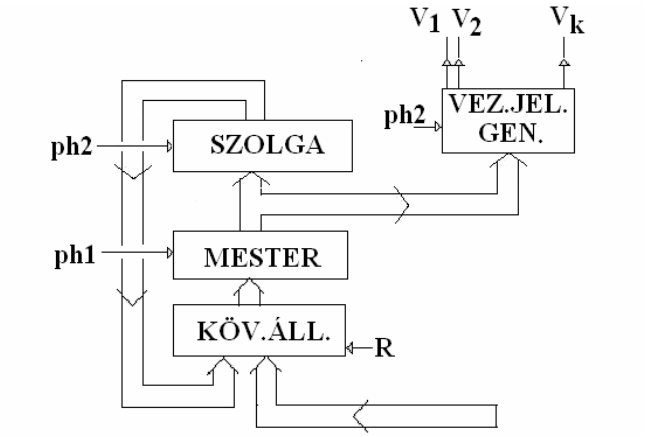
3.52. ábra. A vezérlő-jel generátorok. Az állapotok dekódolt formában, egyetlen bittel szerepelnek az ábrán

3.7.4. FSM típusú vezérlők

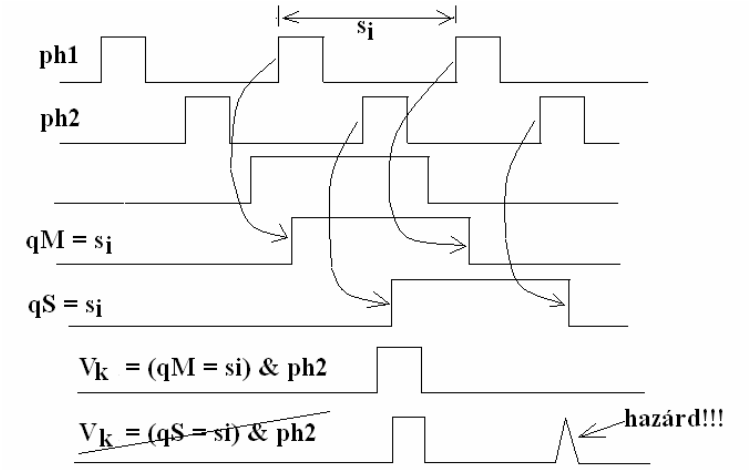
Az FSM (Finite State Machine) típusú vezérlő, nevét arról kapta, hogy az időzítő lényegében véges állapotszámú, MS tárolókból az adott célra felépített szinkron sorrendi hálózat.

Ábránkon egy kétfázisú órajellel működő egység látható (3.53. ábra). Felismerjük a klasszikus, kétfázisú szinkron-sorrendi hálózat struktúrát, ha a flip-flopok MESTER fokozatait is és a SZOLGA fokozatokat is összefogjuk egy-egy él-vezérelt regiszterben. Az idő-diagramon (3.54. ábra) azt mutatjuk be, hogyan kell kivitelezni egy *ph2* fázissal szinkronizált vezérlő-jel generátorát. Látható, hogy csak a MESTER fokozatról levett időzítő-információ eredményez hazárdmentes megoldást. Figyeljünk fel arra, hogy az állapotokhoz tartozó idő-intervallumokat most a *ph1* felfutásai között

definiáljuk. Megjegyezzük, hogy a klasszikus MSI-szintű logikai áramkörök világában inkább a számláló típusú vezérlőt érdemes alkalmazni, hiszen a számláló készen van, és funkciói kihasználhatók, míg az LSI-VLSI világban leginkább a kétfázisú FSM-típusú vezérlő tervezése az ajánlott.



3.53. ábra. Kétfázisú órajellel működő FSM típusú vezérlő



3.54. ábra. Kétfázisú órajellel hajtott FSM típusú időzítő idő-diagramja

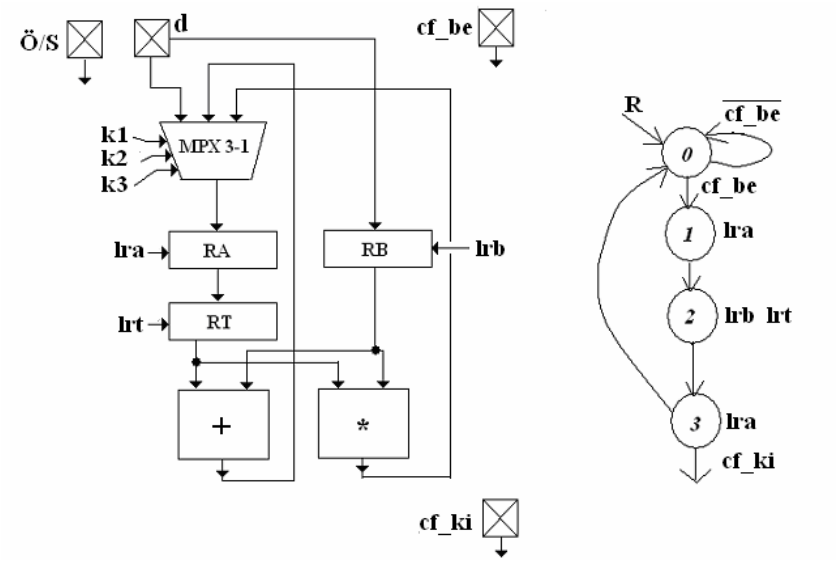
3.7.5. Önálló makrocella tervezése FSM típusú vezérlővel

Az önálló makro-cellák LSI-VLSI áramkörök építőelemei lehetnek. Sajátosságuk, hogy egy speciális feladatra tartalmazzák mind az adatstruktúrát, mind az időzítő-vezérlőt, és beilleszthetők egy nagyobb vezérlési folya-

matba. Fontos, hogy több ilyen önálló egység osztozkodhat az adatstruktúra elemein, ha nem akadályozza ezt erőforrás igények közötti ütközés. Valósítsuk meg a következő önálló (autonóm) makro-cellát:

A megtervezendő egység várakozzon a vezérlési folyamatot átadó másik önálló egységtől vezérelt *cf_be* jel felfutására. Ha ez megérkezett, egymás után két adatot ugyanarról a *d* adat-sín bemenetről írjon be két regiszterbe (RA, RB), majd az Ö/S bemenet szintjétől függően adja össze, vagy szorozza össze azokat, és az eredményt írja vissza az RA regiszterbe. Ha ezt elvégezte, küldje tovább a vezérlést egy harmadik önálló egységnek a *cf_ki* jel felemelésével, ő maga pedig várakozzék újabb indításra, de a *cf_ki* jelet ejtse le alap-állapotába.

