SZÉCHENYI ISTVÁN EGYETEM

DUÁLIS KÉPZÉS

Somogyi Miklós

DIGITÁLIS HÁLÓZATOK

A tantárgy célja:

a kapu szintű digitális hálózatok tervezési elveinek bemutatása és az elvek gyakorlati alkalmazásának elsajátítatása tervezési feladatok megoldásával. A tantárgy alapozó és elengedhetetlen ismereteket nyújt a műszaki informatikai, mechatronikai mérnöki és villamosmérnöki szakirányú tárgyak elsajátításához, továbbá elősegíti bizonyos problémák mérnöki megközelítését, a mérnöki problémamegoldási készség fejlesztését.

(A jegyzet megírásának alapjául szolgált „Dr. Keresztes Péter: Digitális hálózatok" című HEFOP (szabad forrású elektronikus) jegyzete. A hivatkozások (ábrák, rajzok, szövegrészek) a szerző hozzájárulásával és engedélyével kerültek felhasználásra.)

1. Bevezetés

A napjainkban használatos elektronikus áramkörök jelfeldolgozási rendszerét tekintve alapvetően kétféle hálózattípust különböztethetünk meg:

- *analóg hálózatok*
- *digitális hálózatok*

Analóg hálózatok esetében mind a bemeneti, mind a kimeneti jelek valamilyen folytonos változóként határozhatók meg (pl. feszültség, áram, frekvencia stb.), melyek -egy reális értéktartományban- elvileg bármilyen nagyságú értéket felvehetnek.

Digitális hálózatok esetében ezek a jelek diszkrét változókként értelmezhetőek, melyek csupán két lehetséges értéket vehetnek fel: ez a két érték a „0” és az „1”. Ezeket, mint ún. logikai értékeket értelmezhetjük, illetve magukat a bemeneti és kimeneti jeleket, mint logikai változókat definiálhatjuk.

Az elektronikus áramkörök működése esetén a környezetből vett jelek, mint primer bemenő jelek lehetnek mind analóg, mind digitális jellegűek. Azonban maga a központi jelfeldolgozás a modern elektronikus architektúrákban mind hardver, mind pedig szoftver szinten digitális formában történik. Ez azt jelenti, hogy pl. az analóg bemenő jeleket digitalizálni kell, vagyis alkalmazni szükséges egy ún. analóg-digitális átalakítót, mely valamilyen „leképezési”, kódolási módszerrel az analóg jellegű jeleket digitális formában továbbítja a központi feldolgozó egység felé. Az áramkörök kimeneti jelei – hasonlóan a bemeneten előforduló jellemzőkhöz- szintén lehetnek analóg vagy digitális jellegűek. Ennek megfelelően pl. a szükséges analóg kimeneti jelek előállítása esetén itt szintén alkalmazni kell egy átalakító egységet, ami ebben az esetben a központi egység által előállított digitális jeleket a kívánt analóg kimenő jelekké alakítja. Ezek az ún. digitális-analóg átalakítók. Ezek alkalmazásának szükségességét mindig az adott működési specifikáció határozza meg.

A továbbiakban megvizsgáljuk, hogy milyen matematikai modell segítségével lehet leírni a digitális hálózatok működését és annak jellegzetességeit, valamint hogyan lehet ezeket a hálózatokat ún. kapu szintű áramköri realizációval megadni.

2. Digitális hálózatok matematikai leíró modellje: a Boole-algebra

2.1 A bemenő és kimenő jelek, mint logikai változók

A digitális hálózatok működésüket tekintve ún. „kétértékes" leíró jelekkel definiálható hálózatok. Ez a két érték – a megfigyelt objektumot vagy mintavett jelértéket tekintve többféle megközelítéssel is jellemezhető. Egy adott jelhez tartozó kétféle jellemző lehet pl. a következő állítás:

- *igaz-hamis (pl. logikai állítás)*
- *van-nincs (pl. esemény)*
- *bekapcsolt-kikapcsolt állapot (pl. kapcsoló), stb.*

A példákban felsorolt jelértékeket, mint változókat tekinthetjük a Boole-algebrát is meghatározó jellemezőknek: ún. logikai (boolean) változóknak. A digitális hálózatok működését tehát a Boole-algebrával jellemezhető matematikai modell segítségével lehet egyértelműen megadni illetve leírni. A logikai változók két lehetséges értéke ebben az értelemben a „true" és a „false". Ezek az értékek digitális áramköri realizációt tekintve lényegében hozzárendelt feszültségszintekkel jellemezhetőek. Az egyes jelekhez rendelt vezetékek alacsony, nullához közeli feszültségszintje jelenti az adott jel alacsony („low" = „0") logikai értékét, míg egy néhány volt értékű feszültségszint pedig a magas („high" = „1") logikai értéket. Általában (és alapesetekben) ezt tekintjük az ún. „pozitív" logikának. (Ennek épp a fordítottja a „negatív" logika, de a köznapi értelemben, illetve ezen jegyzet későbbi fejezeteiben is általában a pozitív logikát tekintjük alapértelmezettként.)

2.2 A Boole- algebra logikai alapműveletei

A Boole-algebrában a logikai értékek között három logikai alapműveletet definiálható:

- '**ÉS**' művelet ('•')
- '**VAGY**' művelet ('**+**')
- **negáció** (tagadás) művelete ('$\overline{\ }$')

A logikai szorzás vagy más néven konjunkció (’**ÉS**’) egy szorzás műveleti jellel (’•’), a logikai összeadás vagy diszjunkció, (’**VAGY**’) az összeadás műveleti jelével (’**+**’) adható meg, s ezek két operandusú, úgynevezett bináris műveletek. A harmadik művelet a logikai tagadás, vagy másnéven **negáció** (’¯’ = felül-vonás), mely csak egy operandusú, úgynevezett unáris művelet.

A fent ismertetett műveleteket kétértékes logikai rendszerekben a következők szerint lehet értelmezni:

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

2.3 A Boole-algebra logikai azonosságai

A logikai algebrában érvényes azonosságok olyan axiómák, 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.

Asszociativitás:
(A + B) + C = A + (B + C)
(A • B) • C = A • (B • C)

Kommutativitás:
A • B = B • A
A + B = B + A

Disztributivitás:
A • (B + C) = A • B + A • C
A + (B • C) = (A + B) • (A + C)

Ellentmondás törvénye:
A • $\bar{A}$ = 0

Elnyelési törvények 1.
A + (A • B) = A
A • (A + B) = A

Elnyelési törvények 2.

$$\mathbf{A + (\bar{A} \bullet B) = A + B}$$
$$\mathbf{A \bullet (\bar{A} + B) = A \bullet B}$$

Elnyelési törvények 3.

$$\mathbf{A + 1 = 1}$$
$$\mathbf{A \bullet 0 = 0}$$

Harmadik kizárásának törvénye:

$$\mathbf{A + \bar{A} = 1}$$

Identitások törvénye:

$$\mathbf{A + 0 = A}$$
$$\mathbf{A \bullet 1 = A}$$

Idempotencia törvénye:

$$\mathbf{A + A = A}$$
$$\mathbf{A \bullet A = A}$$

Involúció törvénye:

$$\mathbf{\bar{\bar{A}} = A}$$

De-Morgan azonosságok:

$$\mathbf{\overline{A + B} = \bar{A} \bullet \bar{B}}$$
$$\mathbf{\overline{A \bullet B} = \bar{A} + \bar{B}}$$

Példa: értelmezzük a táblázat segítségével a következő azonosságot:

$$\mathbf{A \bullet 1 = A}$$

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

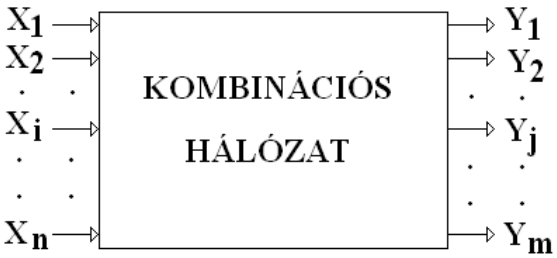
Különös jelentőséggel bír a dualitás elvét magában foglaló két De-Morgan azonosság. Ezek közül a másodikat értelmezzük a kapcsolók körében. Legyen „**A**” és „**B**” két kapcsoló. A kettő „ÉS” kapcsolata egy sorosan kapcsolt kapcsolópárt reprezentál, amely csak akkor vezethet, ha mind az „**A**”, mind 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.

A De-Morgan azonosságokat felhasználva a digitális hálózatok működését leíró függvények átalakításával - egy szükséges specifikáció szerint - bármely típusú logikai kapuszintű realizáció felépíthető a funkcionális teljességet biztosító alapelemekből (NAND és NOR kapuk).

3. Kombinációs hálózatok tervezése

3.1 A kombinációs hálózatok „fekete-doboz” modellje

A kombinációs hálózatok általános felépítését a 3.1.a. ábrán szereplő ún. „fekete-doboz” modellel lehet szemléltetni:



3.1.a. ábra [1]

Kombinációs hálózatok „fekete-doboz” modellje

Egy kombinációs hálózatnak *bemenetei* és *kimenetei* vannak, valamennyi egy *logikai változó*, illetve logikai jel, és ennek megfelelően mindegyik csak a „0” vagy az „1” logikai értéket veheti fel. Ez a kombinációs hálózat fekete-doboz modelljének egyik lényeges tulajdonsága. A bemeneteket az $X_1, X_2, \ldots X_i, \ldots X_n$, a kimeneteket $Y_1, Y_2, \ldots Y_j, \ldots Y_m$ szimbólumokkal jelöltük.
A kombinációs hálózat a bemeneti jelek felett értelmezett bemeneti érték-variációkhoz a kimeneti jelek érték-variációit rendeli. (Minden bemeneti kombinációhoz legfeljebb csak egy kimeneti kombinációt.)

Példa: három bemenet esetében ($n = 3$) a lehetséges bemeneti variációk halmaza

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

amihez négy darab kimeneti változó esetében ($m = 4$) a következő a lehetséges variációk halmaza

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

A példában szereplő kombinációs hálózat egy lehetséges működési specifikációját a 3.1.b ábrán szereplő táblázatban foglaltuk össze.

Bemeneti variációk			Kimeneti variációk			
X_1	X_2	X_3	Y_1	Y_2	Y_3	Y_4
0	0	0	1	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	1	1
0	1	1	1	0	1	1
1	0	0	1	0	0	1
1	0	1	0	0	0	0
1	1	0	1	1	0	1
1	1	1	0	1	0	0

3.1.b. ábra

Egy lehetséges be-és kimeneti specifikáció

3.2 Kombinációs hálózatok specifikációs mélysége

A kombinációs hálózatok specifikációs mélységük szerint két csoportba oszthatóak:

- *teljesen specifikált kombinációs hálózatok*
- *nem teljesen specifikált kombinációs hálózatok*

Egy teljesen specifikált kombinációs hálózat esetében minden bemeneti variációhoz olyan kimeneti variáció tartozik, amelyben minden kimenet értéke specifikálva van.
Nem teljesen specifikált kombinációs hálózatokról akkor beszélünk, ha van legalább egy olyan bemeneti variáció, amelyhez rendelt kimeneti variációkban legalább egy változó értéke közömbös („do not care”).

A fenti meghatározás alapján megállapíthatjuk, hogy a 3.1.b. ábrán szereplő specifikáció egy teljesen specifikált kombinációs hálózat viselkedését írja le.

Általánosságban megállapítható, hogy ha „*n*” számú 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 érték-variációból ismétléssel ki tudunk választani egy 2^n hosszúságú sorozatot.

Ebből következik, hogy összesen $(2m)^{2^n}$ számú „*n*” bemenettel és „*m*” kimenettel rendelkező teljesen specifikált kombinációs hálózat létezik.

3.3 Kombinációs hálózatok viselkedésének logikai függvényekkel történő leírása

3.3.1 Logikai függvények és megadási módjaik

Egy 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”. Amennyiben minden bemeneti variációra létezik „0” vagy „1” logikai értékű kimeneti variáció, akkor az adott hálózat teljesen specifikált. Ha azonban van olyan bemeneti variáció, amelyre nincs a kimeneten határozott logikai értéket megadó előírás (sem „1”, sem „0”), vagyis „közömbös”, azaz ’don’t-care’(’-’), akkor a hálózat nem teljesen specifikált. A kombinációs hálózatok minden egyes kimenetéhez egy „*n*” változós logikai függvény tartozik.

A kombinációs hálózatok működését reprezentáló logikai függvények megadása többféle módon lehetséges:

- *igazságtáblázattal*
- *algebrai kifejezéssel*
- *elvi (grafikus) logikai vázlattal*

3.3.1.1 Logikai függvények megadása igazságtáblázattal

Egy logikai függvény igazságtáblázattal úgy adható meg, hogy minden bemeneti variációt felsorolunk, és megadjuk a hozzájuk rendelt függvényértéket, azaz az („1”, „0”, ’-’) hármas valamelyikét. Az ilyen táblázatot a függvény igazság-táblázatának nevezzük.

Példa: egy három bemenettel (*A,B,C*) és egy kimenettel (*F*) rendelkező teljesen specifikált kombinációs hálózat igazságtáblázata a 3.3.1.1. a. ábrán látható:

Bemeneti variációk			Kimeneti érték
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

3.3.1.1.a ábra

Egy három bemenetű(A,B,C) és egy kimenetű(F) kombinációs hálózat igazságtáblázata

3.3.1.2 Logikai függvények megadása algebrai alakkal

Logikai függvények algebrai alakját az igazságtáblázatból lehet kiolvasni. 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.

Példa: írjuk fel a 3.3.1.1. a. ábrán definiált 3 változós függvény algebrai alakját!

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

(Megjegyezés: a logikai-algebrában, akár csak a klasszikusban, a szorzás jelét felesleges leírni a szorzandók közé, ezért az ilyen függvényalakokban a továbbiakban ezt el is hagyjuk.)

Az így megadott függvényalakot mintermes kanonikus normál függvényalaknak nevezzük.

Mintermes kanonikus normál alak: 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.

Maxtermes kanonikus normál alak: azokat a logikai összegeket (termeket), amelyekben a függvény valamennyi változója szerepel, maxtermeknek nevezzük. A logikai függvény megadásának ezt a módját, azaz azon maxtermek szorzatát, amelyekhez a függvény „0”-át rendel, maxtermes kanonikus normál alaknak nevezzük.

A fenti példában megadott hálózat maxtermes függvényalakja így a következő:

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

A legtöbbször -általános felírási módként- a mintermes formulával találkozhatunk, ezért a továbbiakban mi is ezt a függvénymegadási módot alkalmazzuk.

A De-Morgan azonosság alkalmazása a kanonikus alakok transzformációjára

Az előző példában elsőként felírtuk a teljesen határozott logikai függvény mintermes kanonikus normál alakját. A maxtermes kanonikus alakot a mintermes alakból kétszeri tagadással, kétszeres negálással is származtathatjuk. A már ismert példánkon bemutatjuk, hogy a De-Morgan azonosság alkalmazásával hogyan kapjuk a másik kanonikus alakot.

A mintermekkel, azaz logikai szorzatok összegével adott eredeti függvény negáltja felírható olyan logikai összegek szorzataként, amelyekben ugyancsak minden változó szerepel. Ezeket az összegeket neveztük maxtermeknek.

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

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

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

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

A mintermekkel, azaz logikai szorzatok összegével adott eredeti függvény negáltja felírható olyan logikai összegek szorzataként, amelyekben ugyancsak minden változó szerepel. Ezeket az összegeket neveztük maxtermeknek.

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

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

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

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

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

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

3.3.1.3 Logikai függvények megadása kapuszintű (elvi) logikai vázlattal

A korábbiakban megismert kanonikus függvényalakok grafikus, elvi logikai vázlattal is reprezentálhatók. Ehhez egységes szimbólumrendszerek is tartoznak, melyek elemeinek segítségével ún. kapuszintű realizáción keresztül is szemléltethetjük egy adott áramkör működési tulajdonságait.

Alapvetően kétféle típusú jelölésrendszerrel találkozhatunk a logikai vázlattal megadott áramköri struktúráknál. Az egyik az ún. „európai" jelölésrendszer, a másik pedig az „amerikai", vagy hivatalos (nemzetközi) nevén IEEE jelölés. (A jegyzetben mindkettő előfordul, mert a segédletben szereplő példák megvalósítási rajzai a gyakorlatokhoz tartozó DHLAB szimulációs programból valók.)

Mindkét jelölésrendszerben alapelv, hogy a logikai műveleteket logikai szimbólumok, a grafikus elvi vázlatokban ún. logikai kapuk (továbbiakban csak röviden kapuk) reprezentálják.

A logikai teljesség mindenkori kifejezhetőségének biztosítása szempontjából alapvető megfeleltetési szimbólumok elnevezései a következők:

- *negáció* → INVERTER kapu („INV"*****)
- *szorzás* → ÉS kapu („AND" vagy „&")
- *összegzés* → VAGY kapu („OR"vagy „1"******)

valamint az utóbbi kettő negáltjai:

- *negált szorzat* → NEM-ÉS kapu („NAND")
- *negált összeg* → NEM-VAGY kapu („NOR").

Továbbá gyakrabban alkalmazott logikai művelet még az antivalencia, és szimbóluma:

- *antivalencia* → KIZÁRÓ VAGY kapu („EXOR" vagy „XOR")

és ennek tagadása, az ekvivalencia

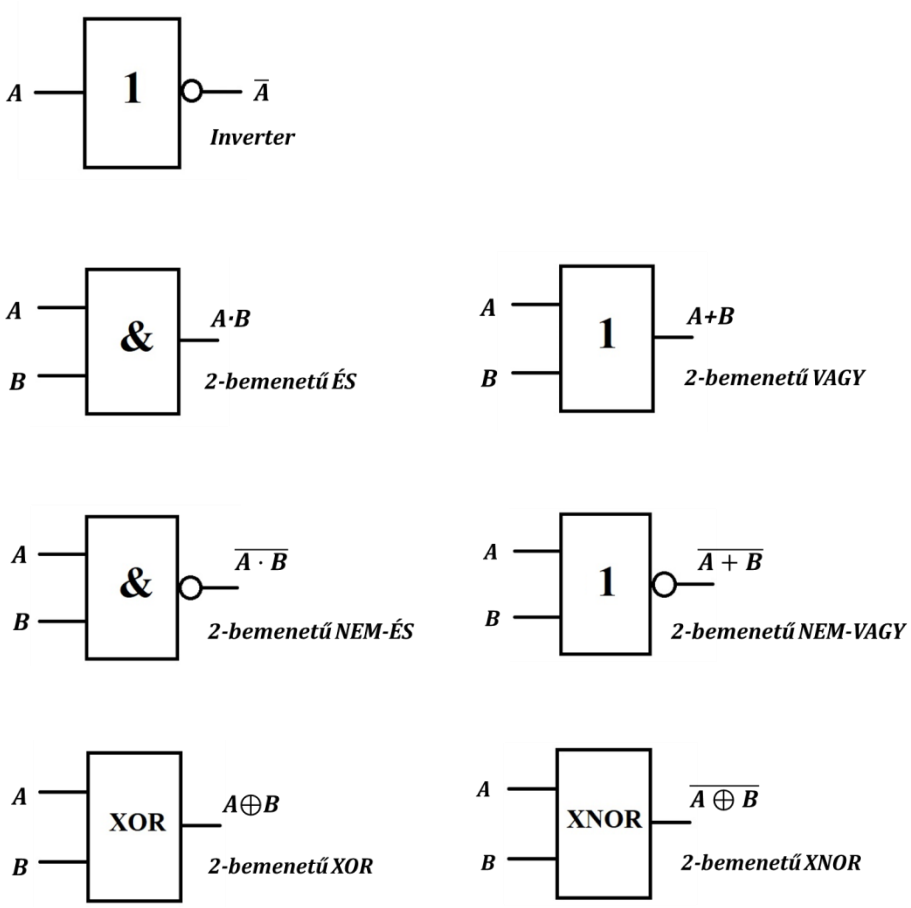
- *ekvivalencia* → KIZÁRÓ NEM-VAGY kapu („EXNOR" vagy „XNOR").

(*Megjegyzés*:

***** inverterek esetében szokásos még az „1" jelölés alkalmazása is a szimbólumban

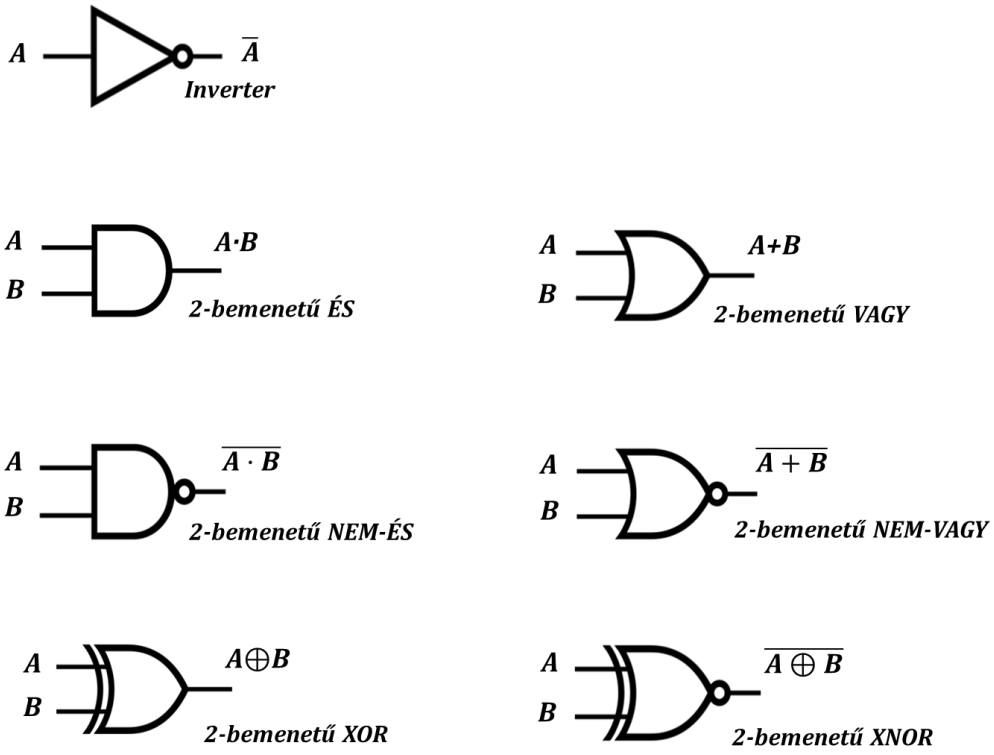
****** VAGY kapuk szimbólumában szerepelhet a „+" jelölés is)

A „negált eredmény” műveletek (INV, NAND, NOR) esetében – amennyiben a szimbólumban található jelölés: INV=”1”, NAND=”&”, NOR=”1” - az „eredmény” oldalán egy „o” szimbólum jelzi a kimeneti vezetéken magának az eredménynek a tagadását. Néhány alapvető, leggyakrabban használt szimbólum a 3.3.1.3 a és 3.3.1.3.b ábrákon látható:



3.3.1.3 a ábra

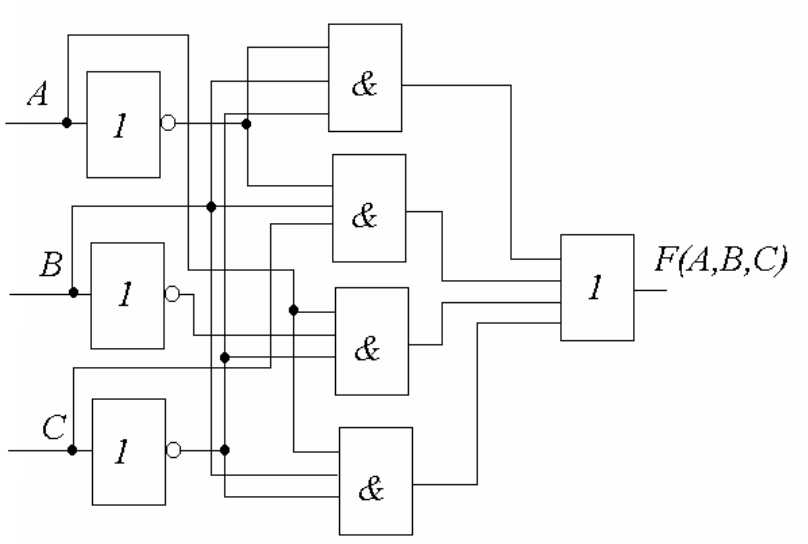
Grafikus logikai kapu-szimbólumok (európai szabvány)



3.3.1.3.b ábra

Kapu-szimbólumok az IEEE szabvány szerint

Példa: a 3.3.1.1.a ábrán látható igazságtáblából származtatott mintermes kanonikus függvényalak logikai vázlattal történő megadására látunk példát a 3.3.1.3.b ábrán:



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

3.3.1.3.b ábra[1]

Logikai hálózat megadása grafikus szimbólumokkal

3.3.2 Nevezetes kétváltozós logikai függvények

Kétváltozós (X_1, X_2) logikai függvények esetében léteznek kitüntetett, más néven nevezetes kétváltozós logikai függvények, melyek közül többet gyakran használunk logikai hálózatok építése során, mint kapu-áramköröket. Ezek értelmezéséhez nyújt segítséget a 3.3.2.a ábrán látható táblázat.

BEM. VÁLT.		FÜGGVÉNYÉRTÉKEK															
X_1	X_2	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
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**

3.3.2.a ábra

Az összes lehetséges kétváltozós logikai függvény (F_0 - F_{15})

A táblázatban szereplő függvények közül nevezetes kétváltozós logikai függvényeknek tekintik a következőket:

- **'0'** (F_0) **és '1'**(F_{15}) **generátor:**

  Az F_0 értéke állandóan '0', nem érzékeny a bemenetekre. Negáltja az F_{15}, amelynek értéke állandóan '1' .

- **'ÉS'**(F_1) **és 'NEM-ÉS'**(F_{14}) **függvény, illetve kapu:**

  Az F_1 és az F_{14} értékei éppen egymás negáltjai. Az F_1 nevezetes, csak akkor szolgáltat '1'-et, ha mindkét bemeneti változó értéke '1', az F_{14} pedig ekkor éppen '0'-t szolgáltat. Az előbbi neve 'ÉS', az utóbbié 'NEM-ÉS', röviden 'NÉS'.

- **'VAGY'(F_7) és 'NEM-VAGY'(F_8) függvény, illetve kapu:**

  Az F_7 és az F_8 függvények ugyancsak egymás negáltjai. Az F_7 értéke '1', ha legalább ez egyik bemenet '1'. Az F_8 értéke éppen ilyenkor '0'. Az előbbi a 'VAGY', az utóbbi a 'NEM-VAGY' ('NVAGY') függvény, illetve kapu.

- **ANTIVALENCIA(F_6) és EKVIVALENCIA(F_9) függvény, illetve 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(EXOR, XOR). Ez az F_6 függvény. Negáltja az F_9, éppen ezekben az esetekben '0', és egyezés esetén '1'. Ez utóbbi neve EKVIVALENCIA, vagy KIZÁRÓ-NEM-VAGY(EXNOR, XNOR). Az ezeknek megfelelő kapu áramköröket igen gyakran használjuk. A KIZÁRÓ-VAGY művelet jelölésére gyakran használják a '$\oplus$' szimbólumot.

3.3.3 Logikai függvények egyszerűsítése

Ebben a fejezetben azt a négy egyszerűsítési eljárást mutatjuk be, melyek a teljesen határozott logikai függvények esetében általánosságban alkalmazhatóak. Ezek a következők:

- *egyszerűsítés algebrai módszerrel*
- *Quine módszer*
- *Quine-McCluskey módszer*
- *Karnaugh táblás módszer*

3.3.3.1 Algebrai módszer

Egyszerűsítsük a bemutatott azonosságokat kihasználva a már megismert háromváltozós logikai függvényünket! A disztributivitást kifejező azonosságok arra mutatnak, hogy a klasszikusan kiemelésnek nevezett átalakítás itt is végrehajtható. A logikai összeg első két szorzat-termjéből és a második két szorzat-termjéből is kiemelhető egy-egy kétváltozós szorzat. Ezután felismerhetjük a zárójelben lévő összegek '1' értékét, valamint azt, hogy a '1'-el való beszorzást nem kell feltüntetni. Az eredmény kifejezés tehát azonos a kiindulásival, de jóval egyszerűbb annál:

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

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

3.3.3.2 A Quine módszer

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

Példa: adott a korábbi példánkban szereplő mintermes függvény leírás, melyhez az alábbi táblázatot készíthetjük.

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	$AB\overline{C}$ ✓		

3.3.3.2.a ábra[1]

A Quine-módszer oszlopai

Gondoljunk arra, hogyan hajtottuk végre az algebrai egyszerűsítést az

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

és az

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

mintermek közös tényezőjének, az

$$\overline{A}B$$

kiemelésével. A zárójelen belül a

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

marad. Ezzel a két összevont minterm helyén a közös szorzó-tényező marad, a zárójelen belül maradó változó eltűnik. A Quine-módszer lényege ennek az eljárásnak a szisztematikus ismétlése mindaddig, amíg ilyen eltüntethető változó már nem marad. Ha a változók száma 'n', először az 'n' tényezős minterm párokat vonjuk így össze 'n-1' tényezős termekké, azután a kiadódó 'n-1' tényezős term-párokat 'n-2' tényezős termekké, azután a kiadódó 'n-2' tényezős term-párokat 'n-3' tényezős termekké, és így tovább.
Az 3.3.3.2.a ábrán mutatjuk a szisztematikus eljárást ismert példánkra. Azok a mintermek, amelyekhez a függvény '1'-et rendel, az 'I.'-el jelölt oszlopban láthatók, bináris értékeikkel megcímezve. A II. oszlopban az összevonható minterm-párok címei és az összevonások eredményei láthatók. Ha a II. oszlopban lennének összevonható kéttényezős szorzatok, akkor a harmadik oszlop nem lenne üres.
Figyeljük meg, a kiemeléssel összevonható termek sajátossága, hogy azok mindig csak egyetlen változóban különböznek egymástól. Azt mondjuk, hogy az ilyen termek 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 (3.3.3.2.b 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) $\boxed{A\overline{C}}$			⊛	⊛

3.3.3.2.b ábra[1]

Prímimplikánsok lefedési táblája

Nyilvánvaló például, hogy az

$$\overline{A}B$$

prímimplikáns az

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

és az

$$\overline{A}BC$$

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:

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

A fentiekben megismert néhány új fogalom összefoglalva:

- **prímimplikáns:**
  olyan term, amelyből nem hagyható el több változó

- **megkülönböztetett minterm :**
  olyan minterm, amelyet csak egy prímimplikáns fed le

- **lényeges prímimplikáns:**
  olyan prímimplikáns, amely csak egy megkülönböztetett mintermet tartalmaz.

3.3.3.3 A Quine-McCluskey módszer

A Quine-McCluskey egyszerűsítési módszer részletes elemzése és bemutatása nem képezi e tárgy törzsanyagának részét, ezért röviden összefoglalva a következőket kell tudni a módszer lényegéről: ez az egyszerűsítési 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.

A módszer lényegének összefoglalása az alábbi linken tanulmányozható:

https://en.wikipedia.org/wiki/Quine%E2%80%93McCluskey_algorithm

3.3.3.4 A Karnaugh táblás (grafikus) módszer

A Karnaugh-táblás minimalizálás lényegében a Quine-eljárás geometriai reprezentációja. A táblázat a nevét az elv kidolgozójának, Maurice Karnaugh amerikai fizikusnak a nevéről kapta.

Az egyszerűsítés elvégzéséhez szükséges táblázat megalkotásának elve a következő: egy 'n' változós függvény lehetséges mintermjeinek megfeleltetünk egy-egy négyzetet, és úgy helyezzük el őket, hogy -mind vízszintesen, mind függőlegesen- az egymástól egységnyi távolságra lévő, azaz csak egyetlen bit-ben különböző mintermeket reprezentáló négyzetek szomszédosak legyenek, vagyis így bináris kódszavuk ún. Hamming-távolsága '1'. Ennek az a nagy előnye, hogy igen könnyen észrevesszük az összevonható mintermeket illetve termeket, hiszen azok egymás mellett helyezkednek el.

Attól függően, hogy egy adott függvényt illetve mintermet hány változó határoz meg, az adott bitszám szerint nevezzük el az elkészített táblázatot (pl. kétváltozós Karnaugh tábla, háromváltozós Karnaugh tábla stb.)

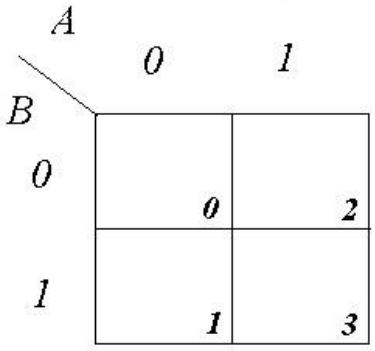
Példák:

- *kétváltozós Karnaugh tábla*

Két változó (*A,B*) esetén 4 lehetséges mintermünk van. Ezeket egy 2 x 2 –es négyzeten helyezzük el (3.3.3.4.a ábra). Az egyes négyzetek jobb alsó sarkában megjelöltük az adott binárisan kódolt mintermhez tartozó decimális számértéket is.

Megjegyzés: általános elvként alkalmazzuk a felsorolási sorrend szerinti helyiérték értelmezést, vagyis esetünkben '*AB*' kétváltozós függvényben '*A*' képezi a legmagasabb

helyiértékű bitet (MSB, =**M**ost **S**ignificant **B**it). A további példákban is ezt az értelmezést használjuk.



3.3.3.4.a ábra[1]

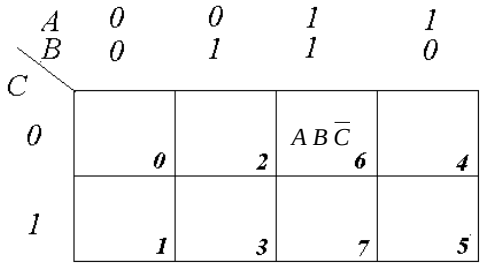
Kétváltozós (A,B) Karnaugh tábla

- *háromváltozós Karnaugh tábla*

Három változó esetén 8 lehetséges mintermünk van. Ezeket egy 4 x 2 –es téglalapon helyezzük el, megfelelő 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 Hamming-távolságra vannak egymástól. Ezért a különleges és szokatlan sorrend, azaz (0 1) után az (1 1) következik!
Nézzük az 3.3.3.4.b 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:

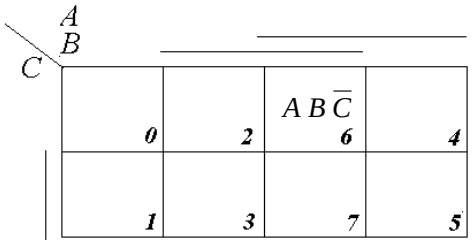
$$A\,B\,\overline{C}$$

A Karnaugh táblákhoz tartozó bináris változók logikai értékei –az adott sorokban és oszlopokban- másfajta jelöléssel is megadhatóak. Erre mutat példát a 3.3.3.4.c ábra szerinti táblázatforma.



3.3.3.4.b ábra

Háromváltozós Karnaugh tábla



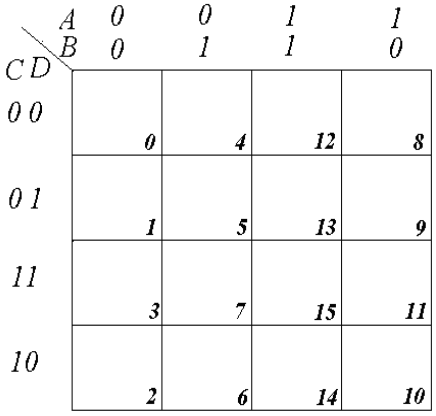
3.3.3.4.c ábra

Háromváltozós Karnaugh táblázat más logikai érték jelöléssel

A táblázatok szerkesztésénél figyeljünk 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.

- *négyváltozós Karnaugh tábla*

A négyváltozós Karnaugh tábla összeállítása és peremezése elvi újdonságot nem jelent, de a széleken elhelyezkedő mintermek közötti szomszédosságokat –az előzőekben említett okból kifolyóan- érdemes megvizsgálni (3.3.3.4.d ábra).



3.3.3.4.d ábra[1]

Négyváltozós Karnaugh tábla

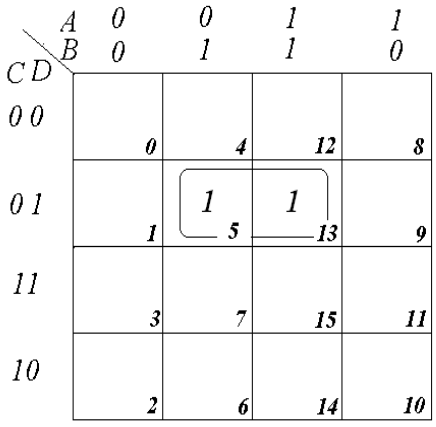
Az egyszerűsítés elve, azaz a lehető legnagyobb összevont termek kiolvasása a táblázatból

Példa:

egy négyváltozós Karnaugh táblában az 5-ös és 13-as (decimális számértékű) mintermekhez ’1’-et rendel egy teljesen határozott logikai függvény (3.3.3.4.e ábra). Az összevonhatóság elvét követve az egyesített (összevont) term itt egy kétnégyzetes téglalap, amelynek a címéből az ’*A*’ változó értéke már hiányzik, hiszen az *’A’* az a változó, amely a VAGY művelettel kiesik.

Így az összevont cím algebrai alakja :

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



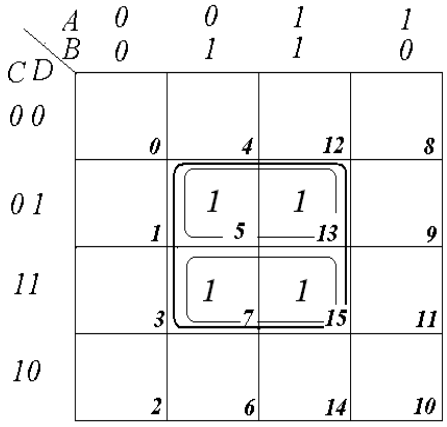
3.3.3.4.e ábra[1]

Összevont term létrehozása

Megjegyzés: a táblázatban felesleges a '0'-ás minterm értékek jelölése, így az - a könnyebb átláthatóságot szem előtt tartva - elhagyható.

Tegyük most fel, hogy az eddigi mintermeken kívül szerepel a függvényben a 7-es és a 15-ös is (3.3.3.4.f ábra). Ez utóbbiakat egymással összevonva felismerjük, hogy az így kialakult két mintermes két term ugyancsak szomszédos, és a C változó is kiejthető. Ezzel egy geometriailag még nagyobb, négy mintermes term adódott az összevonással, amely már csak két változós. Kiolvasva:

$$B\,D$$



3.3.3.4.f ábra[1]

Két szomszédos term összevonásával képzett kétváltozós term

3.3.4 Teljesen és nem teljesen határozott függvények egyszerűsítése Karnaugh táblán

3.3.4.1 Teljesen specifikált függvények egyszerűsítése a Karnaugh tábla segítségével

A tervezés lépései

1. lépés: a szükséges méretű(változószámú) Karnaugh tábla felvétele
2. lépés: a függvény '1' értékeihez tartozó mintermek bejelölése
3. lépés: az összes prímimplikáns-terület kijelölése összevonással
4. lépés: az egyszerűsített függvény felírása a rendelkezésre álló prímimplikánsok közül történő válogatással

Példa:

egyszerűsítsük Karnaugh táblán a korábbi példában szereplő, teljesen határozott függvényünket!

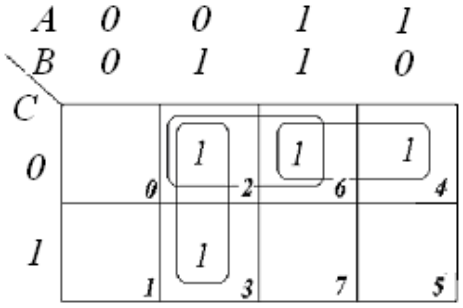
$$F(A,B,C) = \overline{A}\,B\,\overline{C} + \overline{A}B\,C + A\,\overline{B}\,\overline{C} + A\,B\overline{C}$$

Az 1. a 2. és a 3. lépést illusztráljuk az 3.3.4.1.a 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 Karnaugh táblán végezhető el a legegyszerűbben. Képzeljük el, hogy a prímimplikánsok négyszögei lemezek, amelyek felemelhetők a tábláról. Ha ezeket sorra felemeljük, és azt találjuk, hogy az felemelt term által fedett valamennyi minterm a többi term által fedve marad, akkor a felemelt term felesleges, eltávolítható. Ezzel szemben, ha a felemelés következtében valamelyik minterm fedetlen marad, a prímimplikáns elhagyhatatlan.