Ami az adatstruktúrát illeti, világos, hogy legalább két regiszterre és két funkciós egységre, egy összeadóra és egy szorzóra van szükségünk. Belátható azonban, hogy az egyik operandusnak az eredménnyel való felülírása megköveteli egy átmeneti regiszter (RT) bevezetését is. Nyilvánvaló ugyanis, hogy a funkciós egységek kombinációs egységek, és bemenetükön nem írhatjuk felül azt az adatot, amely az eredmény stabilan tartásához és tárolásához szükséges. Ezért az RA tartalmát átmentjük egy átmeneti tárolóba, ezt kapcsoljuk a funkciós egységekre, és így az RA befogadhatja az eredményt. Azt is könnyen kitalálhatjuk, hogy az RA regiszterbe végül is

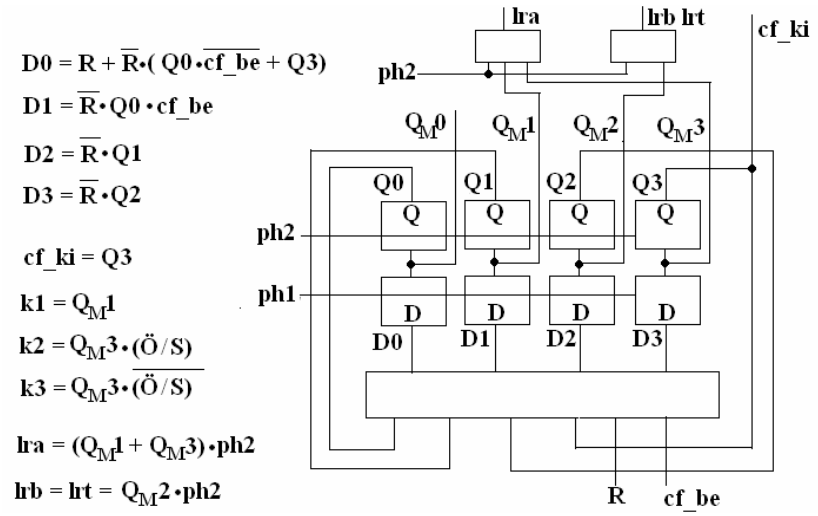


3.55. ábra. FSM típusú időzítő-vezérlővel működő autonóm egység adatstruktúrája és időzítőjének állapotgráfja

három különböző forrásból kell tudni adatot betölteni. Ezek a *d* adatsín, az összeadó kimenete és a szorzó kimenete.

Ezek után felvázolható a 3.55. ábra bal oldalán látható adatstruktúra. Mindhárom regiszterből „kilógnak" a betöltő-jelek, *(lra, lrb, lrt)* a 3-1 multiplexerből pedig a három kiválasztó bemenet, *(k1, k2, k3)*. Természetesen a műveletkiválasztó jel is szerepet kap a vezérlőben, csakúgy, mint az FSM-t elindító *cf_be* külső vezérlő bemenet. Természetesen szolgáltatni kell a *cf_ki* vezérlés-továbbadó jelet is.

Az ábra jobb oldalán az FSM állapot-gráffal megadott ütemezést láthatjuk. A 0 sorszámú. állapot a kezdeti állapot, ahová az R jel kényszeríti az időzítőt. Ebben az állapotban várakozni kell a vezérlési folyamat átadására, azaz a *cf_be* jel felemelkedésére. Innen kezdve sorrendben követik egymást az 1, 2 és 3 sorszámú állapotok. Az 1-es állapotban az *lra*-n beíró-impulzust kell kiváltani. Ezalatt a *d* sínre kell kapcsolódnia az RA regiszter bemeneteinek, azaz a multiplexer *k1* kiválasztó jelét kell magasra emelni. A 2-es állapotban egyszerre töltjük a második adatot a *d*-ről az RB regiszterbe, illetve a RA-ból az első adatot az átmeneti regiszterbe. Az *lrb* és *lrt* betöltő-impulzusok tehát egyszerre jelentkeznek. Innen kezdve a két funkciós egység kimenetén megjelenik az összeg, illetve a szorzat. A 3-as állapotban ismételten az *lra*-ra adunk impulzust, de ez alatt a *k2* vagy a *k3* kiválasztó bemenetek egyikének kell magasan lennie.

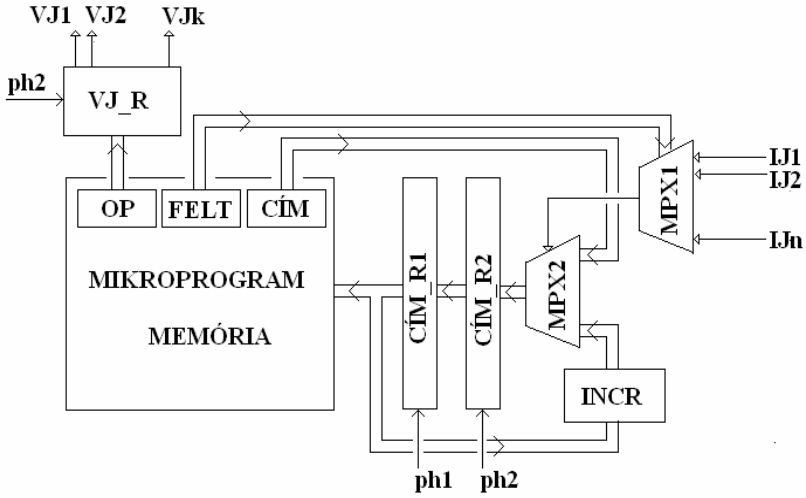


3.56. ábra. A feladatban megfogalmazott időzítő vezérlő megvalósítása. Az időzítőt 1-es súlyú állapotkóddal realizáltuk

A megvalósított időzítő-vezérlő a 3.56. ábrán látható, sematikusan. A D bemenetekre csatlakozó fekete doboz tartalmát a bal oldali logikai kifejezések mutatják.

3.7.6. Vezérlés mikroprogramozással

A mikroprogramozott vezérlő-egység elve, hogy mind a vezérlő-akciókat, mind az elágazásra vonatkozó információkat egy címezhető memóriában, (lehet írható/olvasható, vagy csak olvasható is), az ütemezésnek megfelelő sorrendben helyezzük el, mikro-utasítások formájában. A 3.57. ábrán látható egység egy adott mikroutasítás kiolvasása után vagy a következő mikroutasítás kiolvasásába kezd, vagy elugrik egy megadott címre. Azt, hogy ezek közül melyik eset valósul meg, feltételek teljesülésétől függ. Nyilvánvaló, hogy ez a két lehetőség kettős elágazás megvalósítására is lehetőséget nyújt, hiszen a következő címen elhelyezhetünk egy feltétel nélküli ugrást.