3.3.4.1.a ábra[1]

Lehetséges prímimplikánsok

Az egyszerűsítéshez szóba jöhető prímimplikánsok:

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

A teljes lefedettség szükséges és elégséges elvét követve látható, hogy

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

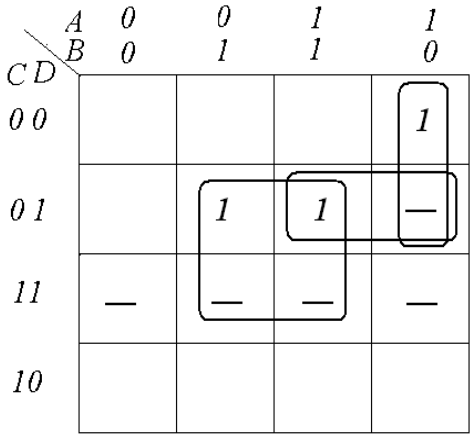
elhagyható, felesleges term.

3.3.4.2 Nem teljesen specifikált függvények egyszerűsítése a Karnaugh 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 Karnaugh-táblába az ’1’-es és közömbös (don’t care) 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’-ás mintermeknek tekintjük őket.

Példa:

adott egy négyváltozós (*A,B,C,D*), 'közömbös' mintermekkel is rendelkező logikai függvény Karnaugh táblás alakja (3.3.4.2.a ábra):



3.3.4.2.a ábra[1]

Egy nem teljesen specifikált, négyváltozós logikai függvény Karnaugh táblás alakja, a közömbös bejegyzések közötti válogatással előállított prímimplikánsok megadásával

A prímimplikánsok között egy „négymintermes", tehát kétváltozós, és a közömbös bejegyzéssel rendelkező lehetséges prímimplikánsok közötti válogatással előállított két 'kétmintermes', vagyis 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.

(Megjegyzés: a függvényalakban a teljes lefedettséget biztosító prímimplikánsok mellett felírt felesleges, ún. redundáns prímimplikánsok alkalmazása a gyakorlati, kapuszintű megvalósítás során bizonyos esetekben szükséges lehet, melynek pontos okaival a későbbiekben részletesen foglalkozni fogunk.)

A tervezés lépései

1. lépés: a szükséges méretű(változószámú) Karnaugh tábla felvétele
2. lépés: a függvény ’1’ és ’közömbös’ értékeihez tartozó minternek bejelölése
3. lépés: a közömbös bejegyzéssel is rendelkezhető lehetséges prímimplikánsok kijelölése
4. lépés: az egyszerűsített függvény felírása a rendelkezésre álló prímimplikánsok közül történő válogatással

3.4 Egykimenetű kombinációs hálózatok tervezése

3.4.1 Teljesen specifikált egykimenetű kombinációs hálózatok tervezése

Az egykimenetű, teljesen specifikált kombinációs hálózatot egyetlen, teljesen határozott logikai függvénnyel specifikáljuk. Ennek meghatározása történhet igazságtábla megadásának segítségével, Karnaugh táblás ábrázolással, az ’1’-es mintermek számjegyes felsorolásával vagy algebrai alak megadásával. A tervezés szükséges lépései a következők:

1. igazságtábla és/vagy Karnaugh tábla megadása,
2. függvény kiolvasás és egyszerűsítés a Karnaugh táblán (a minimalizálás céljából),
3. a rendelkezésre álló logikai építőelemekhez igazodó függvényalak felírás (átalakítás),
4. kapuszintű realizáció a rendelkezésre álló logikai építőelemek segítségével.

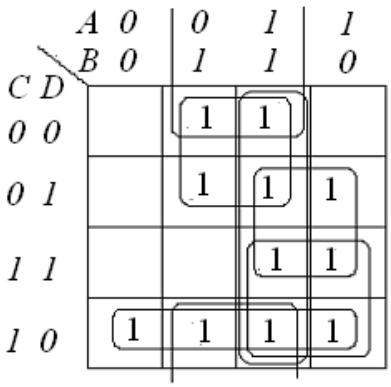
Igazságtábla vagy számjegyes minterm felsorolás esetén az ’1’-es mintermek táblázatba vitele után megindulhat a prímimplikánsok megkeresése, és a szükséges prímimplikánsok kiválasztása. Algebrai alak esetén a megadott szorzattermek Karnaugh-táblán történő ábrázolása után célszerű egy egyszerűbb lefedést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

Példa:

Tervezzük meg NEM-ÉS (NAND) kapukkal a következő specifikációval megadott négybemenetű(A,B,C,D), egykimenetű (F), teljesen specifikált kombinációs hálózatot!

$$F('1') : (2, 4, 5, 6, 9, 10, 11, 12, 13, 14, 15)$$

Az 3.4.1. a ábra mutatja a Karnaugh táblát az összevonásokkal, illetve a lehetséges prímimplikánsokkal:



3.4.1.a ábra[1]

Az F függvény lehetséges prímimplikánsai

Kiolvasva a prímimplikánsokat:

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

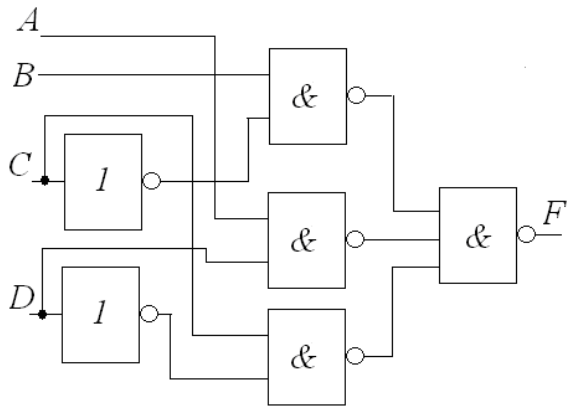
Meghatározva a minimális irredundáns lefedést :

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

Mivel a feladatkiírás szerint a realizációt tisztán NAND kapuk felhasználásával kell elvégezni, ezért a kapott függvényalakra a következő De-Morgan átalakítást kell alkalmaznunk:

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

Az áramköri realizáció rajza a 3.4.1.b ábrán látható:



3.4.1.b ábra[1]

NAND-NAND realizáció

Megjegyzés: NAND realizáció esetében az inverter alkalmazása megengedett, mind „kvázi NEM-ÉS" tag (tekinthető egy két bemenetét közösített NEM-ÉS kapu elemnek).

3.4.2 Nem teljesen specifikált egykimenetű kombinációs hálózatok tervezése

Az egykimenetű, nem teljesen specifikált kombinációs hálózatot egyetlen, nem teljesen határozott logikai függvénnyel specifikáljuk. Ennek meghatározása történhet igazságtábla megadásának segítségével, Karnaugh táblás ábrázolással, az ’1’-es és a közömbös (don’t care = ’-’) mintermek számjegyes felsorolásával vagy algebrai alak megadásával. A tervezés szükséges lépései a következők:

1. igazságtábla és/vagy Karnaugh tábla megadása,
2. függvény kiolvasás és egyszerűsítés a Karnaugh táblán (a minimalizálás céljából),
3. a rendelkezésre álló logikai építőelemekhez igazodó függvényalak felírás (átalakítás),
4. kapuszintű realizáció a rendelkezésre álló logikai építőelemek segítségével.

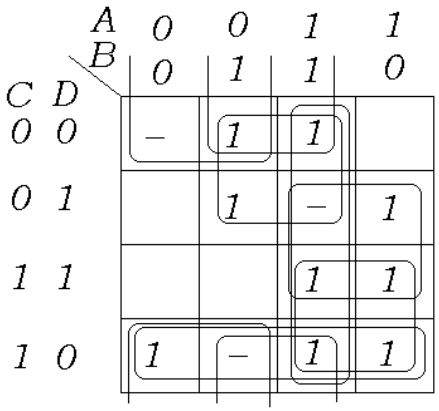
Példa:

Tervezzük meg NEM-ÉS (NAND) kapukkal a következő specifikációval megadott négybemenetű(A,B,C,D), egykimenetű(F), nem teljesen specifikált kombinációs hálózatot!

$$F(\text{'1'}) : (2, 4, 5, 9, 10, 11, 12, 14, 15)$$

$$F(\text{'-'}): (0, 6, 13)$$

Az 3.4.2.a ábra mutatja a Karnaugh táblát az összevonásokkal, illetve a lehetséges prímimplikánsokkal:



3.4.2.a ábra[1]

Az F('1') és F('-') függvények lehetséges prímimplikánsai

A Karnaugh táblából 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 a végleges, közömbös mintermek felhasználásával felírt *F('1')* logikai függvény termek logikai összegével felírt alakja:

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

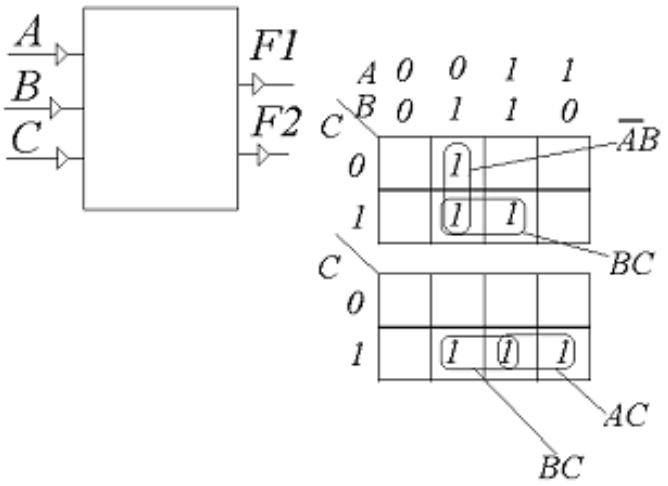
Az áramköri realizáció –az előző példa megoldásaként adódott végső függvényalakkal történő egyezőség miatt- azonos a 3.4.1.b ábrán láthatóval.

3.5 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 vannak egyszerűbb alakra vezető megoldások is. Ennek illusztrálására tekintsük a következő tervezési feladatotokat.

Példa1:

adott egy három(*A,B,C*) bemenetű és két(*F1, F2*) kimenetű kombinációs hálózat 3.5.a ábrán látható Karnaugh táblák szerinti specifikációja:



3.5.a ábra[1]

Egy három bemenetű és két kimenetű kombinációs hálózat Karnaugh táblás specifikációja

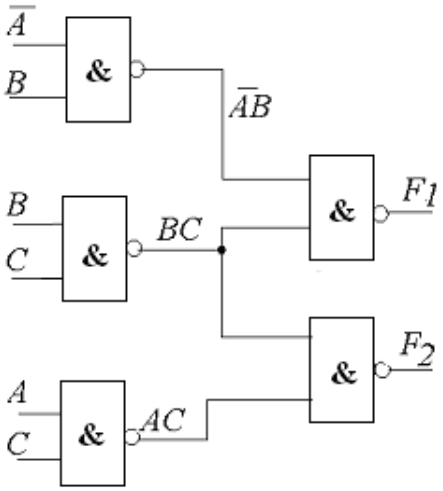
Ha a két kimenet függvényének prímimplikánsait egymástól függetlenül keressük meg, a következő két kifejezést kapjuk:

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

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

A két függvény prímimplikánsai között találunk egy közös, mindkét lefedésben prímimplikáns szerepet játszó termet, a ’*BC*’-t. Ezt nyilvánvalóan csak egyszer, azaz egy ÉS vagy egy NEM-ÉS kapuval realizáljuk, hiszen mindkét második szinten levő VAGY, illetve NEM-ÉS kapuba bevezethető az ezt reprezentáló logikai jel (3.5.b 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 Karnaugh 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.

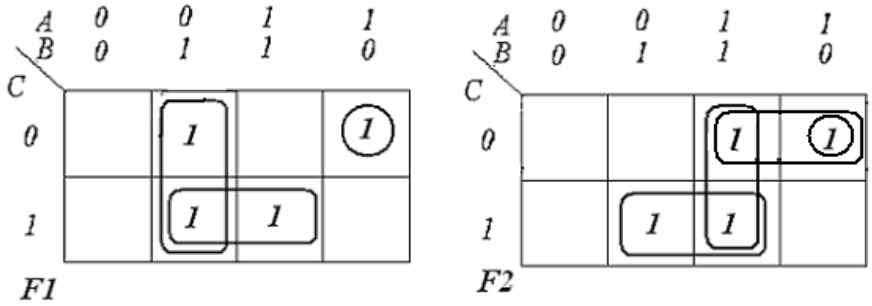


3.5.b ábra[1]

NAND realizáció a közös prímimplikáns(BC) egyszeri megvalósításával

Példa2:

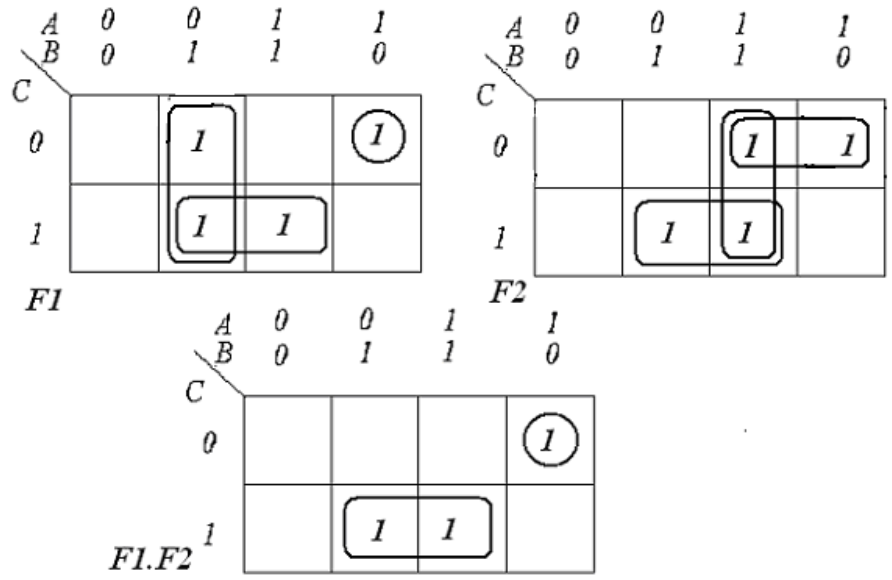
adott egy három(*A,B,C*) bemenetű és két(*F1, F2*) kimenetű kombinációs hálózat 3.5.c ábrán látható Karnaugh-táblák szerinti specifikációja:



3.5.c ábra[1]

'A,B,C' bemenetű és 'F1','F2' kimenetű kombinációs hálózat Karnaugh táblás specifikációja

Az előző 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 (3.5.d ábra).



3.5.d ábra[1]

Legnagyobb közös implikánsok keresése a szorzat-függvény prímimplikánsainak kijelölésével

A prímimplikáns készleteket a következő halmazok alkotják:

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

A 'BC' közös prímimplikáns, az $A\bar{B}\bar{C}$ pedig két nem közös prímimplikáns közös része, azaz egy legnagyobb közös implikáns. A két-kimenetű hálózat függvényeinek lehetséges lefedő termjeit tehát az 3.5.c ábrán látható csoportokból állítjuk össze.

Az 3.5.c ábra alapján elvégzett lefedés-vizsgálat és a felesleges implikánsok eltávolítása során előnyben részesítjük a legnagyobb közös implikánsokat. Így az *F2* teljes lefedéséhez és felépítéséhez az

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

helyett a közös

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

termet használjuk. Természetes, hogy a '*BC*'-t csak egyszer kell realizálni.

<u>A fentiek több kimenetre való általánosítása alapján megfogalmazható a több-kimenetű hálózatok egyszerűsítésének lépéssorozata:</u>

1. *felírandó valamennyi kimenethez rendelt függvény prímimplikánsa,*
2. *az összes lehetséges függvény-szorzat prímimplikánsainak megadása,*
3. *az egyes kimeneti függvények mintermjeinek lefedése a saját, más kimenetekhez nem tartozó prímimplikánsokkal, illetve azokkal a maximális közös implikánsokkal, amelyek az adott függvénynek implikánsai*

3.6 Hazárdok

A kombinációs hálózatok legfőbb ismérve - a tanultak alapján - a következők szerint foglalható össze: egy 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.

Az eddigiek során a kombinációs hálózatokat statikusan szemléltük, nem foglalkoztunk a tranziensekkel, azaz az átmeneti jelenségekkel. Általában megköveteljük, hogy a bemeneti változások hatására a kimenet a specifikációnak megfelelően reagáljon. Ez azonban a gyakorlatban, vagyis az áramköri realizációk esetében nem mindig garantálható. Ezeket a jelenségeket, vagyis a specifikációtól való eltéréseket hazárdoknak nevezzük.

3.6.1 Statikus hazárd

3.6.1.1 A statikus hazárd meghatározása

Statikus hazárdról akkor beszélünk, ha egyetlen bemeneti változó logikai értékének megváltozásakor a kimenet a specifikáció szerint nem változna, de a realizált hálózat kimenetén mégis átmeneti változás zajlik le.
Ez az átmeneti értékváltás –a kimenő logikai változó(k) lehetséges bináris értékeiből kiindulva- kétféle statikus hazárdot eredményezhet, melynek alapján a két hazárdtípus a következő:

- *’0’-ás típusú hazárd*
- *’1’-es típusú hazárd*

A ’0’-ás típusú statikus hazárdról akkor beszélünk, ha a specifikált hálózat kimenete a bemeneti változás ellenére alacsony logikai szinten (’0’) kell, hogy maradjon, de a realizált hálózat - átmenetileg - egy magas, ’1’-es logikai szintű impulzust mutat.

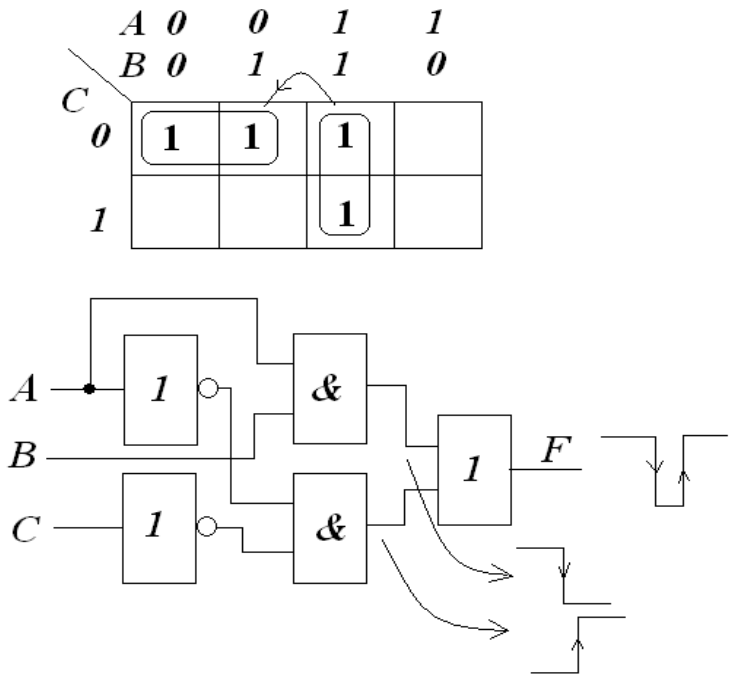
’1’-es típusú statikus hazárdról pedig akkor beszélünk, ha a specifikált hálózat kimenete a bemeneti változás ellenére magas logikai szinten (’1’) kell, hogy maradjon, de a realizált hálózat - átmenetileg - egy alacsony, ’0’-ás logikai szintű impulzust mutat.

3.6.1.2 Statikus hazárdok kiküszöbölése

A statikus hazárdok megszüntethetők ún. redundáns implikánsok bevezetésével.

Példa:

a 3.6.1.2.a ábrán láthatóak egy Karnaugh táblás ábrázolással definiált hárombenetű és egykimenetű kombinációs hálózat függvény prímimplikánsai, valamint az ehhez tartozó egyszerű kapu implementáció. Az eddigi tanulmányaink alapján kijelenthető, hogy a két prímimplikáns ($\bar{A}\bar{C}$, AB) összegszerű felírásával megadható az hálózat viselkedésének logikailag korrekt leírása.



3.6.1.2.a ábra[1]

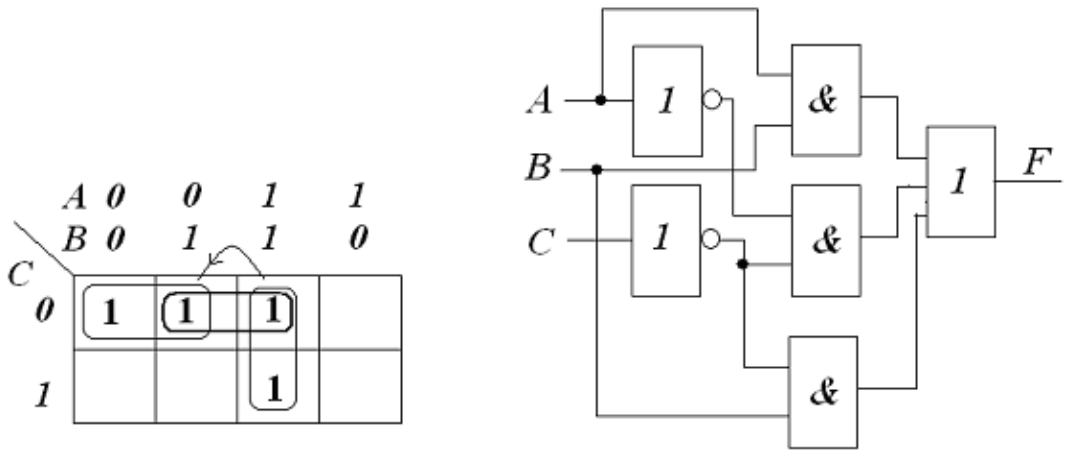
Egy 'A,B,C', bemenetű és 'F' kimenetű hálózat Karnaugh táblás és kapu szintű specifikációja, '1'-es tipusú statikus hazárddal

Vizsgáljuk most meg, hogy mi történik pl. abban az esetben. ha az *ABC* bemeneteken az 1 1 0 → 0 1 0 bitkombináció változás végbemegy, vagyis azt az átmenetet, amikor az *A* változó értéke magas szintről alacsony szintre vált.

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 magas szinten 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.

Ebben az esetben megoldást jelent egy ún. átfedő, vagy redundáns prímimplikáns felírása illetve alkalmazása (3.6.1.2.b ábra). Az '1'-es típusú statikus hazárd veszély mindig abból adódik, hogy különböző prímimplikánsokhoz tartozó '1'-es mintermek kerülnek egymás mellé. Az áthidaló implikáns ($B\bar{C}$) bevezetésével elérjük, hogy az átmenet alatt a kimenet logikai szintje állandó maradjon. A redundáns, ugyanakkor hazárd-mentesítő implikánssal együtt felírt kifejezés:

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



3.6.1.2.b ábra[1]

Statikus hazárdmentesített hálózat

3.6.2 Dinamikus hazárd

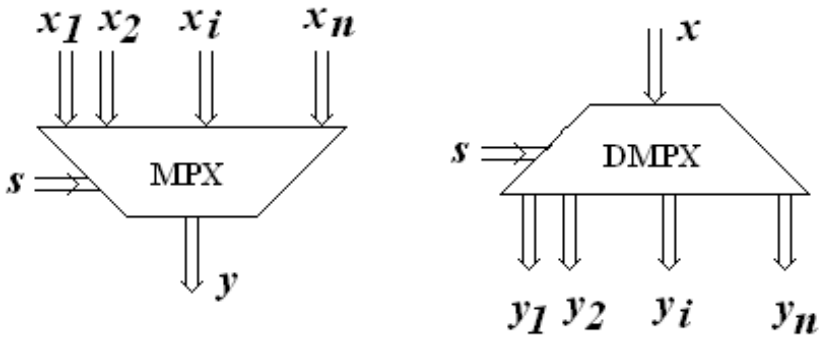
Dinamikus hazárdról beszélünk, ha egy bemeneti-változó értékváltására a kimenetnek logikai értéket kell váltania, de ez egy átmeneti visszatérés kíséretében zajlik le. Belátható, hogy a dinamikus hazárd többszintű hálózatokban lép fel akkor, ha a hálózat valamely része nincs statikus hazárdoktól mentesítve.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

3.6.3 Funkcionális hazárd

Funkcionális hazárd esetén több bemeneti változó együttes változása a kimeneten nem előírásszerű, többszörös szintváltást eredményezhet. Ezek a hazárdok csak késleltetési manipulációkkal küszöbölhetők ki, de az a legjobb, ha a tranziensek továbbterjedését szinkronizációval megakadályozzuk.

3.7 Programozható kombinációs hálózatok

A digitális architektúrákban alkalmazott kombinációs hálózatok egy speciális családját alkotják a multiplexerek (MPX vagy MUX) és a demultiplexerek (DMPX vagy DEMUX). Ezek lényegében az összetett digitális egységek azon csoportjába sorolhatók, melyek adatút kijelölő egységekként funkcionálnak. Általános felépítésüket a 3.7.a ábrán tanulmányozhatjuk:



3.7.a ábra[1]

A multiplexerek(baloldali kép) és demultiplexerek(jobboldali kép) szinbólumai

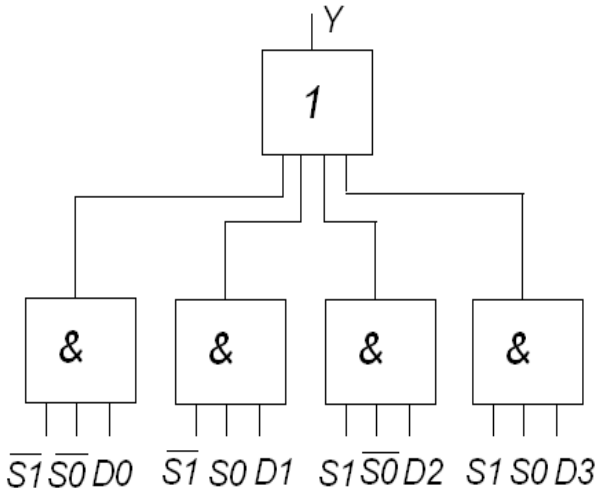
Az adatutak kijelölése vezérlőbemenetek segítségével (S), címzéssel történik. Multiplexereknél több forrás közül kijelöljük azt (x_i), amelynek adata a kimenetre (y) kerül, míg a demultiplexerek esetén azt a kimenetet (y_i) címezzük meg, amelyen az egyetlen forrás adatát (x) meg kívánjuk jeleníteni. A címzés általában bináris kóddal történik, így eleve kizárt az ellentmondásos kettős címzés.

Ebben a modulrészben a multiplexerek felépítésével foglalkozunk részletesebben, mivel a következő fejezetben a szinkron sorrendi hálózatok tervezésekor, mint programozható logikai hálózati elemet használjuk majd fel egy speciális célarchitektúra alkotóelemeként.

3.7.1 A multiplexerek általános felépítése

Példa:

vizsgáljuk meg egy négybemenetű, egykimenetű multiplexer (MPX4-1) felépítését a 3.7.1.a ábra segítségével:



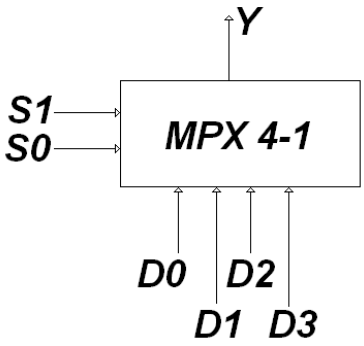
3.7.1.a ábra[1]

Egy MPX4-1 multiplexer kapuszintű felépítése

Ez a multiplexer négy darab 3-bemenetű 'ÉS' kapuból, és egy darab 4-bemenetű 'VAGY' kapuból, illetve az ezeknek megfelelő bemenetszámú 'NEM-ÉS' kapukból áll. (A kiválasztó bemenetek negált szintjei inverterekkel vannak megvalósítva.)

Természetesen a realizáció a gyakorlatban NAND-NAND kivitelben is létezik. Belátható, hogy ezek az eszközök valóban kombinációs hálózatként épülnek fel.

A példánkban szereplő multiplexer közepes integráltságú logikai áramkörökben (MSI= Medium-Scale Integration) használatos jelölése a 3.7.1.b ábrán látható:



3.7.1.b ábra[1]

MPX4-1 multiplexer MSI szimbóluma

3.7.2 A multiplexer, mint programozható logikai hálózat

A multiplexerek - adatút választó funkciójuk mellett – az AND-OR felépítésű struktúrájukból fakadóan alkalmasak egyszerűbb logikai függvények kapuszintű megvalósítására is. Ezzel lényegében ún. programozott logikai hálózatként működnek. Ennek lényegét a következők szerint lehet értelmezni: a mintermes kanonikus alakban megadott függvény mintermjeit a címző (vezérlő) bemenetekre adott címek képviselik, és a megcímzett adatbemenetre rá kell kapcsolnunk az adott mintermhez tartozó logikai értéket. Ezeknek a logikai konstansoknak a bemenetekre való kapcsolását tekinthetjük a multiplexerek programozásának.

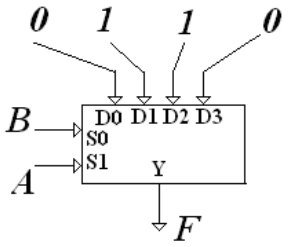
Példa:

valósítsuk meg két változó (*A*,*B*) esetében az EXOR függvényt (*F*) multiplexer felprogramozásával!

Ebben az esetben – a fenti meghatározás szerint – az F függvény A,B változói adják a multiplexerünk két címző bemenetét (3.7.2.a ábra). Ez azt is jelenti egyben, hogy a szóban forgó függvény realizálásához egy MPX4-1-es multiplexert kell felprogramoznunk. Az EXOR relációt leíró függvényalak (F) két minterme ($\bar{A}B, A\bar{B}$) a multiplexer D_1 és D_2 adatbemenetét címzi meg, és ezekre az adatbemenetekre kell rákapcsolni konstans értékként a mintermhez rendelt '1'-es logikai értéket:

$$F = \bar{A}B + A\bar{B}$$

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

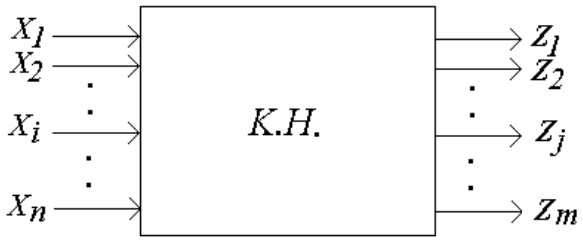


3.7.2.a ábra[1]

EXOR függvény(F) megvalósítása MPX4-1 multiplexerrel

Összefoglalás *(kombinációs hálózatok)*

A kombinációs hálózatok felépítését és viselkedését „fekete-doboz" modellként értelmezve (3.1.c ábra) a következő kijelentéseket tehetjük:



3.1.c ábra[1]

A kombinációs hálózat „fekete-doboz" modellje

A modellt a következő logikai egyenletrendszerrel írhatjuk le:

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

ahol

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

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.

Megjegyzés: a modell az időbeli, tranziens viselkedést nem írja le, de a kapuszintű áramköri realizációkban a bemeneti változások hatása időkésleltetésekkel('Δt') jelentkezik a kimeneteken. Azaz: egy 't' időpontban megváltoztatott bemeneti kombináció hatása valamilyen 'Δt' késleltetéssel jelenik meg a kimeneten, 't+Δt' időpontban. Ezt az időt válaszidőnek is nevezik.

A kombinációs hálózatok (KH) általános algebrai modellje az alábbiak foglalható össze:

KH = < I, δ, O >

I : az $x_1, x_2, \ldots x_n$ bemenetek felett értelmezett összes bemeneti variáció részhalmaza,

O : az $y_1, y_2, \ldots y_m$ kimenetek felett értelmezett kimeneti variációk halmaza,

δ : függvény, ami az **I** elemeit az **O** *halmazba* képezi le :

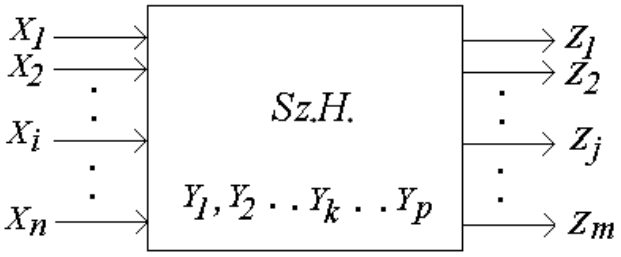
$\delta : \mathbf{I} \rightarrow \mathbf{O}$, azaz

$\delta(i_j) = o_k$, ahol i_j az **I**, o_k az **O** halmaz egy-egy eleme.

4. Sorrendi hálózatok tervezése

4.1 A sorrendi hálózatok „fekete-doboz” modellje

A sorrendi, más néven szekvenciális hálózatok „fekete-doboz” modellje a 4.1.a ábrán látható:



4.1.a ábra[1]

Szekvenciális hálózat „fekete-doboz” modellje

A hálózat működését a bemeneti($X_1, X_2, \ldots X_i, \ldots X_n$) és kimeneti($Z_1, Z_2, \ldots Z_j, \ldots Z_m$) jelek mellett az ún. másodlagos vagy szekunder változók($Y_1, Y_2, \ldots Y_k, \ldots Y_p$) is jellemzik. Ezek jelenléte biztosítja a „szekvencialitást”, vagyis a sorrendi hálózatok azon tulajdonságát, mely szerint a kimeneti kombináció nem csak a pillanatnyi bemeneti kombinációtól függ, hanem a korábbi bemeneti kombinációktól, sőt azok sorrendjétől is. A modellt leíró függvénycsoportok a következőként jellemezhetőek:

$$Z_1 = f_{z1}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$
$$Z_2 = f_{z2}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Z_j = f_{zj}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Z_m = f_{zm}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$

$$Y'_1 = f_{y1}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$
$$Y'_2 = f_{y2}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Y'_k = f_{yk}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Y'_p = f_{yp}(X_1, \ldots X_i, \ldots X_n, Y_1, \ldots Y_k, \ldots Y_p)$$

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 szekunder változók új értékeit felső vonással(*Y'*) különböztetjük meg az aktuálisaktól.)

Általánosságban a hálózat teljes működésének leírására kétféle függvényt kell definiálni: az első, amit kimeneti függvénynek nevezünk(f_z) a bemeneti kombinációk halmazának és a szekunder változók kombinációiból álló halmaznak a szorzatát képezi le a kimeneti kombinációk halmazába. A második(f_y) 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:

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

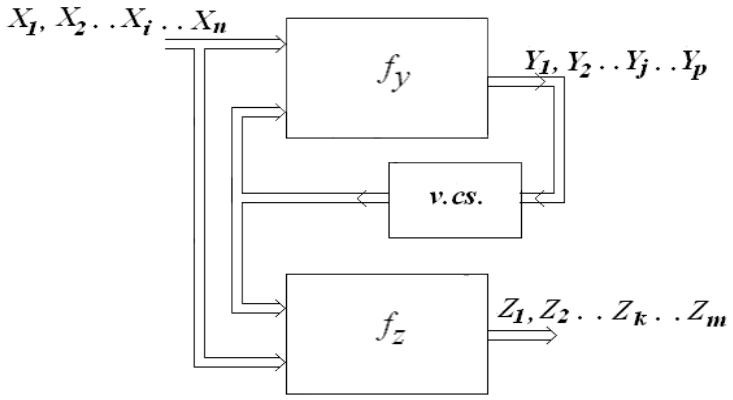
A kimeneti függvény megvalósítása egy olyan kombinációs hálózat, amelynek bemeneteire a hálózat bemenetei és az állapotváltozók csatlakoznak, tehát a bemeneti kombináció és állapot kombináció párok alkotnak egy bemeneti kombinációt az f_z hálózaton. Ha ezeket egy *t* időpontbeli értékkel jellemezzük, akkor a hálózat kimenetein valamilyen tranziens(Δt) 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-al jellemzett hálózat bemeneteire.

$$f_y : (x_i^t, y_k^t) \rightarrow y_l^{t+\Delta t}$$
$$f_z : (x_i^t, y_k^t) \rightarrow z_j^t$$
$$x_i^t : egy\ bemeneti\ kombináció\ a\ t\ \ pillanatban$$
$$z_j^t : egy\ kimeneti\ kombináció\ a\ t\ \ pillanatban$$
$$y_k^t : egy\ szekundér\ változó\ kombináció\ a\ t\ \ pillanatban$$
$$y_l^{t+\Delta t} : egy\ szekundér\ változó\ kombináció\ a\ (t+\Delta t)\ \ pillanatban$$

4.2 A sorrendi hálózatok strukturális modelljei

4.2.1 Mealy-típusú sorrendi hálózat

A szekvenciális hálózat legáltalánosabb belső struktúrája a 4.2.1.a ábrán látható:



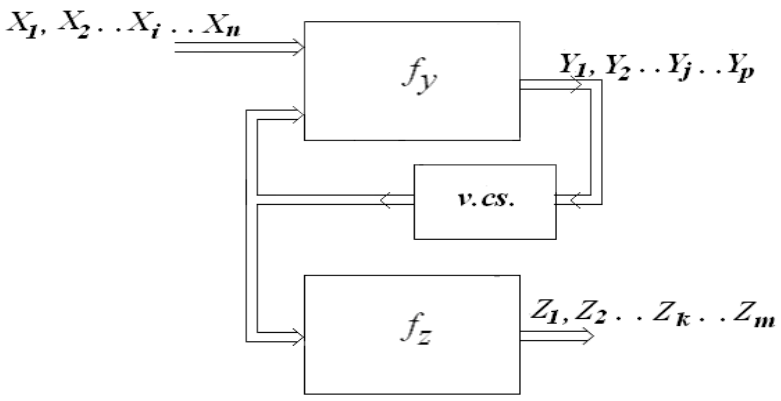
4.2.1.a ábra[1]

Mealy-típusú sorrendi hálózat sematikus ábrája

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

4.2.2 Moore-típusú sorrendi hálózat

A sorrendi hálózatoknak azt a típusát, amelynek kimeneti kombinációira csak a belső állapotok hatnak, Moore-típusú sorrendi hálózatnak nevezzük. A Moore-féle szekvenciális hálózatban a kimeneti kombinációk a belső állapotok átkódolt formáit szolgáltatják a kimeneteken (4.2.2.a ábra).



4.2.2.a ábra[1]

Moore-típusú sorrendi hálózat sematikus struktúrája

A Mealy- és Moore-típusú hálózatstruktúrák aszinkron és szinkron sorrendi hálózatok esetében egyaránt alkalmazható.

4.3 Aszinkron és szinkron sorrendi hálózatok

4.3.1 Aszinkron sorrendi hálózatok

Az aszinkron sorrendi hálózat f_y hálózatának visszacsatolása órajel nélküli, ún, '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

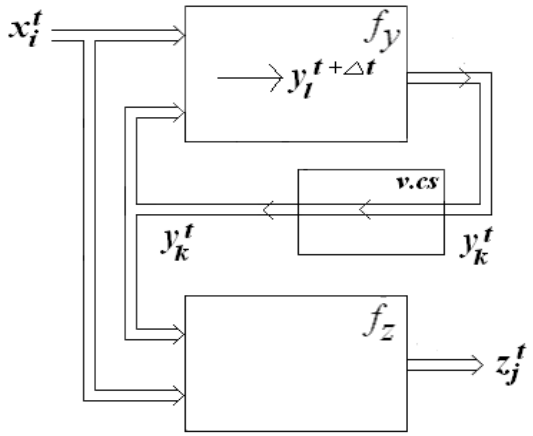
$$f_y(x_i, y_j) = y_j$$

4.3.1.1 Közvetlen visszacsatolású aszinkron sorrendi hálózatok

A 4.3.1.1.a ábra egy közvetlenül, a kimenetről vezetékekkel visszacsatolt aszinkron sorrendi hálózat struktúrájának egy olyan működési fázisát ábrázolja, amikor egy új bemeneti kombinációt már rákapcsoltunk a hálózatra, és az f_y kombinációs hálózat belsejében már megindult a következő állapot kombináció generálása($t+\Delta t$). Az f_y hálózat kimenetein azonban a változás még nem jelent meg(t), az egyelőre 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.