3.57. ábra. A mikroprogramozott időzítő-vezérlő egység sémája

A mikroprogram utasítás-szó felépítése

Az utasítás-szó három részből áll. Az első az akciókat kiváltó, ún. operációs rész, (OP) a második a bemeneti jelek (feltételek) címzésére alkalmas ún. feltétel-rész (FELT), míg a harmadik pedig az a memória cím, ahová akkor ugrunk, ha a FELT rész által kiválasztott jel magas szintű.

Kritikus az OP rész méretének megválasztása. Ha az OP részben minden egyes vezérlőjelhez tartozik egy bit, akkor ezt horizontális mikrokód-

nak nevezik. Előnye, hogy egyetlen mikroutasítással akár valamennyi vezérlő-jel aktivizálható egyetlen mikroutasítással, hátrány viszont, hogy a memória horizontális mérete maximális. Ha az OP rész csak a vezérlőjelek egyikét címzi meg, akkor a mikroutasítás szó a lehető legrövidebb, de egy mikroutasítással csak egyetlen egy vezérlőjel aktivizálható. Ez megnöveli a mikroprogram végrehajtásának idejét. Az ilyen mikroprogramot nevezik vertikális mikrokódnak.

Nyilvánvaló, hogy a két szélsőséges megoldás között számos közbenső változat létezik. Például párhuzamosan címezhető vezérlőjel csoportokon belül egyedi jeleket címezünk.

Nyilvánvaló tehát, hogy a teljes mikroprogram méretét nemcsak a folyamat ütemeinek a száma, hanem a mikroutasítás mérete is befolyásolja.

A mikroprogramozott vezérlő egység időbeli működése

A 3.57. ábra alapján könnyen követhető a működés. A *ph1* órajel felfutására beíródik egy memória-cím a CÍM_R1 regiszterbe, és ennek megfelelően a megcímzett utasítás szó részei megjelennek a memória kimenetein. Ez a mikroutasítás elővétele. Ennek hatására a VJ_R vezérlőjel regiszter bemenetén megjelenik az akciókat vezérlő OP rész, az MPX2 kimenetén megjelenik a FELT rész által megcímzett feltétel, illetve bemenőjel logikai szintje, és az MPX2 két bemenetén megjelenik egyrészt a potenciális ugrás címe, másrészt az INCR hálózaton keresztül az aktuális cím inkrementáltja. Az INCR tulajdonképpen egy összeadó, amelynek egyik operandusa mindig 1.

A következő esemény a *ph2* órajel felfutása. Erre a kiolvasott OP rész megjelenik a vezérlő-jeleken, és az MPX1 szintjétől függően vagy az inkrementált cím, vagy az ugrás címe töltődik be a CÍM_R2 regiszterbe. Ezzel a következő *ph1* felfutásra megindul a következő mikroutasítás elővétele.

4. Bevezetés a mikroprocesszoros rendszerek tervezésébe

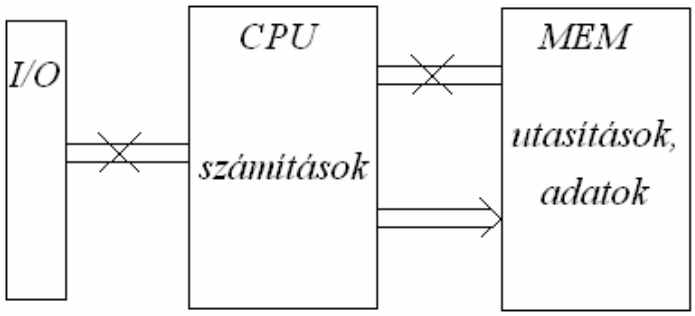
A fejezet célja, hogy az olvasó mikroprocesszoros rendszerek legalapvetőbb fogalmait megismerje, kellő alapismereteket szerezve az ilyen rendszerek tervezési módszereinek elsajátításához.

4.1. A Neumann-féle architektúra

A mikroprocesszorok technikája az elmúlt század hetvenes éveiben alakult ki. Néhány kivételtől eltekintve mind a klasszikus, mind a mai mikroprocesszorok a Neumann-féle architektúra hagyományos vonásait őrzik.
A mikroprocesszoros rendszer fő részei a következők:

- A CPU, azaz a központi egység, amely többnyire maga a mikroprocesszor,
- A MEMÓRIA, amelyben a programot (utasításokat) és az adatok nagy részét is tároljuk valamint az
- I/O, azaz kommunikációs egységek, amelyek a környezettel való adatcserét biztosítják.

A rendszernek ezt a felbontását a 4.1. ábra mutatja. A CPU a memóriával két sín-rendszeren keresztül kommunikál. Az egyikkel címzi a memóriát (ADDRESS-BUS, CÍM-SÍN), a másikon keresztül pedig utasítások érkeznek a CPU-ba, illetve adatok jönnek-mennek. Az I/O egységekkel való kapcsolat mindig kétirányú.



4.1. ábra. Mikroprocesszoros rendszer fő elemei

A CPU a rendszer legaktívabb eleme. Ha bekapcsoljuk, azonnal megcímzi a memória legelső celláját, és feltételezi, hogy amit onnan kiolvas, az egy utasítás. Innen kezdve, tehát az első utasítás végrehajtásától kezdve a processzor vagy a következő utasítást vesz elő, vagy elugrik arra a címre, amelyet az éppen végrehajtott utasítás diktál, és ott folytatja a program végrehajtását. Ezt a rendet persze néhány sajátos és rendkívüli esemény felboríthatja. Ezekkel később részletesen foglalkozunk. A Neumann architektúra sajátossága tehát az utasítások sorrendben történő (szekvenciális) végrehajtása.

4.2. Címzési módok

A CPU a memóriában tárolt utasításokat és adatokat címeik segítségével éri el. A CÍM (ADDRESS) a címbuszon, (ADDRESS-BUS) megjelenő bináris vektor. Ha azt akarjuk, hogy ugyanazt a memóriát több CPU is használhassa, a címbuszt a CPU-nak „lebegtetnie”, azaz szabadon hagyni is tudnia kell. A címbusz tehát egyirányú, háromállapotú sín. Feltételezzük, hogy minden utasítás egyetlen operandusra hivatkozik, mert a másik operandus helye adott. Ez általában az AKKUMULÁTOR regiszter. Így minden utasításnak egy memória címet kell meghatároznia. Ez lehet:

- implicit
- idézetes
- direkt
- indirekt
- indexelt

Az egyes címzési módok jelentése a következő:

Implicit címzés: az utasítás kódja utal az operandus helyére (pl. egy kitüntetett regiszter)

Idézetes címzés: az utasítás tartalmazza magát az operandust.