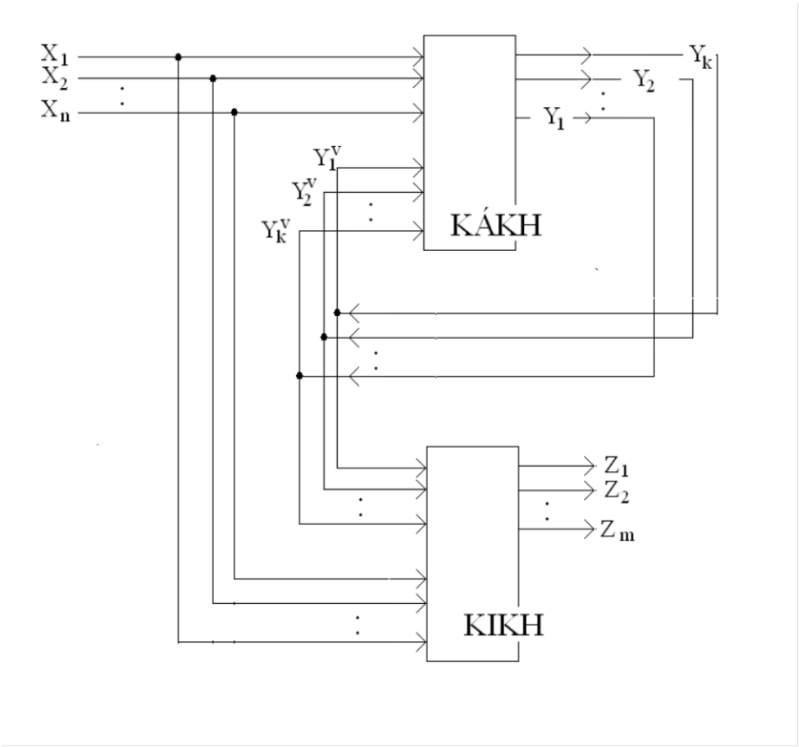
Tegyük fel, hogy a következő állapot kombináció az új bemeneti kombinációval olyan párost alkot az f_y bemenetein, amelyhez az f_y hálózat az ugyanezt a következő állapot-kombinációt rendeli. Ez azt jelenti, hogy az új állapot kombináció *stabilizálódik(t+Δt)*. Így végül a kimeneti kombináció is nyugalomba 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.



4.3.1.1.a ábra[1]

Közvetlen visszacsatolásos szekvenciális hálózat sematikus ábrája

A 4.3.1.1.b ábrán egy közvetlen visszacsatolással felépített Mealy-típusú aszinkron sorrendi hálózat sematikus ábrája látható. (Az ábrán az *n*-edik időpillanathoz képesti következő, *n+1*-edik időpillanatban bekövetkező állapotot előállító kombinációs hálózat jelölése 'KÁKH', a kimeneti értékeket(Z_m) előállító kombinációs hálózat jelölése 'KIKH'. A visszacsatolt Y_k állapot változók jelölése Y_k^v.)

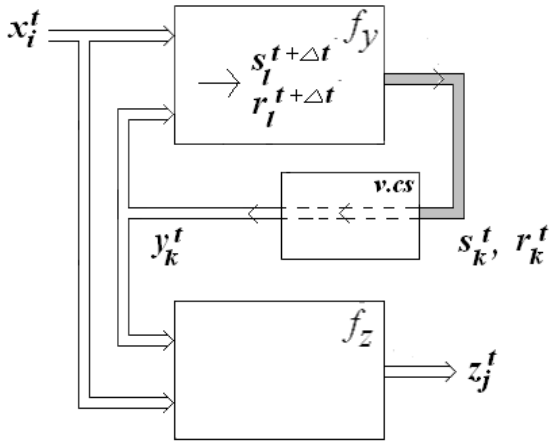


4.3.1.1.b ábra[1]

Mealy-típusú aszinkron sorrendi hálózat felépítése közvetlen visszacsatoló ágakkal

4.3.1.2 SR tárolókkal visszacsatolt aszinkron sorrendi hálózatok

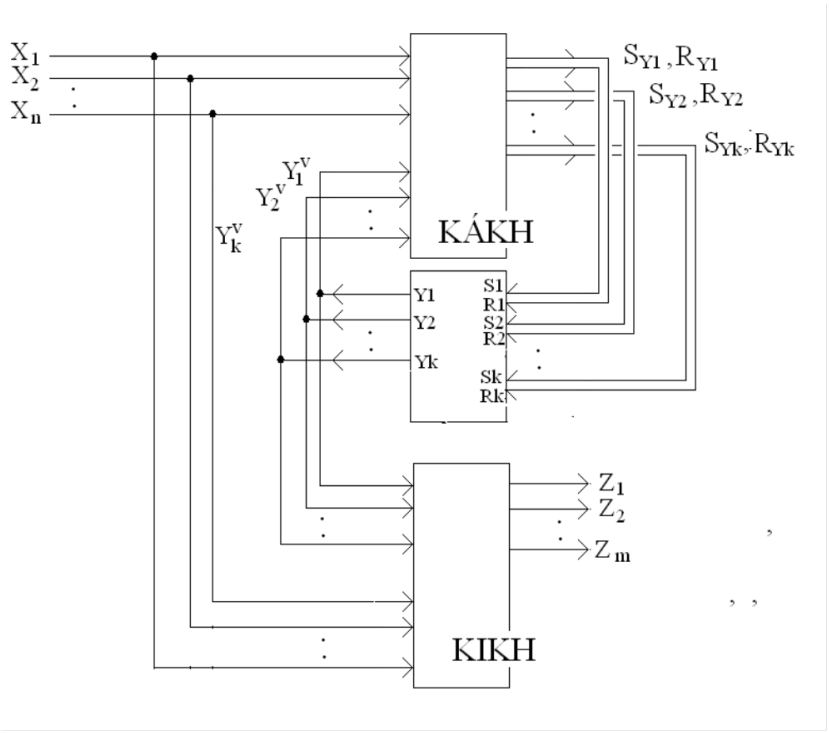
A 4.3.1.2.a ábrán egy olyan SR tárolókkal visszacsatolt aszinkron sorrendi hálózat struktúráját láthatjuk, ami azt a működési pillanatot rögzíti, amikor egy új bemeneti kombinációt már rákapcsoltunk a hálózatra, és az f_y kombinációs hálózat belsejében már megindult a következő állapotot generáló SR kombináció kialakulása ($t+\Delta t$). Az f_y hálózat kimenetein azonban a változás még nem jelent meg (t), az egyelőre még az aktuális állapotot generáló SR 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 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 SR kombinációt rendeli. Ez azt jelenti, hogy az új állapot kombináció stabilizálódik ($t+\Delta t$). 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.



4.3.1.2.a ábra[1]

SR tárolókkal visszacsatolt aszinkron sorrendi hálózat struktúrája

A 4.3.1.2.b ábrán egy olyan Mealy-típusú aszinkron sorrendi hálózat struktúrája látható, melynek visszacsatoló ágaiban SR tárolók találhatók.



4.3.1.2.b ábra[1]

Mealy-típusú aszinkron sorrendi hálózat felépítése SR tárolós visszacsatoló ágakkal

4.3.2 Szinkron sorrendi hálózatok

A szinkron sorrendi hálózat f_y hálózatának visszacsatolása órajellel('clk'), ún. MS (Master-Slave) tárolókon keresztül érvényesül. A gyakorlatban vagy D-MS, vagy JK-MS tárolós visszacsatolást használnak. Mindkét módszer közös sajátossága, hogy az órajel minden állapot kombináció fennállásának időintervallumát 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 alakítják. Ennek megfelelően sajátos jelöléseket alkalmazhatunk, a *t* időbeli kombinációkat inkább az *n* természetes számmal, a *t+Δt* időbeli kombinációkat inkább az *n+1* természetes számmal, mint sorszámokkal jelöljük. A szinkron tárolók (D-MS, JK-MS) kimeneti kombinációinak jelölésére inkább a már megismert *q* szimbólumot használjuk. Megjegyezzük, hogy az ábrákon mindkét jelölésrendszer látható, de a későbbiekben szinkron hálózatok esetén csak a most bevezetett jelölésrendszert alkalmazzuk.

4.3.2.1 Szinkron hálózatok visszacsatoló ága D-MS tárolókkal

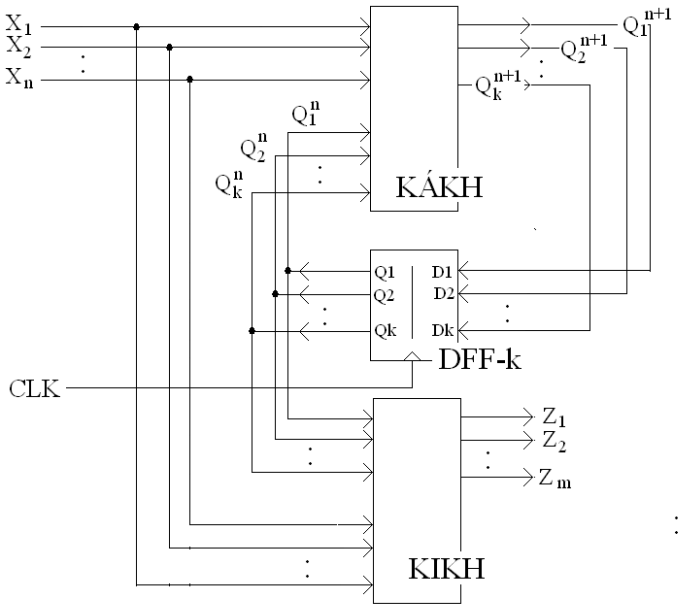
A 4.3.2.1.a és 4.3.2.1.b ábrákon egy-egy szinkron sorrendi hálózat látható Mealy-, illetve Moore-típusú hálózat struktúrában, D-MS tárolókat tartalmazó visszacsatolásokkal.

Értelmezzük a Mealy-típusú hálózat működését a 4.3.2.1.a ábra tanulmányozásával! ezen az ábrán egy olyan időpillanat látható, amikor is az *n-1*-edik órajel ciklus már lejátszódott, és az *n*-edik felfutó él következik. Ennek megfelelően a hálózatra már rákapcsoltuk az *n*-edik 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 $Q_k^{n+1} (= D_k^{n+1})$ kombináció a visszacsatoló flip-flopok bemenetein várakozik, az f_y hálózatra csatlakozó kimeneteik változatlanok, és az aktuális állapot kombinációt, a Q_k^n-et őrzik. Az f_z kombinációs hálózat a bemenetére jutó aktuális állapot kombináció és a bemeneti kombináció eredményeképpen szolgáltatja az aktuális kimeneti kombinációt. Ekkor érkezik az órajel felfutó éle. A következő állapotkód a MESTER tárolókba kerül, miközben a SZOLGA kimenetek változatlanok.

A következő változás akkor következik be, amikor az órajel lefutó éle megérkezik. Ennek hatására a D-MS tárolók kimenetein megjelenik az eddig következő állapot

kombinációnak nevezett kombináció, és aktuális állapottá válik. Ha ezt követően megváltoztatjuk a bemeneti kombinációt, akkor az f_z kimenetein egy új aktuális kimeneti kombináció, és az f_y kimenetein egy újabb következő állapot kódja jelenik meg.

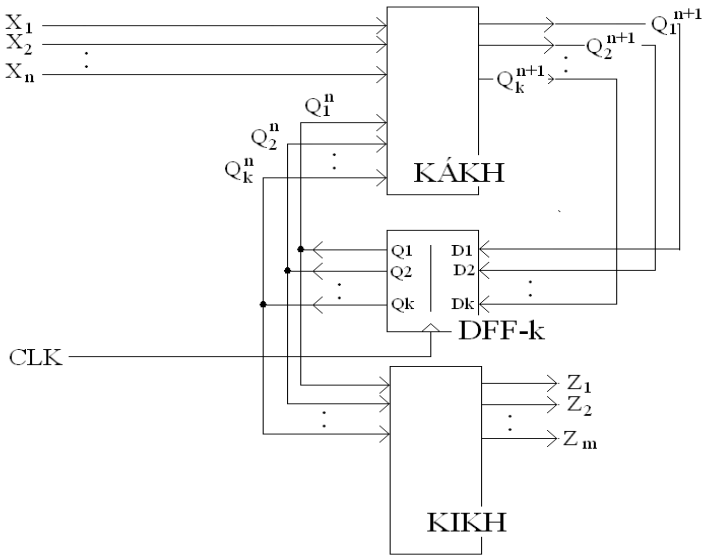
Megjegyzés: a MS tárolók egy speciális csoportját alkotják az élvezérelt tárolók, más néven „flip-flop"-ok. Kialakításukkal és működésükkel a következő, szinkron tárolókat is bemutató fejezetben ismerkedünk meg részletesen.



4.3.2.1.a ábra[1]

Mealy-típusú szinkron sorrendi hálózat D-MS tárolókkal

A Moore-típusú hálózat struktúra esetében a folyamat hasonlóképpen értelmezhető, de ebben az esetben természetesen a kimeneti értékek(Z_m) előállítása (KIKH) csupán a D flip-flopok által szolgáltatott szekuder változók értékei alapján történik (4.3.2.1.b ábra):

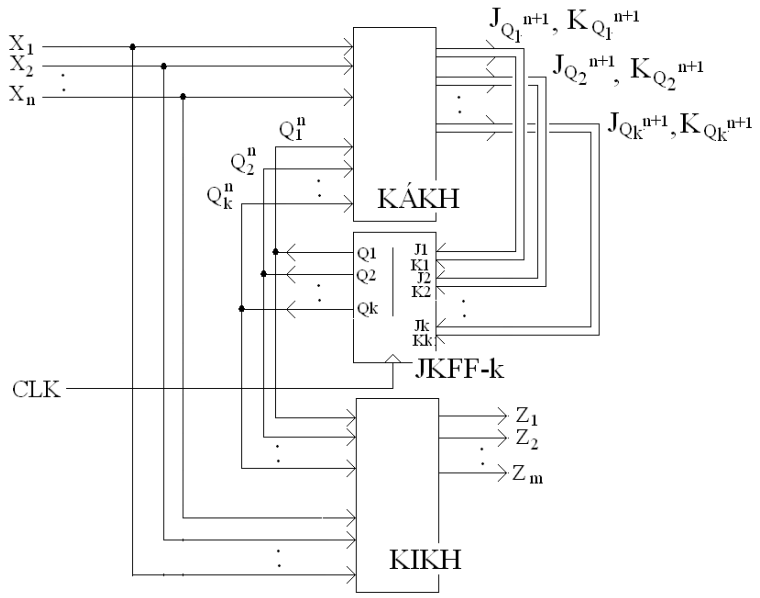


4.3.2.1.b ábra[1]

Moore-típusú szinkron sorrendi hálózat D-MS tárolókkal

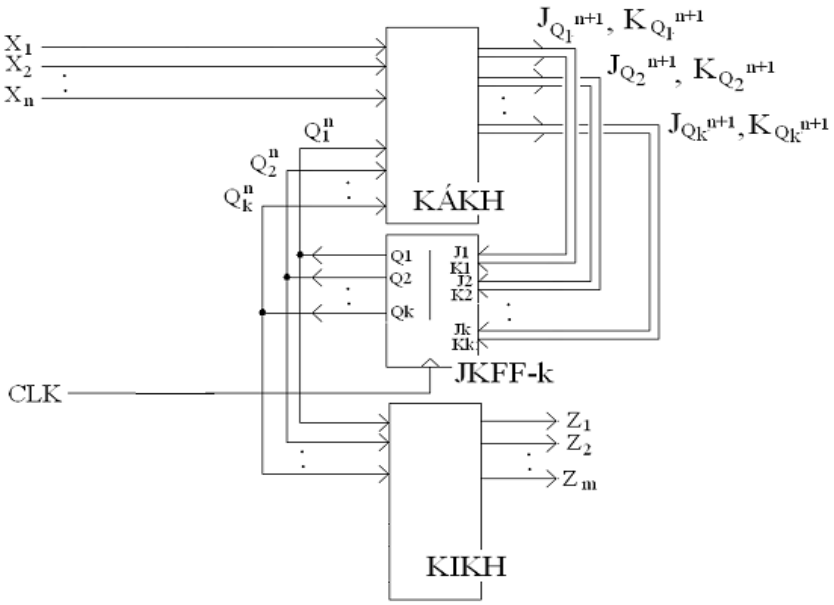
4.3.2.2 Szinkron hálózatok visszacsatoló ága JK-MS tárolókkal

JK-MS tárolóelemek alkalmazása esetében hasonló kialakítású Mealy- és Moore-struktúrákat láthatunk (4.3.2.2.a és 4.3.2.2.b ábrák), mint a D-MS tárolók használatánál, de ebben az esetben természetesen a visszacsatoló ágakban a D bemenetek helyett most külön J és K flip-flop bemeneteink vannak:



4.3.2.2.a ábra[1]

Mealy-típusú szinkron sorrendi hálózat JK-MS tárolókkal



4.3.2.2.b ábra[1]

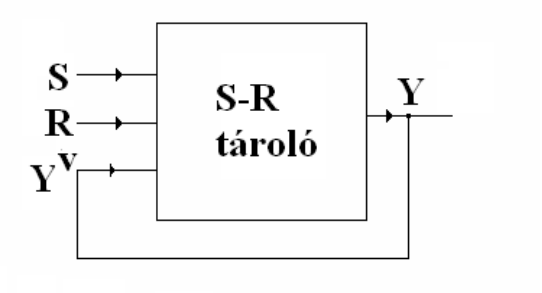
Moore-típusú szinkron sorrendi hálózat JK-MS tárolókkal

4.4 Tárolók sorrendi hálózatokban

4.4.1 Az SR tároló

A sorrendi hálózatokban a szekvencia elvének biztosítására –az előbbiekben megismertek szerint- különböző megoldások léteznek: egyrészt alkalmazhatunk közvetlen visszacsatolást (időbeni késleltetés elve a következő állapot(ok) és a kimeneti érték(ek) előállítására), illetve használhatunk különböző felépítésű, specifikus tárolóelemeket.

Aszinkron sorrendi hálózatokban gyakran használt tárolóelem az aszinkron SR tároló (egyes esetekben RS tárolóként is szokás említeni), melynek két bemenete S(set) és R(reset), kimenete Y, sematikus felépítése a 4.4.1.a ábrán látható:



4.4.1.a ábra[1]

Aszinkron SR tároló sematikus ábrája

Jól látható, hogy lényegében egy olyan kombinációs hálózatról van szó, melynek kimenete a bemenetre, mint „harmadik" bemeneti logikai váltózó érkezik vissza. Működését az egyszerű, illetve összetett igazságtábla alapján lehet értelmezni (4.4.1.b ábra):

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	-

4.4.1.b ábra[1]

Az aszinkron SR tároló igazságtáblái

Értelmezzük az egyszerű igazságtábla alapján az SR tároló működését: 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. Mindkét bemenet (S,R) együttes magas szintje tiltott kombiációnak számít („don’t care”).

Vizsgáljuk meg részletesen az első esetet (SR=00), vagyis amikor mind az S, mind pedig az R bemenet is alacsony szinten van! Azt látjuk a táblázatban, hogy ilyenkor a tároló Y kimenete megőrzi a korábbi értékét (Y^V), vagyis attól függően hol ’1’, hol pedig ’0’ logikai szinten van SR=00 esetén. Ez tehát azt jelenti, hogy ebben az esetben egyértelműen egy sorrendi hálózattal állunk szemben, hiszen ugyanarra az adott bemeneti bitkombinációra más-más választ ad a kimenetünk (attól függően, hogy <u>*korábban*</u> milyen más bemeneti értéket kapcsoltunk a hálózatunkra). Példánkban tehát a kimenet saját korábbi értékétől is függ.

Az egyszerű igazságtáblában tehát a kimenetre vonatkozó következő állapot(érték) oszlopában a határozott logikai értékek mellett szimbólumo(ka)t is találhatunk. A teljes működés értelmezéséhez fejtsük ki az egyes következő kimeneti értékekhez tartozó szimbólumok logikai értékeit a bemenetek, illetve az előző kimeneti érték függvényében: így kapjuk az összetett igazságtáblát. Ezzel 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 indexel. Ennek megfelelően az Y^v szintjeit tekintjük aktuális kimeneti értékeknek, és a Y szintjeit következő kimeneti értékeknek.