Direkt címzés: Az utasítás tartalmazza az operandus címét.

Indirekt címzés: Az utasítás tartalmazza azt a címet, ahol az operandus címe megtalálható.

Indexelt címzés: az utasítás egyrészt hivatkozik egy index-regiszter nevére, és megad egy címnövekményt, amit az index regiszter tartalmához adva megkapjuk az operandus címét.

4.3. Utasítások és adatok

Az utasításokat és adatokat a CPU a memóriából veszi, és oda is helyezi el az eredményeket.

Az ADAT-SÍN (DATA-BUS) tehát kétirányú. Ahhoz, hogy a memóriához más CPU is hozzáférjen, az ADAT-SÍN-t is lebegtetni kell. Ez tehát kétirányú, háromállapotú sín. Ha egy CPU-s rendszerben más, intelligens vezérlő-egységek is használják a CPU-hoz rendelt memóriát, akkor a CPU lekapcsolódása a másik egységnek közvetlen memória elérést, DMA-t (Direct Memory Access) tesz lehetővé. Ha több CPU használja ugyanazt a memóriát, akkor memóriában csatolt multiprocesszoros rendszerről beszélünk.

4.4. Szekvenciális program

Az utasítások a memóriában növekvő címük sorrendjében helyezkednek el. A CPU ebben a sorrendben hajtja végre azokat. A címzés első közelítésben tehát számlálás (PC, Program-Counter). Eltérések ettől a szigorú monotonitástól:

- RESET, azaz kezdeti állapotba állítás. Ilyenkor a CPU nullázza a programszámlálót, és a zérus címen újra kezdi a program végrehajtását.
- Ugró utasítás (JUMP), amely lehet feltétel nélküli, vagy feltételes.

Az utasítás tartalmazza a címet, ahová ugrani kell. Feltételes ugrás esetén az ugrás csak a feltétel teljesülése esetén következik be. A feltételt szintén az utasítás tartalmazza.

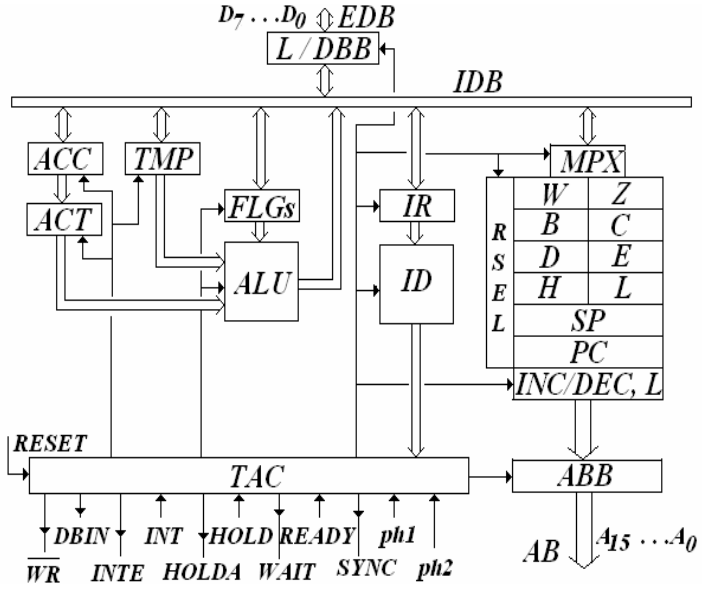
- Szubrutin hívás (CALL) és visszatérés (RETURN)
  Az utasítás tartalmazza a szubrutin (alprogram) kezdőcímét. A processzor ide ugrik, és elkezdi ennek az utasítás-sorozatnak, azaz az alprogramnak a szekvenciális végrehajtását. Az alprogram végén egy RETURN, azaz visszatérési utasítás található. A hívás helyére visszatérni a CPU csak akkor tud, ha a hívás előtti következő utasítás címét a hívás végrehajtásakor el kell tárolni a memória meghatározott részében (VEREM, STACK). Mivel alprogramokat tetszőlegesen lehet egymásba ágyazni, a VEREM speciális része a memóriának.
- Megszakítás (INTERRUPT)
  A program futását valamilyen esemény (nem programozzuk, de számítunk rá típusú) megszakítja. Ilyenkor le kell futtatni egy speciális, megszakítási alprogramot, amelynek a végén elhelyezett RETURN vissza-

térést eredményez ugyan a megszakított program következő utasításához, de a megszakítási alprogram más változásokat is okozhat, ezért a VEREM-ben való tárolás megszakítás kiszolgálásakor sokkal bonyolultabb, mint a CALL-RETURN esetén.

- Tartás, HOLD
  A HOLD állapotot egy azonos nevű bemenő-jel kezdeményezi, ha magas szintre áll. Hatására alkalmas pillanatban processzor megáll, és úgynevezett HOLD állapotba kerül. Ez azt jelenti, hogy a CPU mind az ADAT, mind a CÍM síneket elengedi (lebegteti), és ezzel DMA-ra ad lehetőséget. Ebből az állapotból kilépni a HOLD jel leejtésével lehet.
- Megállítás, HALT
  Ez olyan speciális utasítás, amelynek hatására a processzor előrehaladása megáll. Egy speciális várakozó állapotba kerül a processzor, amiből megszakítással lehet kikerülni.

4.5. Egyszerű mikroprocesszor architektúra

A mikroprocesszor architektúrájának fő elemei egyrészt az utasítások elővételének és értelmezésének, másrészt végrehajtásának a feladatait látják el (4.2. ábra).



4.2. ábra. Egy egyszerű mikroprocesszor regiszter-átviteli szintű architektúrája

Vegyük sorra először az utasítás elővétel és értelmezés fő elemeit. Az utasítás-elővétel (FETCH) a programszámlálóból (PC) indul. Maga a PC az ábra jobb oldalán látható regiszter-halmaz egyik eleme. Egy inkrementálást végrehajtó egység csatlakozik hozzá (INC/DECR). A PC tartalma az utasítás-elővétel első fázisában az ABB (cím-sín meghajtó) egységre kerül. Ezzel a cím megjelenik a memória címző bemenetén, és a memória a megcímzett memória-cella tartalmát a KÜLSŐ ADAT-SÍN-re (EDB) helyezi. Ez az adat az L/DBB adatsín tároló és meghajtó fokozaton át a BELSŐ ADAT-SÍN-re (IDB) kerül, ahonnan a processzor azt beírja az IR utasítás-regiszterbe. Az utasítás-regiszterre csatlakozó ID utasításdekóder kimenetein megjelennek azok a jel-szintek, amelyek az adott utasítás végrehajtásának logikai feltételeit biztosítják. Fontos, hogy a FETCH végrehajtásával egyidejűleg a PC inkrementálódik, tehát máris a következő utasítás címére mutat.

Az utasítások végrehajtásának legfontosabb eleme az aritmetikai-logikai egység (ALU) és a hozzá kapcsolódó speciális regiszterek. Vegyünk egy kétoperandusú aritmetikai utasítást. Ennek egyik operandusa mindig az akkumulátor (ACC) regiszterben van, már az utasítás végrehajtásának megkezdése előtt. A másik operandust be kell hozni az operandus-regiszterbe (TMP). Azt hogy azt honnan és hogyan kell ide hozni, azt a már az IR-ben lévő utasítás-kód határozza meg. Tegyük fel, hogy a szóban forgó utasítás direkt-címzéssel határozza meg a másik operandus címét. A processzornak tehát be kell hoznia ezt a címet az utasítás kódot közvetlenül követő memória-cellákból, majd ezt a címet kell az ABB-be juttatnia. Miután a memória kiadta ezt a megcímzett másik operandust, az a belső adatsínen át bejut a TMP regiszterbe. Most kezdődhet a tulajdonképpeni aritmetikai számítás. Az ACC tartalma először az ACT átmeneti regiszterbe kerül, felszabadítva ezzel a helyet az eredmény számára az ACC regiszterben. Az aritmetikai művelet eredménye az ACT és a TMP regiszterekbe érkező operandusok megjelenésétől számított műveleti időn belül megjelenik az ALU egység kimenetén. Az utolsó mozzanat, hogy az ALU kimenetének értéke megjelenik az IDB-n, és az ACC-ba kerül. Az ALU-hoz csatlakozó mutatók (FLGs) bitek az ALU-ban születő eredmények néhány fontos ismérvét mutatják.

Fontos szerepet tölt be a regiszter-tár, mind a címzésben, mind adattárolásban.

A processzor időzítő-vezérlőjének (TAC, Timing and Control) feladata, hogy az utasítás-elővételi és végrehajtási ciklusok állapotai alatt a tenni-

valóknak megfelelő vezérlő-jeleket generálja. Ezek a vezérlő-jelek lehetnek az órajel valamelyik fázisával szinkronizált impulzusok, például regiszterek beíró jelei, vagy valamely állapot valamelyik órajel-fázisára felfutó, valamely másik állapot valamelyik órajel fázisára lefutó jel, például ALU funkció beállítás. Az időzítő fogadja és generálja is azokat a vezérlő jeleket, amelyek a környezetből származnak, illetve a környezetnek szólnak. A TAC által generált vezérlő-jelek a rendszert értesítik a következőkről:

- a processzor írási memória vagy I/O műveletet hajt végre (WR),
- a processzor memóriából, vagy I/O egységből olvasását hajt végre (DBIN),
- a processzor engedélyezi a megszakítást (INTE),
- a processzor a tartó-állapotba való átmenetre vonatkozó kérést nyugtázza (HOLDA),
- a processzor várakozó állapotban van (WAIT)
- a processzor kihelyezte az adatbuszra a státusz-információt (SYNC)

Az időzítő-vezérlő fogadja a következő környezeti vezérlő-jeleket:

- megszakítás-kérés a processzorhoz (INT),
- kérés tartó-állapotba való átmenetre (HOLD)

A processzor megszüntetheti a várakozó állapotot, a memória vagy I/O elkészült (READY)

4.6. A mikroprocesszor időbeli működése

A processzorok időzítés-vezérlésének állapotai gépi ciklusokba vannak sorolva. A gépi ciklus azon állapotok halmaza, amelyek egy memóriával való kommunikációhoz kellenek Egy gépi állapot a kétfázisú, nem-átlapolt órajel első fázisának (ph1) két felfutása közötti időtartomány. Egy gépi ciklus változó számú állapotból állhat, míg utasítások végrehajtása pedig változó számú gépi ciklusból igényel. Kitüntetett gépi ciklus az első, azaz M1 ciklus, ami a FETCH, azaz az utasítás-elővétel. A legrövidebb végrehajtású utasítások az M1 alatt végre is hajtódnak A leghosszabb végrehajtású utasítások a fenti akár 4-6 gépi ciklust is igényelhetnek.

4.7. Az utasításkészlet

A mikroprocesszorok utasításkészletét alkotó utasításokat funkciójuk szerint csoportosítjuk, a következőképpen.

- adatmozgatók
- aritmetikaiak
- elágazók
- verem, I/O és vezérlők

Adatmozgató utasítások végrehajtásakor a memória-helyek, vagy a memória és a regisztertár közötti adatátvitel zajlik. Az aritmetikai utasítások végrehajtásakor operandusok közötti számítások vagy logikai műveletek mennek végbe a processzorban. Az elágazó utasítások során feltétel nélkül, vagy adott logikai feltételek fennállásakor megváltozik a memóriában levő utasítások cím szerinti sorrendben való elővétele.

4.8. Néhány utasítás végrehajtása

A processzor működésének megértése érdekében vizsgáljuk meg, milyen vezérlési szekvencia tartozik néhány utasítás elővételéhez és végrehajtásához.