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

4.4.1.1 Az állapot-átmeneti tábla (állapottábla)

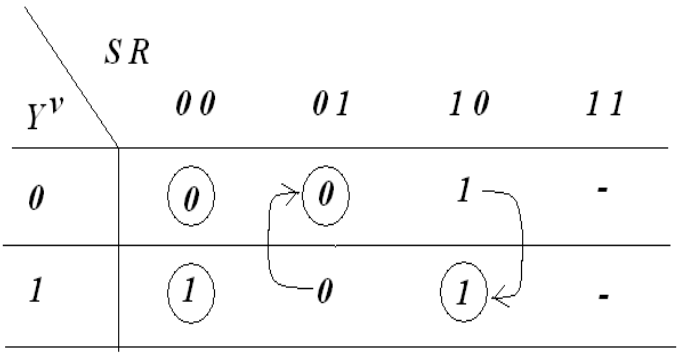
Adjuk meg az SR tároló viselkedésének leírását egy ún. állapot-átmeneti tábla segítségével! Ezt a táblát rövidebb meghatározással állapottáblának is szokták nevezni. Megadásának módja pedig a következő (4.4.1.1.a ábra):

a táblázat 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 SR=00 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 az új következő kimeneti értéken. Azaz az SR=00 bitkombinációnál az Y=0 stabil kimeneti állapot. Ezzel szemben az SR=10 és $Y^v=0$ állapot 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. A stabil állapotokat a táblázatban bekarikázva jelöljük.

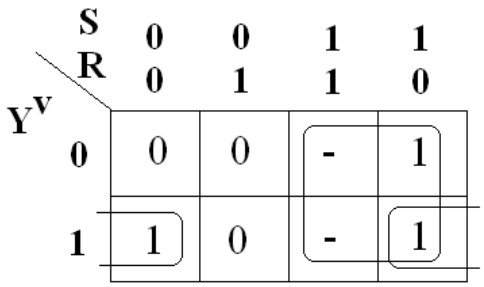


4.4.1.1.a ábra[1]

Az SR tároló állapottáblája

4.4.1.2 Az SR tároló működésének megadása algebrai függvényalakkal és logikai kapuhálózattal

A sorrendi hálózatok viselkedését leíró állapottábla felhasználásával könnyen megadható egy olyan Karnaugh tábla, melynek segítségével felírható – a már megismert minimalizálási eljárás alapján- az adott hálózat legegyszerűbb algebrai alakja. Ehhez nem kell mást tennünk, mint megszerkeszteni a megfelelő számú bemenetekkel rendelkező Karnaugh táblát, és egyszerűen csak át kell másolnunk az állapottáblából az adott kombinációkhoz tartozó következő állapot(kimenet) értékeket. SR tároló esetében ez a következőképpen néz ki (4.4.1.2.a ábra):



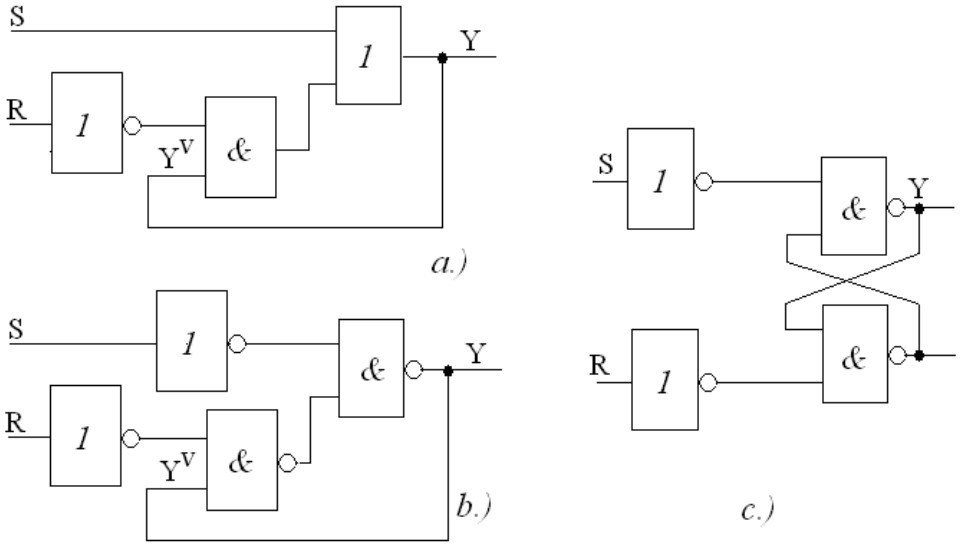
4.4.1.2.a ábra[1]

SR tároló Karnaugh táblája

A minimalizálás érdekében elvégzett lefedések eredményeképpen a következő algebrai alakokat kapjuk:

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

Ezzel a megadási móddal lehetőség nyílik többféle kapurealizációra: AND-OR(4.4.1.2.b ábra *a.)* rajz), illetve NAND-NAND (4.4.1.2.b ábra *b.)* és *c.)* rajz):



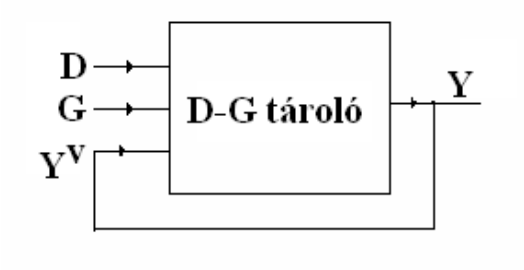
4.4.1.2.b ábra[1]

SR tároló kapuszintű realizációi

Megjegyzés: a korábbiakban(3.4.1.) már említettük, hogy NAND-NAND realizációkban az inverter, mint „kvázi NAND" elem használata megengedett.

4.4.2 A DG tároló

A DG tároló a 4.4.2.a ábra alapján lényegében egy egybites aszinkron tárolóelemnek tekinthető, mely a G(gate) bemenet felemelésekor írja a kimenetére(Y) a D(data) bemenete aktuális értékét, majd a G lefutásakor ez az érték tárolódik a kimeneten.



4.4.2.a ábra[1]

Aszinkron DG tároló sematikus ábrája

A tároló egyszerű és összetett igazságtáblája a 4.4.2.b ábrán látható:

egyszerű igazságtábla

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

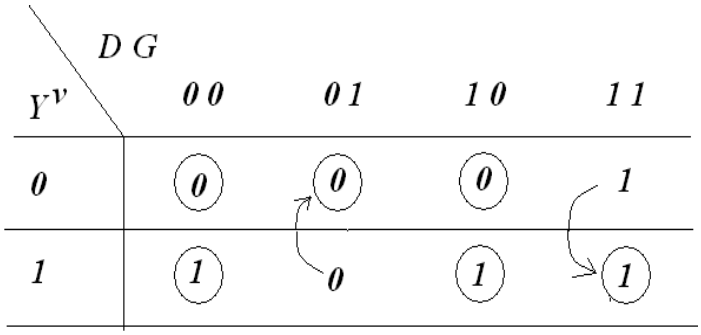
összetett igazságtábla

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

4.4.2.b ábra[1]

Az aszinkron DG tároló igazságtáblái

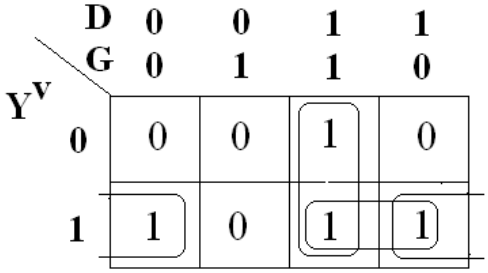
Az összetett igazságtáblából –a korábban megismert módon- könnyen származtatható az állapottábla (4.4.2.c ábra):



4.4.2.c ábra[1]

Az aszinkron DG tároló állapottáblája

A tároló működését leíró függvény felírását – az SR tárolónál alkalmazottakhoz hasonlóan- itt is Karnaugh tábla segítségével tesszük meg (4.4.2.d ábra):



4.4.2.d ábra[1]

Aszinkron DG tároló Karnaugh táblája

A táblában az összes lehetséges prímimplikánst bejelöltük. Ezekből az alábbi függvényalakot felírva elmondhatjuk, hogy az teljes mértékben lefedi a kimeneti ’1’-es

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

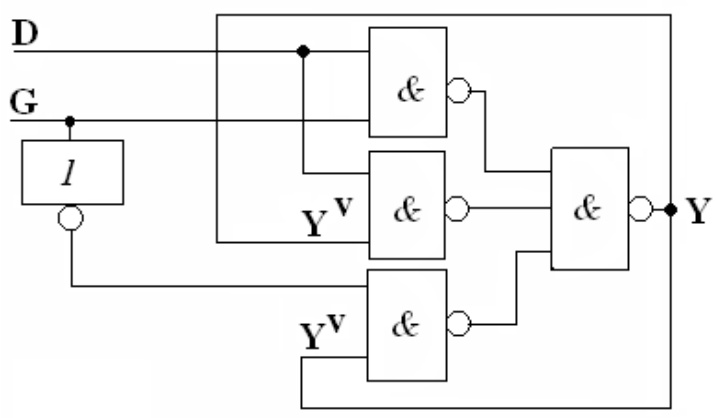
értékeket, vagyis –elméletben- eleget tesz a hálózat kimeneti viselkedésének leírására:

Azonban korábbi tanulmányainkból tudjuk, hogy áramköri realizáció esetében számolnunk kell a kapukésleltetési időkkel, vagyis az előfordulható statikus hazárddal.

Vizsgáljuk meg, hogy mit jelenthet ez a gyakorlatban: tételezzük fel, hogy a D,G, Y^v jelek rendjében ’111’-ben vagyunk, és G alacsony szintre állításával az ’101’ helyzetbe akarunk átmenni. Ha G előbb ’lefut’, minthogy a $\overline{G}$ ’felfutna’, beállhat az ’100’ állapot, és ennek hatására a hálózat ’0’-ban stabilizálódna. A helyes működés érekében kell tehát a statikus hazárdtól való mentesítés, vagyis a redundáns implikáns (DY^v) felírása. Az így kapott mintermes függvényalak:

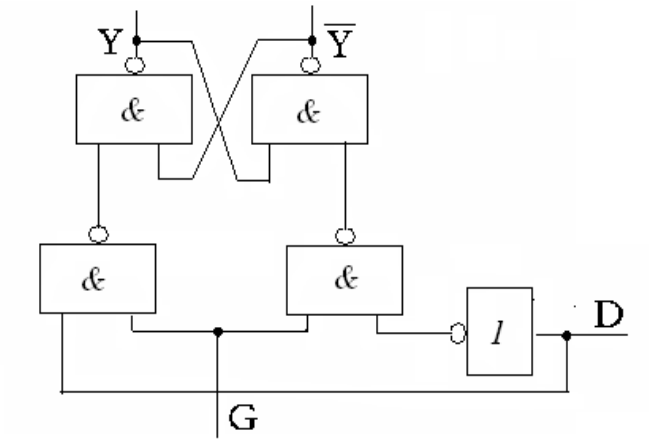
$$Y = D\,G + D\,Y^v + \overline{G}\,Y^v$$

A DG tároló lehetséges NAND-NAND realizációit mutatják a 4.4.2.e és 4.4.2.f ábrák:



4.4.2.e ábra[1]

DG tároló egy lehetséges NAND-NAND realizációja (statikus hazárdmentesített változat)



4.4.2.f ábra[1]

DG tároló NAND-NAND megvalósítása SR tároló sémájából

A DG tároló a digitális hálózatokban leggyakrabban mint klasszikus egybites memóriaelem fordul elő. A korrekt működés érdekében azonban a következő működési fázisok sorrendiségét biztosítani kell: a beírandó adatot a D adatbemeneten előkészítjük, majd a G beíró(kapuzó) 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.

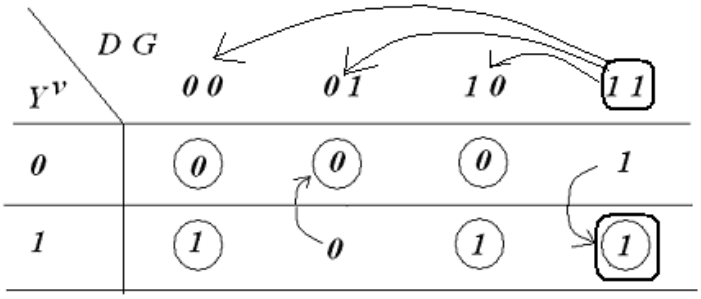
Amennyiben ez a sorrend nem biztosítható minden körülmény között, akkor több probléma is felmerülhet.

Vizsgáljuk meg a DG tároló állapottábláját(4.4.2.g ábra), és kövessük a működést az alább bemenő bitkombináció változás esetén: tételezzük fel, hogy a DGY^v bemenetek ’111’ kombinációja melletti stabil ’1’-es kimeneti állapotból most a DGY^v ’001’ változásának hatására egy másik, szintén stabil ’1’-es kimeneti értéket mutató helyzetbe kerülünk az állapottáblán. Ez azonban két ok miatt sem garantálható teljes mértékben: egyrészt két bemenetet tökéletesen egy időben nem tudunk változtatni, másrészt a két bemeneti változás hatása különböző késleltetésű utakon (áramutakon) érvényesül.

A valóságban tehát nem lehet kizárni, hogy vagy a DG-re nézve az 10, vagy a 01 bemeneti kombináció átmenetileg beáll. Így ha a 01 jelentkezik, azaz D lefutásának hatása előbb érvényesül, a kimeneten beállhat a ’0’ tranziens, amely végül, miután G is ’lefut’, az Y=0 kimenetet stabilizálja.

Ennek a lehetőségnek a kiküszöbölése csak akkor biztosítható, ha megtiltjuk a többszörös bemeneti váltásokat, azaz egyszerre csak egyetlen egy bemeneti jel értéke változhat meg. Ezzel egy olyan általános szabályt is kimondhatunk, mely szerint

visszacsatolt kombinációs hálózatok bemenetei közül egyszerre csak egyet szabad változtatni!



4.4.2.g ábra[1]

Többszörös bemeneti bitváltozás hatása DG tároló esetében ('111'→'001')

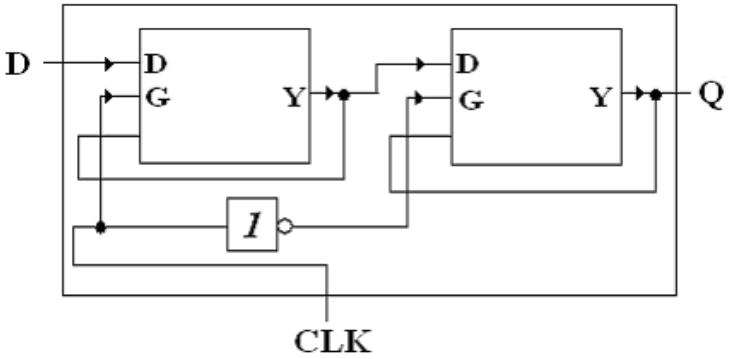
Az aszinkron DG tároló felépítéséből származó további hátránya, hogy a G=1 helyzetben a D-re adott változások kijutnak a kimenetre. *A G=1 helyzetben tehát a tároló a D-bemenet felől „átlátszó" (transzparens).* Ez sok esetben nemkívánatos és további problémák forrását jelentheti. A megoldás ilyenkor olyan tárolók alkalmazása, melyek a bemenet felől „nem átlátszók, azaz nem transzparensek. Erre az igényre nyújtanak megoldást az ún. Mester-Szolga (MS) kialakítású szinkron tárolók.

4.4.3 Mester-szolga (MS) tárolók

4.4.3.1 D-MS tároló

Az aszinkron DG tároló felhasználásával alakítsunk ki olyan tároló komplexumot, mellyel kiküszöbölhető a transzparencia jelensége egy tárolóban! Ennek érdekében a következők szerint járjunk el: kössünk sorba egymás után két aszinkron DG tárolót a 4.4.3.1.a ábra szerinti kialakításban, azaz az első tároló Y kimenetét kössük a második tárolónk D bemenetére, és ennek a második tárolónak az Y kimenetét nevezzük el Q kimenetnek. Az így kapott kétfokozatú tárolónk bemenete legyen tehát az első fokozat D adatbemenete, a kimenete pedig a második fokozat Q kimenete. Továbbá azt a bemenetet, amelyről az első tároló G bemenetét vezéreljük, és amelynek negált változója a másik tároló beírását vezérli, speciális funkcióval ruházzuk fel: órajel ('clock' vagy 'clk') lesz a neve, illetve funkciója. Ezt az órajelet szinkronizáló jelnek is nevezik: innen ered a „szinkron hálózat" elnevezés. Az ilyen szinkronizáló jelet nem tartalmazó áramköröket pedig aszinkron áramköröknek nevezik.

A működést analizálva beláthatjuk, hogy az órajel magas értékénél a D bemenet szintje beíródik az első DG tárolóba, de eközben a második tároló kimenete változatlan, hiszen a G bemenetére ’0’ jut. Az órajel lefutásakor az első fokozat ’átlátszatlanná’ válik, a kimenetének értéke azonban átíródik a második tároló kimenetére. A beírás egy órajelciklussal 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. A D-MS tároló működését tehát az órajel két fázisra bontja: az első a D bemenet mintavételezése és a mintavételezett érték tárolása, miközben a Q kimenet változatlan, őrzi az utolsóként beállt értéket. A második a Q kimenetre a mintavételezett érték rákapcsolása és tárolása, mi-közben a D bemenet változásai már hatástalanok maradnak.



4.4.3.1.a ábra[1]

D-MS tároló aszinkron DG tárolók és központi vezérlőjel(órajel) alkalmazásával

A D-MS tároló igazságtáblája a 4.4.3.1.b ábrán látható:

egyszerű igazságtábla

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

4.4.3.1.b ábra[1]

A D-MS tároló igazságtáblája

Az állapottáblában az órajel-vezérelt tárolóknál a következő kimeneti állapot jelölésére az (*n+1*) felső indexet szokás alkalmazni, mivel az aktuális kimeneti állapotot az *n*-edik órajel ciklus eredményének tekintjük, és így *n* felső indexet alkalmazunk a jelölésre. Azaz az aktuális kimenet jele Q^n, a következőé Q^{n+1} .

A 4.4.3.1.b ábrán látható, hogy a D-MS tároló következő kimeneti állapota nem függ az aktuális kimeneti állapottól, csakis a D bemenettől. Az összetett igazság-tábla tehát azonos az egyszerűvel.

A tároló –adott előírás szerinti- működtetéséhez szükségünk van egy ún. *vezérlési táblára*, amely segítséget nyújt az alkalmazott MS tárolónak a hálózat specifikáció szerinti viselkedésének biztosításához. A D-MS tároló vezérlési táblája az összetett igazságtábla alapján származtatható. A 4.4.3.1.c ábrán 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.

.

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

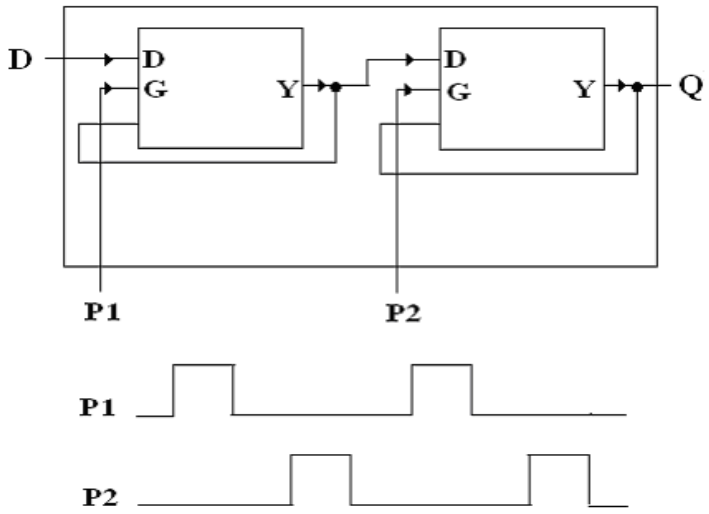
4.4.3.1.c ábra[1]

A D-MS tároló vezérlési táblájának származtatása az összetett igazságtáblából

4.4.3.2 Órajel vezérlések a MS tárolókban

Master-slave tárolókban a két fokozat működésének időbeli elválasztására gyakran kétféle vezérlőjel kialakítást is alkalmaznak.