4.8.1. A „MOVEr,M" (Move from Memory) utasítás végrehajtása

Az utasítás bináris kódja:	0 1 D D D 1 1 0
Az utasítás definíciója:	r(DDD) ← MEM(H_L)
Ciklusok száma:	2
Állapotok száma:	7
Címzés:	regiszter-indirekt
A jelzőbitek:	nem változnak

Az utasítás definíciója szerint a processzor a DDD regiszter-címmel kijelölt gyors-regiszterbe hozza a H_L regiszter-pár aktuális tartalmával megcímzett memória-szót. Az elővétellel együtt két gépi ciklus kell a végrehajtáshoz, és összesen 7 gépi állapot, azaz órajel ciklus. A címzés az indirekt címzésnek egy meghatározott alfaja, amikor a memória-cím egy regiszterpárban található. Az utasítás végrehajtása nem változtatja meg a processzor jelzőbitjeit. A végrehajtás állapotonként:

```
M1, T1:        AB ← PC, EDB ← státuszinformáció
M1, T2:        PC ← PC + 1
```

```
M1, T3:        IR ← EDB
M1, T4:
M2, T1:        ABB ← H_L, EDB ← státuszinformáció
M2, T2:        IDB ← EDB
M2, T3:        r(DDD) ← IDB
```

4.8.2. Az „ADD M” (Add Memory) utasítás végrehajtása

Az utasítás bináris kódja:	1 0 0 0 0 1 1 0
Az utasítás definíciója:	ACC ← MEM(H_L) + ACC
Ciklusok száma:	2
Állapotok száma:	7
Címzés:	regiszter-indirekt
A jelzőbitek:	változnak

A végrehajtás állapotonként:

```
M1, T1:        AB ← PC, EDB ← státuszinformáció

M1, T2:        PC ← PC + 1
M1, T3:        IR ← EDB
M1, T4:        ACT ← ACC
M2, T1:        AB ← H_L, EDB ← státuszinformáció
M2, T2:        IDB ← EDB
M2, T3:        TEMP ← IDB
M1, T1:        IDB ← ACT + TMP
M1, T2:        ACC ← IDB
```

Vegyük észre, hogy az ADD M utasítás végrehajtása átlapolódik a következő utasítás elővételével. Ezt a működés gyorsítására más utasítás végrehajtásánál is alkalmazzák.

4.8.3. A „LDA” (Load Accumulator) utasítás végrehajtása

Az utasítás kódja:	0 0 1 1 1 0 1 0
	laddr
	haddr
Az utasítás definíciója:	ACC ← MEM(haddr_laddr)
Ciklusok száma:	4
Állapotok száma:	13
Címzés:	direkt
A jelzőbitek:	nem változnak

A végrehajtás állapotonként:

```
M1, T1:        AB ← PC, EDB ← státuszinformáció
M1, T2:        PC ← PC + 1
M1, T3:        IR ← EDB
M1, T4:
M2, T1:        AB ← PC, EDB ← státuszinformáció
M2, T2:        PC ← PC + 1
M2, T3:        Z ← MEM(AB)
M3, T1:        AB ← PC, EDB ← státuszinformáció

M3, T2:        PC ← PC + 1
M3, T3:        W ← MEM(AB)
M4, T1:        AB ← W_Z, EDB ← státuszinformáció
M4, T2:        IDB ← EDB
M4, T3:        ACC ← IDB
```

4.8.4. A „CALL” (Call, azaz alprogram hívás) utasítás végrehajtása

Az utasítás kódja:	1 1 0 0 1 1 0 1
	laddr
	haddr
Az utasítás definíciója:	MEM(SP-1) ← h_PC
	MEM(SP-2) ← l_PC
	SP ← SP − 2
	PC ← haddr_laddr
Ciklusok száma:	5
Állapotok száma:	17
Címzés:	közvetlen
A jelzőbitek:	nem változnak

Az utasítás végrehajtása állapotonként:

```
M1, T1:        AB ← PC, EDB ← státuszinformáció
M1, T2:        PC ← PC + 1
M1, T3:        IR ← EDB
M1, T4:
M1, T5:        SP ←SP-1
M2, T1:        AB ← PC, EBD ← státuszinformáció
M2, T2:        PC ← PC + 1
M2, T3:        Z ← MEM(AB)
M3, T1:        AB ← PC, EDB ← státuszinformáció
M3, T2:        PC ← PC + 1
```

```
M3, T3:        W ←      MEM(AB)
M4, T1:        AB← SP, EDB ← státuszinformáció
M4, T2:        SP ← SP -1

M4, T3:        EDB ← hPC
M5, T1:        AB← SP, EDB ← státuszinformáció
M5, T2:        SP ← SP -1

M5, T3:        EDB ← lPC
M1, T1:        AB ← W_Z
M1, T2:        PC ← W_Z + 1
```

4.9. A „RETURN” (Return, azaz visszatérés az alprogramból) utasítás végrehajtása

Az utasítás kódja: 1 1 0 0 1 0 0 1
Az utasítás definíciója: lPC ← MEM(SP)
hPC ← MEM(SP+1)
SP ← SP + 2
Ciklusok száma: 3
Állapotok száma: 10
Címzés: regiszter-indirekt
A jelzőbitek: nem változnak

A végrehajtás, állapotonként:

```
M1, T1:        AB ← PC, EDB ← státuszinformáció
M1, T2:        PC ← PC + 1
M1, T3:        IR ← EDB
M1, T4:
M2, T1:        AB ← SP, EDB ← státuszinformáció
M2, T2:        SP ← SP + 1
M2, T3:        Z ← MEM(AB)
M3, T1:        AB ← SP, EDB ← státuszinformáció
M3, T2:        SP ← SP + 1
M3, T3:        W ← MEM(AB)

M1, T1:        AB ← W_Z
M1, T2:        PC ← W_Z + 1
```

4.10. A mikroprocesszor működésének egyéb sajátosságai

4.10.1. A READY-WAIT jelpáros

Az (Mi,T2) és az (Mi,T3) állapotok között általában memória írás vagy olvasás történik. Ha a memória nem olyan gyors, hogy a számára előírt műveletet el egy állapot alatt lehessen végezni, egy (Mi,Tw) várakozó állapotot kell beiktatni. Ez egy READY-WAIT jelpáros segítségével valósul meg. Amíg a memória nem jelzi a READY segítségével, hogy elkészült, a processzor fenntartja a várakozó állapotot, és a WAIT jelet.

4.10.2. A státusz-információ

A processzor minden gépi ciklus T2 állapotában kiadja a környezetének a ciklusra vonatkozó információt. Ezek mint kódolt bináris vektorok az EDB-n jelennek meg, és a SYNC lefutó élével eltárolhatók.
Néhány jellemző státusz:

UTASÍTÁS ELŐVÉTEL,
MEMÓRIA OLVASÁS,
MEMÓRIA ÍRÁS,
VEREM OLVASÁS,
VEREM ÍRÁS

A státuszinformációt rendszer-vezérlésre használjuk.

4.10.3. A jelzőbitek

Z: 1, ha az ALU-ban képződött eredmény 0,
CY: 1, ha van átvitel a legnagyobb bináris helyiértékről,
S: az ALU-ban képződött eredmény előjele

4.10.4. Az SP értékének beállítása

A veremmutató beállítására speciális utasítást definiáltak. Ez az SPHL utasítás, amely a H_L regisztertár tartalmát betölti a verem-mutatóba.

SP ← H_L

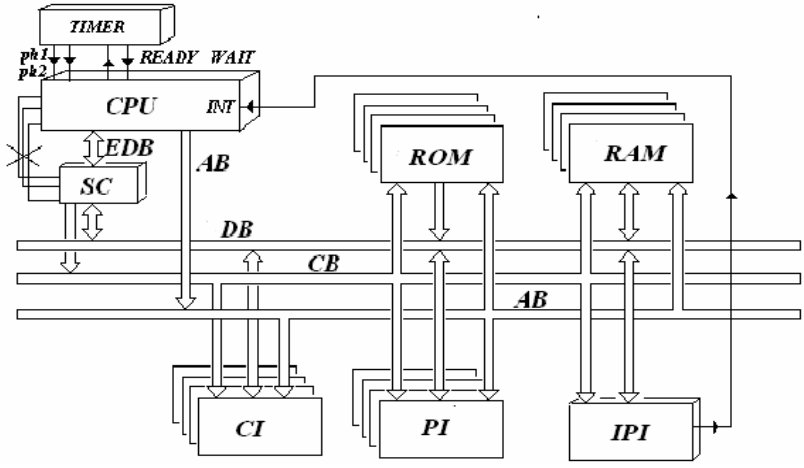
4.10.5. A megszakítások kezelése

Az EI utasítás engedélyezi a megszakítást. Végrehajtás: a belső INTFF (Interrupt flip-flop) 1-be áll, és INTE kimenő jel magasra emelkedik. Ha az INT a külső vezérlő-jelen megérkezik, a következő FETCH ciklus speciális utasítást fogad. Ez nem a memóriából jön, hanem a megszakítást

kérő egységnek kell az EDB-re tennie. Neve RSTn Fontos, hogy a folyamatban lévő utasítást befejezzük, mielőtt az RST n-t elindítjuk. Az RSTn utasítás kódja: 1 1 N N N 1 1 1

4.11. A mikroprocesszoros rendszer

Egy mikroprocesszoros rendszer magán a processzoron kívül természetesen számos más rendszer-elemet is tartalmaz. A 4.3. ábrán bemutatott rendszer a memória modulokat és a processzort kiszolgáló egyéb modulokból áll. A ROM (Read Only Memory), azaz csak olvasható memória) modulok olyan memória áramkörök, amelyekbe adatot írni nem lehet, az azokban rögzített adatok azonban ugyanolyan címzési és vezérlési eljárásokkal olvashatók ki, mint az írható-olvasható memóriák esetében. Az írható olvasható memóriamodulok a RAM (Random Access Memory) egységben foglalnak helyet. Ez az elnevezés nem a funkcióra utal, hanem arra, hogy az adatok közül bármelyiket közel azonos idő alatt érhetjük el. A memóriákkal való kommunikációt a TIMER-egység azzal segíti, hogy a lassú memóriától származó késlekedő READY jelet szinkronizálja a processzor számára, és fogadja a WAIT jelet. Az SC (System Controller, rendszer-vezérlő) a SYNC jellel szinkronban eltárolja a státusz információt, és ennek segítségével megfelelő módon vezérli a processzort, de a rendszer CB (Control Bus) vezérlő-sínjét is. A CB összeköti a processzor külső adat-sínjét a rendszer adat-sínnel (DB). A rendszert az I/O kommunikációt segítő egységek egészítik ki. A kommunikáció több, szabvá-

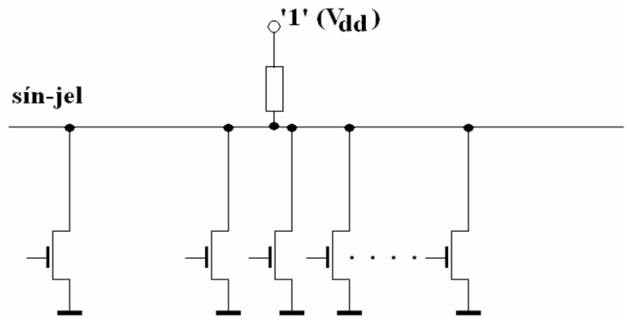


4.3. ábra. Mikroprocesszoros rendszer

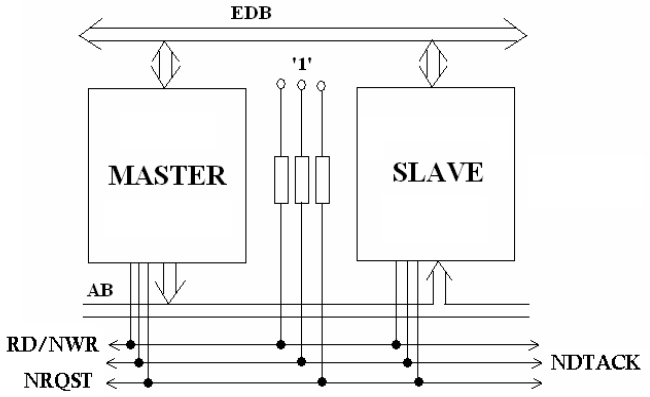
nyos rendszerét segítő kommunikációs interfész (CI), a perifériákkal való kommunikációt vezérlő periféria interfész (PI), és a megszakítás-kéréseket prioritási sorrendbe állító IPI (Interrupt Priority Inteface) modul.

4.12. Kommunikáció mikroprocesszoros rendszerelemek között

A több CPU-t, illetve más intelligens egységet is tartalmazó mikroprocesszoros rendszer elemei között adatok átvitele zajlik. Az adatátvitelt kezdeményező egység a MASTER, a passzív fél pedig a SLAVE egység. Általában egy CPU a MASTER, és több potenciális SLAVE kapcsolódik a MASTER-hez. Ahogyan az adat- és a cím-sín is közös és egyetlen a rendszerben, hasonló megoldás alakult ki a vezérlő jelek sínjére is. A probléma az, hogyan használhatja valamennyi egység ugyanazt a vezérlő-vonalat. A megoldást a 4.4. ábra mutatja. Egy kiragadott vezérlő-vezeték valamennyi egységhez csatlakozik, de a kommunikációs folyamat megindulása előtt minden egység lebegteti azt. A vonal ellenállásokon keresztül a pozitív tápfeszültségre van kötve. Így alapállapotban egy „gyenge” 1-es szint van a vonalon. Ezután a kommunikációban aktív egység a vonalra csatlakozó MOSFET kapcsolója segítségével 0 szintet kényszeríthet a vonalra, amelyet a többiek érzékelnek. A 4.5. ábrán azt látjuk, milyen vezérlő-jelek szükségesek egy MASTER és egy általa kijelölt SLAVE közötti adatátvitel lebonyolításához. A SLAVE a címsínen felismeri a saját címét, ezzel tudomásul veszi a kijelölést. Az adat-sín szolgál mind az írott, mind az olvasott adat átvitelére. A MASTER a RD/NWR jelen kódolja, hogy olvasni vagy írni akar. A kommunikáció megindítása a MASTER feladata a negatív logikájú NRQST (REQUEST) jel 0-ra való lehúzásával. A kijelölt SLAVE a NDTACK jel lehúzásával jelzi, hogy eleget tett a MASTER felszólításának.