Az egyik az ún. *kétfázisú, nem átlapolt órajel* használata. Erre látunk példát D-MS tárolóban a 4.4.3.2.a ábrán:

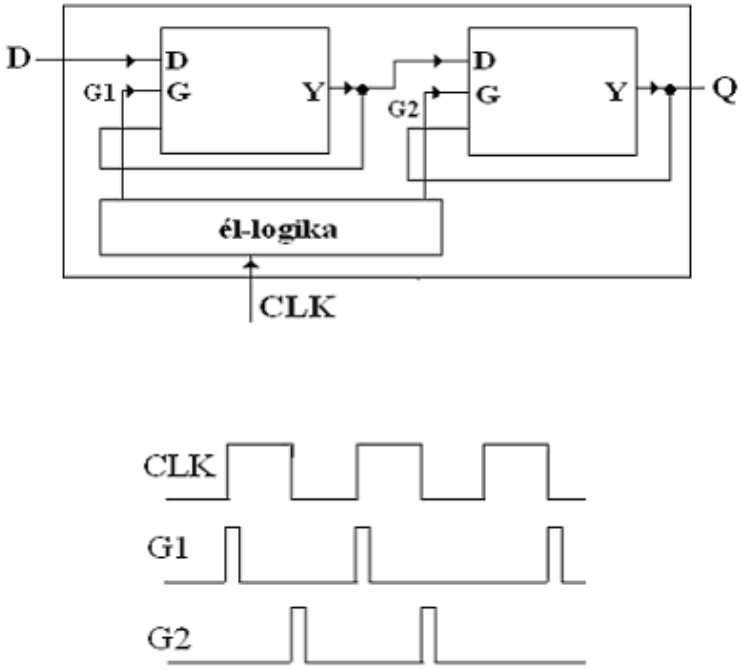


4.4.3.2.a ábra[1]

Kétfázisú órajel használata D-MS tárolóban

A két, P1 és P2 órajel-impulzus időben nem átlapoltan érkezik a mester és a szolga fokozatra, ezzel biztosítva a biztonságos elválasztást.

A másik megoldás az ún. *élvezérelt órajel* alkalmazása. Ennek kialakítását láthatjuk a 4.4.3.2.b ábrán, D-MS tároló esetében:



4.4.3.2.b ábra[1]

D-MS flip-flop élvezérelt órajellel

Az alkalmazás előnye, hogy maga a vezérlő órajel(CLK) egyfázisú, és egy speciális, néhány logikai kapuból álló kiegészítő áramkör alkalmazásával állítják elő az él-logikának megfelelő impulzusokat (G1 és G2). Az ábrán szereplő élvezérlés esetünkben azt jelenti, hogy az órajel(CLK) felfutó élére az egyik(mester), a lefutó élre a másik(szolga) tároló működik. Az ilyen él-logikával megvalósított MS tárolókat *lefutó élre működő tárolóknak* nevezik, és a gyártók ezt a specifikációban szereplő, a működési fázisokat megadó ún. funkciótáblában (function table) az órajelnél egy '↓' szimbólummal szokták jelölni. *Felfutó élre működő MS tárolók* esetében az élvezérlés éppen fordított, ilyenkor a funkciótáblában az órajelhez rendelt szimbólum is ennek megfelelően a '↑' jel (4.4.3.4.b ábra). Az élvezérelt tárolókat más néven „flip-flop"-oknak is nevezik.

A fentiekben bemutatott órajel ütemező megoldások természetesen más (JK,T) MS tárolók esetében is azonos kialakítással történnek.

4.4.3.3 JK-MS tároló

A JK-MS tároló egy olyan mester-szolga tároló, melynek egy J és egy K adatbemenete és egy Q adatkimenete van. A tároló működését leíró egyszerű és összetett igazságtáblát a 4.4.3.3.a ábrán láthatjuk:

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

4.4.3.3.a ábra[1]

A JK-MS tároló igazságtáblái

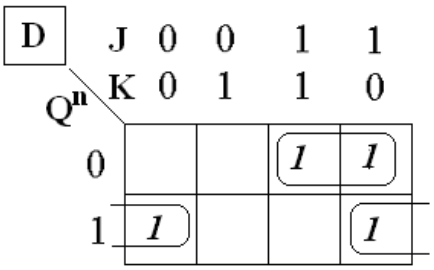
JK-MS tároló megvalósításához kézenfekvő megoldásnak tűnik a D-MS tároló felhasználása, hiszen annak vezérlési táblája (ami megfeleltethető az egyszerű igazságtáblájának) alapján a D bemenet vezérlése könnyen felírható: egyszerűen behelyettesítve a JK-MS összetett igazságtáblájában a következő állapotot meghatározó Q^{n+1} helyébe a D bemenet vezérlését (4.4.3.3.b ábra), a táblázatból kiolvashatjuk a szükséges hálózatot leíró algebrai függvényalakot (4.4.3.3.c ábra). Ehhez utolsó lépésként a szükséges Karnaugh táblát kitöltjük az igazságtábla alapján, és a kijelölt prímimplikánsokból felírt függvényalakból származtatható a D-MS tárolóval realizálható NAND-NAND hálózat (4.4.3.3.d ábra).

Megjegyzés: a kitöltött Karnaugh táblát tanulmányozva, korábbi tanulmányaink alapján feltételezhetjük, hogy statikus hazárd veszélye állhat fent a kijelölt két prímimplikáns ($J\overline{Q^n}$, $\overline{K}Q^n$) átmeneti mintermjei között. A hazárdmentesítés azonban ebben az esetben nem szükséges, mivel szabályszerűen az órajelet csak akkor lehet felemelni, ha a D-t meghajtó hálózat tranziensei befejeződtek, és D már nyugalmi állapotba került.

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

4.4.3.3.b ábra[1]

D-MS tároló vezérlése a JK-MS megvalósításához



4.4.3.3.c ábra[1]

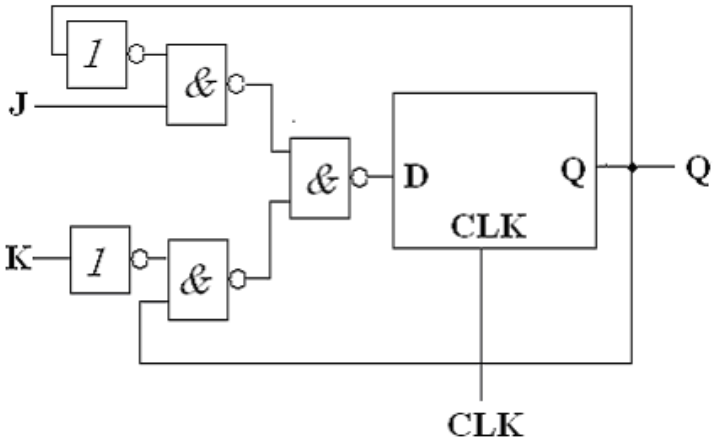
A D-MS tároló felhasználásával megvalósított JK-MS tároló Karnaugh táblája a kijelölt prímimplikánsokkal($J\overline{Q^n}$, $\overline{K}Q^n$)

A D bemenetre felírt algebrai alak:

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

illetve ennek NAND-NAND változata:

$$D = \overline{\overline{(J\overline{Q^n})} \cdot \overline{\overline{K}Q^n}}$$



4.4.3.3.d ábra[1]

JK-MS tároló NAND-NAND realizációja D-MS tároló felhasználásával

A JK-MS tároló vezérlési táblájának az összetett igazságtáblából származtatható alakja a 4.4.3.3.e ábrán látható. Az összetett igazságtáblában minden lehetséges változáshoz van egy olyan bemenet a kettő közül, amelynek szintje lehet ’0’ is és lehet ’1’ is, azaz közömbös. Ezek a „don’t care” bejegyzések a következő állapot kombinációt generáló kombinációs hálózat egyszerűsítéséhez vezetnek:

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

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

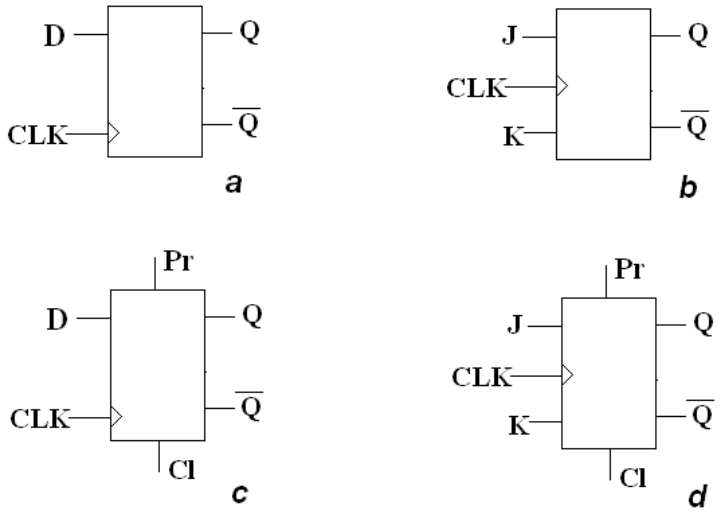
4.4.3.3.e ábra[1]

A JK-MS tároló vezérlési táblájának származtatása az összetett igazságtáblából

4.4.3.4 Mester-szolga tárolók segédbemenetei

A MS tárolók esetében a szükséges vagy előírt kezdeti állapot beállítását ún. segédbemenetekkel (’Pr’; ’Cl’ vagy ’Clr’) lehet elvégezni. Ezek legtöbbször a gyártók által eleve kialakításra kerülnek egy tárolóelemet tartalmazó integrált áramköri tokban, de ennek hiányában – egy, az adatbemenetekre kapcsolódó - kiegészítő áramkör segítségével is biztosítható a funkció (ennek megvalósításával a későbbiekben részletesen is foglalkozunk)(4.4.3.4.a ábra). A ’preset’ (’Pr’) bemenet a tárolót a funkcionális bemenetektől függetlenül logikai magas szintre, a ’clear’ (’Cl’ vagy ’Clr’) a funkcionális bemenetektől függetlenül logikai alacsony szintre állítja.

Megjegyzés: a segédbemenetek egyes flip-flopoknál az órajeltől függetlenül állítják be a kimenetet, másoknál az órajel valamelyik élének hatására (ez az adott tároló működését specifikáló funkciótáblából egyértelműen kiolvasható(4.4.3.4.b ábra)). Előbbieket aszinkron segédbemeneteknek, utóbbiakat szinkron segédbemeneteknek nevezzük.



4.4.3.4.a ábra[1]

D-MS és JK-MS tárolók szokásos szimbólumai segédbemenetekkel, és azok nélkül

INPUTS				OUTPUTS	
PRE	CLR	CLK	D	Q	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H	H
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q^n	$\overline{Q}^n$

4.4.3.4.b ábra

A Texas Instruments által gyártott HC(T)7474 tokozású D-MS flip-flop funkciótáblázata

4.5 Szinkron sorrendi hálózatok tervezése mintapéldákon keresztül I.

Ebben a fejezetben mintapéldákon keresztül mutatjuk be a szinkron hálózatok tervezési eljárásait, megismerjük az egyes tervezési lépések sorrendjét, valamint a megtervezett hálózatok logikai kapuszintű megvalósítását.

4.5.1 Szinkron sorrendi hálózat tervezése: 1. mintafeladat

Példa:

adott egy kétbemenetű (X_1, X_2) és egy kimenettel (Z) rendelkező szinkron sorrendi hálózat. A hálózat az első X_1=X_2 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 JK-MS tárolókkal!

4.5.1.1 A feladat Mealy-típusú hálózat modelljének megoldása

A feladat megoldása során egy szisztematikus módszert követünk, mely eljárás általánosságban használható a sorrendi hálózatok egy adott specifikáció alapján történő elvi és gyakorlati megvalósítására. Ennek általános lépései a következők:

1. *Állapot-átmeneti gráf felrajzolása.*
2. *Előzetes szimbolikus állapottábla felvétele.*
3. *Összevont szimbolikus állapottábla megszerkesztése.*
4. *Kódolt állapottábla elkészítése.*
5. *A specifikációra érvényes vezérlési tábla elkészítése (tárolók használata esetében).*
6. *A szekunder változó(k) és a kimenet(ek) függvényeinek Karnaugh tábla segítségével történő megadása.*
7. *Kezdeti állapotról történő gondoskodás.*
8. *Realizáció logikai kapukkal (és tárolókkal).*

Megjegyzés: a tervezés lépései egyes esetekben kiegészülnek egyéb lépésekkel is (pl. kritikus versenyhelyzet elemzése a szekunder változók kódolásánál, lényeges hazárdok vizsgálata stb.) A tervezés teljes, konkrét lépéssorozatát mindig az adott specifikáció illetve hálózattípus határozza meg. Ezekre látunk példákat majd a következő fejezetekben.)

Megoldás:

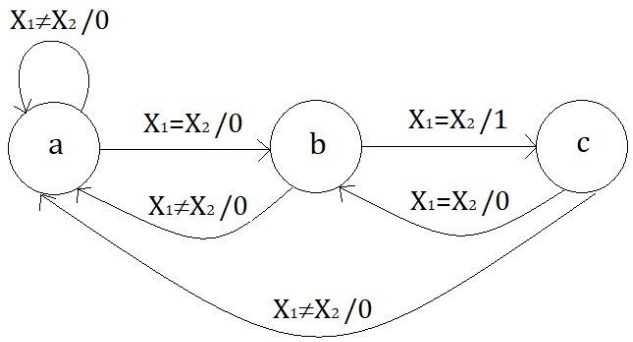
1. *Az állapot-átmeneti gráf felrajzolása* (4.5.1.1.a ábra)

Első lépésként célszerű a hálózat működését egy ún. állapot-átmeneti gráfon, vagy röviden állapotgráfon szemléltetni. Az állapotgráf csomópontjai szimbolikus (legtöbbször betű 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 órajelciklus hatására 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. (*Megjegyzés: a Moore-típusú hálózatok állapotgráfja ettől abban különbözik, hogy a gráf csomópontjaihoz, azaz az állapotokhoz rendeljük hozzá a kimeneti kombinációkat.)*

Az állapotgráf elkészítésének első mozzanataként határozzunk meg, illetve vegyünk fel egy kezdőállapotot, amibe a hálózat bekapcsoláskor kerül. Nevezzük ezt el '*a*' állapotnak.

Az ezután jelentkező első órajelciklus hálózatra gyakorolt hatásának eredménye attól függ, milyen bemeneti bitkombináció érkezik. Ha $\boldsymbol{X_1}$ nem egyenlő $\boldsymbol{X_2}$-vel ($\boldsymbol{X_1 \neq X_2}$, azaz *01* vagy *10*), ez a vizsgálat szempontjából azt jelenti, hogy nem kell előkészíteni a kimenetet a szükséges 0→1 váltáshoz, tehát mindaddig, amíg *01* vagy *10* érkezik a bemenetre, maradunk a kezdeti állapotban(a), és a kimenet, $\boldsymbol{Z}$ alacsony szinten marad. Azt viszont, hogy megérkezik az első $\boldsymbol{X_1 = X_2}$ egyezőség (*00* vagy *11*), a hálózatnak meg kell jegyeznie, hogy a második együttállást jelezni tudja a kimeneten. Vagyis az *a* állapotból a *00* vagy az *11* bemeneti kombinációkra egy másik, *b* állapotba kerül a hálózat, és a *b* szimbólum jelentése, hogy megjött az első, számunkra kedvező bemeneti bitkombináció. A $\boldsymbol{Z}$ kimenet értéke az *a*-ból *b*-be tartó átmenet alatt természetesen *0* marad, hiszen ez még csak az első kedvező bemeneti kombináció.

Ha *b* állapotban tartózkodunk, és a következő órajelciklus a *01* vagy *10* bemeneti kombinációt fogadja, akkor a hálózatot vissza kell irányítani a kezdeti *a* állapotba, a $\boldsymbol{Z}$ kimenet pedig továbbra is alacsony logikai értéken van, a vizsgálat újból kezdődik elölről. Ezzel szemben, ha $\boldsymbol{X_1}$ és a $\boldsymbol{X_2}$ bemeneteken a *00* vagy az *11* bitkombináció érkezik, hálózatunk egy újabb, ezúttal *c* állapotba kerül, és a $\boldsymbol{Z}$=1 értékkel jelzi, hogy megjött a második együttállás. A *c* állapotból való elágazásnál, ha *01* vagy *10* érkezik a bemeneteken, a kezdőállapotba(a) kell mennünk, hiszen újra az első együttállásra kell várni. Érkezhet azonban *00* vagy *11* is a következő órajel mintavételezéskor, és akkor máris a *b* állapotba kell mennünk. Természetesen mindkét esetben a kimenet alacsony szintre vált vissza.



4.5.1.1.a ábra

Az 1. mintafeladat Mealy-típusú állapot-átmeneti gráfja

2. *Az előzetes, szimbolikus állapottábla felvétele* (4.5.1.1.b ábra)

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

Megjegyzés: az állapotgráf jól szemlélteti grafikus ábrázolásmódon keresztül az adott hálózat mindenkori viselkedését, de a konkrét tervezési folyamathoz nem feltétlenül szükséges, el is hagyható. Amennyiben a specifikáció viszonylag egyszerű és könnyen értelmezhető kritériumokat tartalmaz, akkor elég a megoldás első lépéseként azonnal felírni az előzetes szimbolikus állapottáblát. Ha azonban konkrét áramköri realizáción (vagy szimulációs program segítségével) vizsgáljuk a megvalósított hálózatot, a működés egymást követő fázisai az állapotgráfon keresztül könnyedén leellenőrizhetőek.

aktuális állapot	**következő állapot/kimenet**	
	$X_1 \neq X_2$	$X_1 = X_2$
a	$a/0$	$b/0$
b	$a/0$	$c/1$
c	$a/0$	$b/0$

4.5.1.1.b ábra

Az 1. mintafeladat előzetes szimbolikus állapottáblája(Mealy-modell)

Ennél a feladatnál egyetlen gráf-él, vagy a tábla egyetlen oszlopa lényegében két bemeneti bitkombinációt képvisel: $\boldsymbol{X_1 \neq X_2}$ (*01;10*) és $\boldsymbol{X_1 = X_2}$(*00;11*). Természetesen megadható lett volna mindkettő állapotonként négy éllel, illetve négy oszloppal is, de célszerű kihasználni az ilyen és ehhez hasonló egyszerűsítési lehetőségeket.

Vegyük észre: a hálózat elágazásait tulajdonképpen egyetlen jel vezérelheti! Ez pedig analógiát mutat egy nevezetes kétváltozós függvénnyel, ami nem más, mint az EXNOR

függvény, illetve ekvivalencia kapu. Vezessük be tehát az *E* logikai változót, amelyre így igaz, hogy:

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

3. *Összevont szimbolikus állapottábla megszerkesztése* (4.5.1.1.c ábra)

Az állapotgráf szerkesztésekor előfordulhat, 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. Ez azonban nem jelent problémát az adott feladat végső megoldását tekintve, hiszen az első, előzetes állapottábla állapotainak számát szisztematikus módszerekkel minimalizálni lehet. Ennek általános megfontolásaival egy későbbi fejezetben részletesen foglalkozunk, most azonban egy magától értetődő kritériumot már megfogalmazhatunk arra vonatkozóan, hogy mikor nem kell megkülönböztetni két állapotot az előzetes állapottáblán: *az előzetes állapottábla két állapotát nem kell megkülönböztetni, ezért azok összevonhatók, ha bemeneti kombinációnként megegyeznek a hozzájuk rendelt kimeneti kombinációk, és bemenő kombinációnként ugyanarra a következő állapotra vezetnek.*

Keressünk ez alapján összevonható állapotokat az előzetes szimbolikus állapottáblánkban! Az állapottábla alapos vizsgálatából kiderül, hogy a fenti feltétel az *a* és a *c* állapotokra teljesül, hiszen a mindkettőből kiinduló bemeneti kombinációkhoz ugyanazok a kimeneti értékek tartoznak: *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*. *Következtetés*: az *a* és *c* állapot megkülönböztetése felesleges, azokat össze lehet vonni. Legyen az új, összevont állapotunk jele *ac* ! Ez alapján elkészíthetjük az összevont szimbolikus állapottáblánkat, amely most már csak a feltétlenül szükséges állapotokat tartalmazza. Ebben az összevont állapotok kerülnek bejegyzésre mindazokon a helyeken, ahol az előzetes táblában az összevont állapotok valamelyike szerepelt.