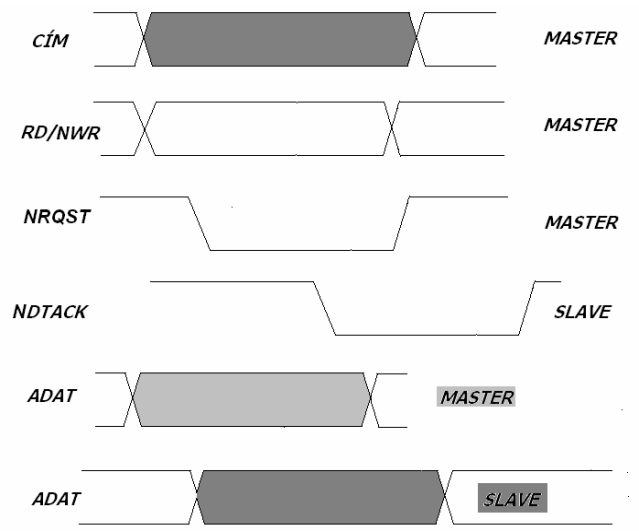


4.4. ábra. negatív logikájú vezérlő-vezeték



4.5. ábra. A MASTER és egy SLAVE kapcsolata

Kövessük most a kommunikáció időbeli lefolyását a 4.6. ábra segítségével. A MASTER, amennyiben írást kezdeményez, a kijelölni kívánt SLAVE címét, az átvinni kívánt adatot elhelyezi az AB és az EDB síneken. Ezután lerántja az NRQST jelet. Ezt valamennyi egység érzékeli, és mindenki megvizsgálja az AB-n lévő címet. A kijelölt SLAVE felismeri a címét, és tudomásul veszi azt is, hogy az EDB-n lévő adatot be kell olvasnia. Ezt megteszi, majd az NDTACK vonal lerántásával jelzi, hogy a feladatot teljesítette. Ezután a MASTER elengedi az NRQST vonalat, amire a SLAVE az NDTACK vonal elengedésével válaszol.



4.6. ábra. Írás és olvasás a MASTER kezdeményezésére

Vizsgáljuk most az olvasás folyamatát. Ilyenkor a MASTER az EDB-t lebegteti, hiszen azt majd a SLAVE-nak kell meghajtania. A MASTER most is az NRQST-vel kezdi a folyamatot, de most az RD/NWR jel magasan marad. A SLAVE a cím és a szándék felismerése után kiteszi adatát az EDB-re, és az NDTACK jelet lerántja. Erre a MASTER, miután az EDB-n lévő adatot beolvasta, elengedi az NRQST-t, amire a SLAVE az NDTACK elengedésével válaszol. Ezt a kommunikációs folyamatot HAND-SHAKE (kézfogásos) adatátvitelnek nevezik.

4.13. A mikroprocesszoros rendszerek ASSEMBLY szintű programozása

Egy igen egyszerű példával illusztráljuk a gépi kódhoz igen közel álló ASSEMBLY nyelvű programozás sajátosságait. Tegyük fel, hogy egy címlista elemei a memóriában szétszórt adatok címeit tartalmazza. A címlista hossza nincs meghatározva, ezért a végét egy speciális cím jelzi, legyen ez az FFFFH, azaz a maximális 4-jegyű hexadecimális sorszám. A program feladata, hogy a címlista elemeit egyenként beolvassa, megvizsgálja, hogy nem a lista-végét jelző cím-e, és ha nem, akkor hozza be a memóriából a listaelemmel megcímzett adatot, és adja hozzá a már kialakult részösszeghez. Az olvasó feladata, hogy a mellékelt utasítás-definíciók alapján értelmezze az ASSEMBLY nyelvű programot.

Címke	Mnem.	Paraméter	Megjegyzés
	LXI	H, LIST	A lista kezdőcíme a H_L be
	CALL	SUMMA	rutin hívása
SUMMA:	XRA	A	Az ACC törlése
LOOP:	MOV	C, A	Az ACC áttöltése a C regiszterbe
	MOV	E, M	Az első adat címének alsó része az E-be kerül (H_L címzés)
	INX	H	A H_L növelése
	MOV	A, M	Az első adat címének felső része ACC-ba kerül
	CPI	FF	Ha ACC tartalma FF, Z nulla lesz
	JZ	BACK	Ha vége, ugrás a BACK címkére
	MOV	D, A	Ha nincs vége, cím alsó fele D-be
	LDAX	D	adatbyte a D_E ben megadott címről az ACC-ba

```
        ADD   C       ACC hozzáadása C-hez
        JC    OVER    Ha CY=1, túlcsordulás, ugrás OVER-re
        INX   H       H_L növelése
        JMP   LOOP    Visszaugrás
BACK:   MOV   A, C    Összeg vissza az ACC-ba
        RET           Visszatérés
OVER:   NOP           A túlcsordulás lekezelésének kezdete
```

A még nem ismert utasítások definíciói:

`LXI: h r ← byte3, l r ← byte2`
(az utasításkód utáni két byte az r-regiszterpárba kerül)

`XRA: ACC ← A (XOR) r`
(az ACC és a megcímzett regiszter XOR eredménye az ACC-ba kerül)

`INX: h r_l r ← h r_l r + 1`
(az r regiszterpár tartalma inkrementálódik)

`CPI: ACC – byte2`
(az ACC tertalmából az ALU kivonja az utasítás utáni byte-t, és beállítja a jelzőbiteket)

`JZ:` ha Z = '1', akkor `PC ← byte3_byte2`
(ugrás, ha Z = '1')

`LDAX: ACC ← MEM(h r_l r)`
(az ACC-t betöltjük a megcímzett regiszterpárban levő memóriacímről)

`JC:` ha CY = '1', akkor `PC ← byte3_byte2`
(ugrás, ha a CY jelzőbit = '1')

`JMP: PC ← byte3_byte2`
(ugrás feltétel nélkül)

`NOP:`
(No-operation, azaz üres utasítás)