Megjegyzés: amennyiben nincsenek a feltételrendszer szerint összevonható állapotok, úgy a következő, kódolt állapottáblát előállító lépéssorozat alapja az előzetes szimbolikus állapottábla.

aktuális állapot	következő állapot/kimenet	
	$\bar{E}$	E
ac	$ac/0$	$b/0$
b	$ac/0$	$ac/1$

4.5.1.1.c ábra

Az 1.mintafeladat összevont szimbolikus állapottáblája(Mealy-modell)

4. *A kódolt állapottábla elkészítése* (4.5.1.1.d ábra)

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 állapotkód hossza, azaz a szükséges MS tárolók száma, N_a pedig a kódolt összevont állapottáblán az állapotok száma. A kódolt állapottábla elrendezése tehát nagyon hasonló, mint az összevont (vagy előzetes) szimbolikus állapottábláé: egyszerűen be kell helyettesíteni az állapotok helyébe a hozzárendelt kódo(ka)t. Példánkban az összevont szimbolikus állapottáblában két állapotot kell megkülönböztetni. Ezekhez következő lépésként rendeljünk egy Q másodlagos változót, és a hozzárendeléskor mindegy, hogy melyik állapothoz rendeljük a Q=0, és melyikhez a Q=1 értéket. Esetünkben az *ac* állapothoz a Q=0, a *b* állapothoz a Q=1 bináris kódértéket rendeltük:

kódolt aktuális állapot, Q^n	kódolt következő állapot/kimenet	
	$\bar{E}$ Q^{n+1}/Z	E Q^{n+1}/Z
0	*0/0*	*1/0*
1	*0/0*	*0/1*

4.5.1.1.d ábra

Az 1. mintafeladat kódolt állapottáblája(Mealy-modell)

5. *A vezérlési tábla elkészítése* (4.5.1.1.f ábra)

A tervezési folyamatnak ezen a pontján – amennyiben nincs külön előírás az alkalmazandó tárolók típusáról - 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 JK-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 saját vezérlési tábláira, nevezetesen arra, hogy adott kimeneti változáshoz milyen szinteket kell kapcsolnunk a bemenetekre.

Mivel a feladat specifikációja JK-MS tároló(k) alkalmazását írja elő, így ezúttal a JK-MS flip-flop saját vezérlési tábláját kell segítségül hívni. A vezérlés kódolt állapottáblából történő kiolvasásának folyamatára mutat példát a 4.5.1.1.e ábra:

kódolt aktuális állapot, Q^n	kódolt következő állapot/kimenet	
	$\bar{E}$ Q^{n+1}	E Q^{n+1}
0	*0/0*	*1/0*
1	*0/0*	*0/1*

4.5.1.1.e ábra

A vezérlés kiolvasásának folyamata: $Q^n \rightarrow Q^{n+1}$

A kiolvasást követően a 4.5.1.1.f. ábra mutatja a következő lépésként előállítandó táblákat. Mivel egyetlen szekunder változónk van (Q), ezért ennek megvalósításához egyetlen JK-MS tárolóra van szükségünk A baloldali, JK-MS saját vezérlési táblájából kiolvassuk azokat a szükséges J és K vezérléseket, melyek a kódolt állapottáblában meghatározott következő Q^{n+1} értékeket előállítják a tárolónk kimenetén. Az így felírt táblázat tartalmazza a most már a feladatra specifikált J és K bemeneti vezérléseket:

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

$\Rightarrow$

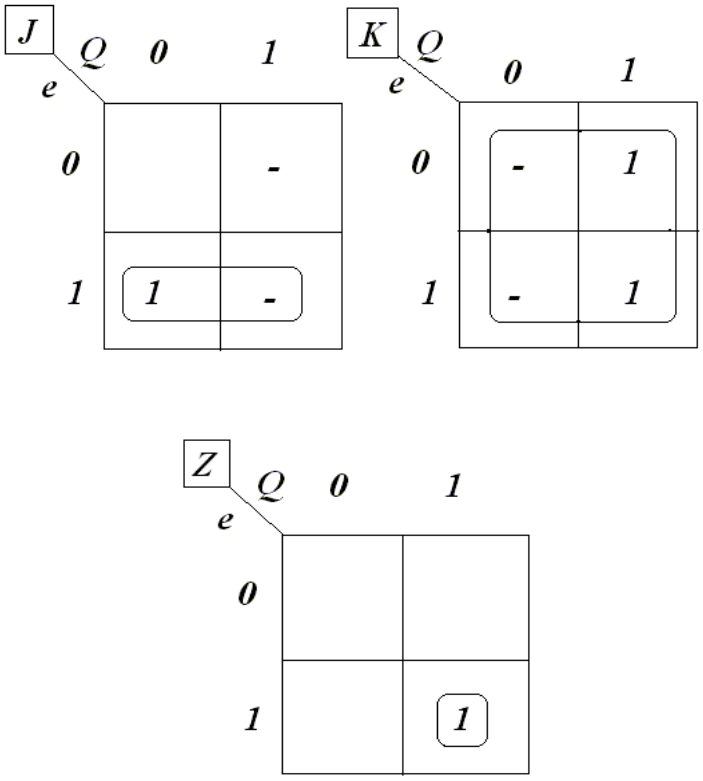
Q	$\bar{E}$ J K	E J K
0	*0 -*	*1 -*
1	*- 1*	*- 1*

4.5.1.1.f ábra

Az 1.mintafeladat JK flip-flopjának vezérlési táblája a kódolt állapottáblából és a tároló saját vezérlési táblájából származtatva(Mealy-modell)

6. *A szekunder változó(k) és a kimenet(ek) függvényeinek Karnaugh tábla segítségével történő megadás* (4.5.1.1.g ábra)

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 Karnaugh táblákat ennek alapján kell felvennünk. A kódolt állapottáblából már látható, hogy esetünkben igen egyszerű, kétváltozós Karnaugh táblákkal megadhatjuk a két kombinációs hálózat logikai függvényeit.



4.5.1.1.g ábra[1]

Az 1.mintafeladat Karnaugh táblái(Mealy-modell)

A táblázatokban a prímimplikánsok kijelölése után kialakult függvényalakok a következő algebrai eredményt adják:

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

7. *Kezdeti állapot beállítása*

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. A kezdeti állapot beállítását eredményező speciális bemenőjelet '*R*' (RESET) szimbólummal jelöljük. Példánkban a hálózat kezdeti állapotában (*a*) az alkalmazott JK-MS tárolónk kimenetén (a kódolt állapottábla szerint) alacsony szintet kell biztosítani, ezért az '*R*' kezdeti állapot beállító jelet a 'clear' bemenetre kell csatlakoztatni, és a nem használt 'preset' bemenetre konstans alacsony szintet kapcsolunk (4.5.1.1.h ábra).

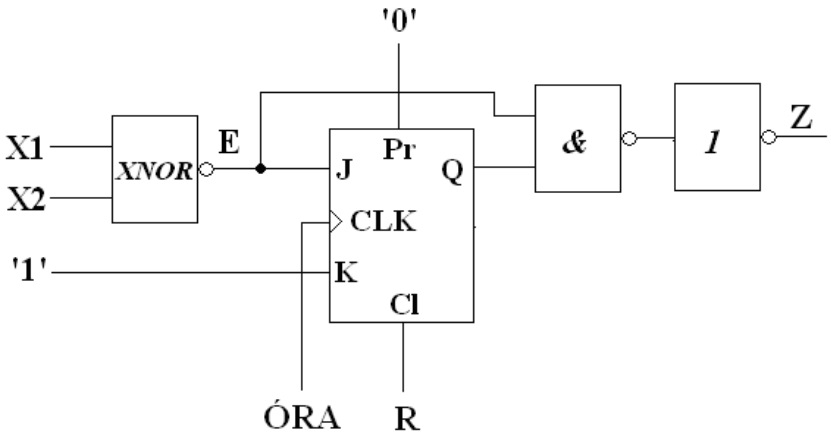
(*Megjegyzés: mindaddig, amíg a kezdeti beállítás más módszereit meg nem ismerjük, csak 'preset' és 'clear' bemenetekkel is rendelkező flip-flopokat használunk.*)

8. *A realizáció logikai kapukkal* (4.5.1.1.h ábra)

A kapott hálózat érdekessége, hogy a realizáció tárolójának kimenete nincs visszacsatolva a következő állapotot generáló hálózat bemenetére, holott a szinkron sorrendi hálózat általános modelljének bemutatásakor ennek lényegét hangsúlyoztuk. Fogadjuk el, hogy egy adott speciális funkció megvalósítása vezethet arra, hogy egyik vagy másik szekunder változó értékétől nem függenek egyik vagy másik flip-flop állapotváltozásai. A Mealy-modell szerint realizált hálózatunk sajátossága, hogy a flip-flop aktuális kimenetétől nem függ annak a következő állapota.

A felvázolt NAND-NAND realizáció alapját szolgáló algebrai függvényalakok:

$$J = E, \quad K = 1, \quad Z = \overline{\overline{QE}}$$

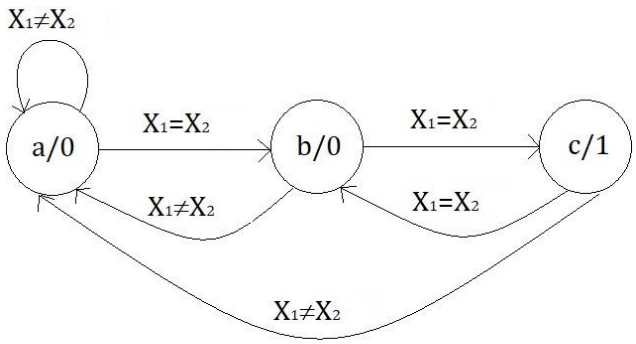


4.5.1.1.h ábra[1]

Az 1.mintafeladat Mealy-típusú NAND-NAND realizációja

4.5.1.2 A feladat Moore-típusú megoldása

A feladat Moore-típusú hálózattal történő megvalósítása során az előbbiekben megismert szisztematikus tervezési módszer – a lépéseket tekintve – sok hasonlóságot mutat a Mealy-modell szerint elvégzett tervezéssel, de lényegét tekintve mégis fontos különbséget mutat. Ezt a különbözőséget már az állapotgráfon az állapotok, valamint az állapot átmenetek jelölése is mutatja (4.5.1.2.a ábra): *Moore-modell esetén a kimeneti kombinációk csak az állapotok függvényei!*



4.5.1.2.a ábra

Az 1.mintafeladat Moore-típusú állapotgráfja

Ez már előrevetíti az állapottábla bejegyzéseinek Mealy-típusú vázlatához képesti különbözőségét (4.5.1.2.b ábra):

aktuális állapot/kimenet	**következő állapot**	
	$X_1 \neq X_2$	$X_1 = X_2$
$a/0$	a	b
$b/0$	a	c
$c/1$	a	b

4.5.1.2.b ábra

Az 1.mintafeladat Morre-modell alapján felállított előzetes szimbolikus állapottáblája

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.* Ez alapján a vizsgálat eredménye most azt mutatja, hogy ilyen párok nincsenek, tehát a Moore-típusú hálózatot három állapottal kell megterveznünk. Az állapotok száma alapján legalább két másodlagos változót kell bevezetnünk (Q_1; Q_2) azok bináris kóddal történő megkülönböztetéséhez.

Az előzetes - és immár végleges - szimbolikus állapottábla bejegyzései alapján megszerkeszthetjük a kódolt állapottáblát (4.5.1.2.c ábra). Az egyes állapotokhoz szabadon hozzárendelhetjük a szekunder változók bármely lehetséges bitkombinációját, így erre többféle megoldás is adódhat. Mivel azonban esetünkben csak három állapotot kell kétbites kombinációkkal megkülönböztetni, bármelyik párosítást is alkalmazzuk, mindenképpen marad egy kihasználatlan, „szabad” kombináció. Ez azt jelenti, hogy ehhez a nem használt, „megmaradó” kódhoz tartozó sorban a kimenet, valamint a következő állapotokat előíró vezérlés értékét közömbös bejegyzésként regisztrálhatjuk.

Az egyes állapotok megkülönböztetésére alkalmazzuk a következő kódolást:

	Q_1	Q_2
a :	0	0
b :	0	1
c :	1	0

A párosítás eredményeként adódik, hogy a nem használt kód az *11* szekunder változó bitkombináció, amelynek kódolt állapottábla szerinti sorába kerülnek a fent említett „don't care" bejegyzések.

kódolt aktuális állapot/kimenet	**kódolt következő állapot**	
	$\bar{E}$	E
$Q_1^n\ Q_2^n/Z$	$Q_1^{n+1}\ Q_2^{n+1}$	$Q_1^{n+1}\ Q_2^{n+1}$
0 0/0	*0 0*	*0 1*
0 1/0	*0 0*	*1 0*
1 0/1	*0 0*	*0 1*
1 1/-	*- -*	*- -*

4.5.1.2.c ábra

Az 1.mintafeladat kódolt állapottáblája (Moore-modell)

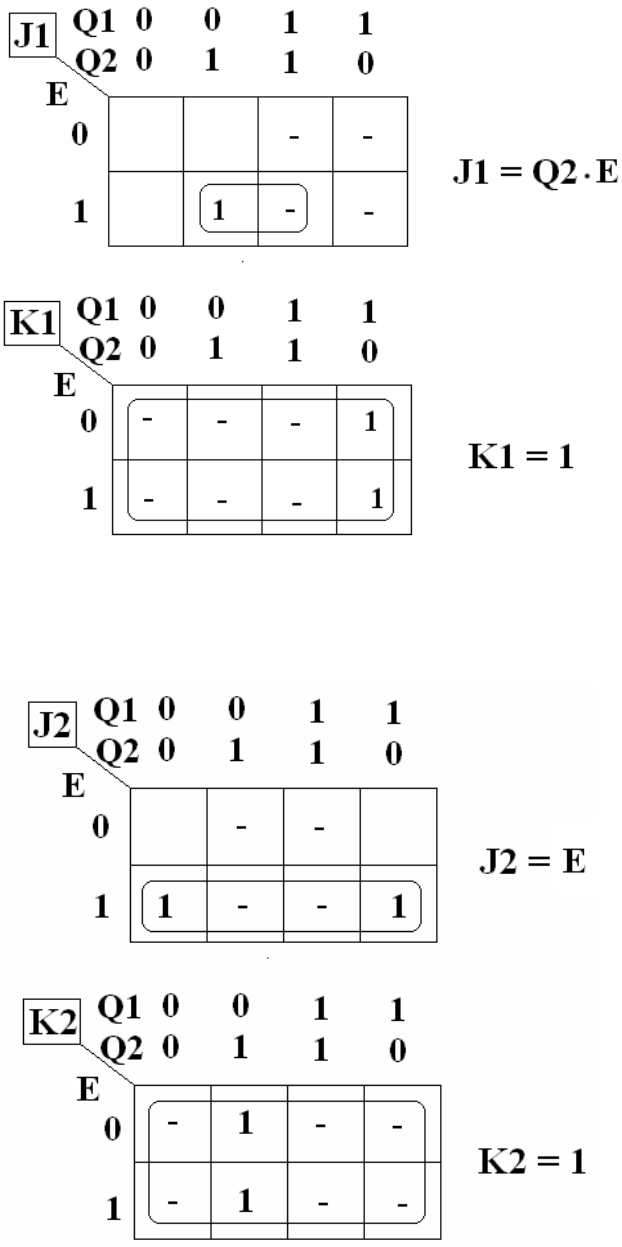
A kódolt állapottábla alapján – a JK-MS tároló saját vezérlési táblájának segítségével – megszerkeszthetjük a feladatra specifikált J és K vezérléseket, esetünkben két tároló bemeneteire (J_1K_1 és J_2K_2) definiálva (4.5.1.2.d ábra):

kódolt aktuális állapot/kimenet	**kódolt következő állapot**			
	$\bar{E}$		E	
$Q_1\ Q_2/Z$	J_1K_1	J_2K_2	J_1K_1	J_2K_2
0 0/0	*0 -*	*0 -*	*0 -*	*1 -*
0 1/0	*0 -*	*- 1*	*1 -*	*- 1*
1 0/1	*- 1*	*0 -*	*- 1*	*1 -*
1 1/-	*- -*	*- -*	*- -*	*- -*

4.5.1.2.d ábra

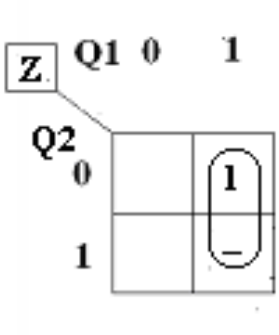
Az 1.mintafeladat JK flip-flopjainak vezérlési táblája a kódolt állapottáblából és a tároló saját vezérlési táblájából származtatva(Moore-modell)

A vezérlési táblázatból kitöltött Karnaugh táblák, az azokban felrajzolt függvény lefedések, valamint az ezekből származtatható algebrai függvényalakok a 4.5.1.2.e és 4.5.1.2.f ábrákon láthatóak. Az állapot kombinációk és a kimeneti kombinációk közötti egyértelmű függés szerint a ***Z*** kimenet csak a *c* állapotban, azaz a Q_1Q_2=*10* állapotkombináció fennállásakor lesz magas szinten.



4.5.1.2.e ábra[1]

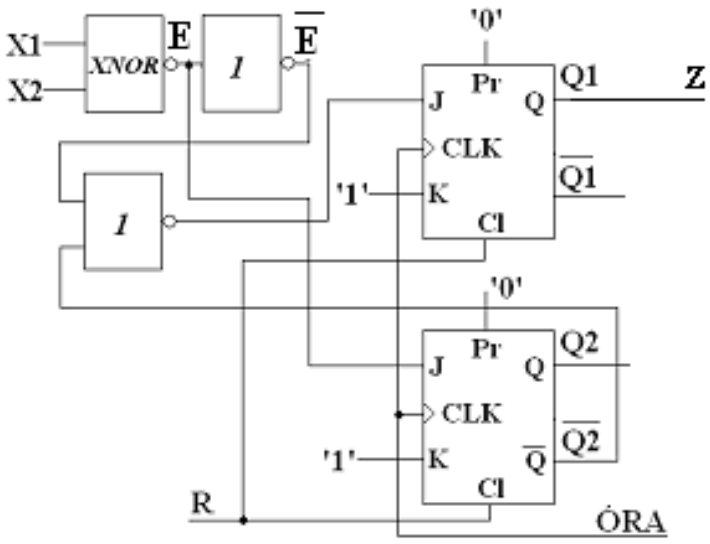
Az 1.mintafeladat J és K bemeneteinek Karnaugh táblái, és a lefedések segítségével megadott algebrai alakok(Moore-modell)



4.5.1.2.f ábra[1]

*Az 1.mintafeladat **Z** kimenetének Karnaugh táblája, és a lefedéssel megadott algebrai alak(Moore-modell)*

A Moore-modell egy lehetséges NAND-NAND realizációját láthatjuk a 4.5.1.2.g ábrán. A stabil kezdeti állapotban (a; $X_1 \neq X_2$) mindkét tároló kiementén alacsony szint kell, hogy megjelenjen, ezért az R beállítójelünket a törlő (CL) bemenetekre kötjük, a 'preset' pedig konstans alacsony szintet kap.



4.5.1.2.g ábra[1]

Az1.mintafeladat Moore-típusú NAND-NAND realizációja

4.5.2 Szinkron sorrendi hálózat tervezése: 2. mintafeladat

Példa:

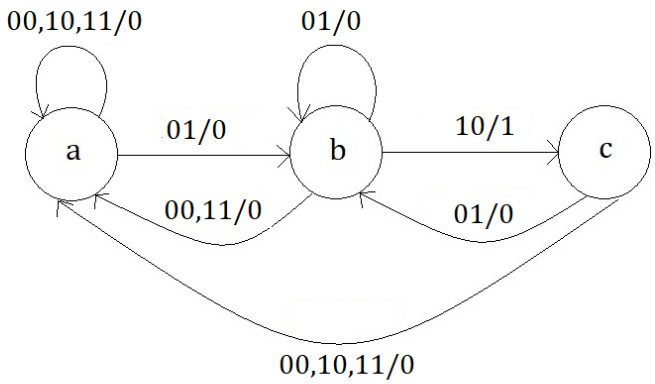
tervezzük meg a lehető legegyszerűbb változatban azt a 'preset' és 'clear' bemenetekkel is rendelkező élvezérelt D-MS tárolókkal felépített Moore-típusú szinkron sorrendi hálózatot, amelynek két bemenete (X_1, X_2) és egy kimenete (Z) van. A hálózat az órajel felfutása előtt fogadja a bemeneti kombinációkat. A hálózat Z kimenete akkor lesz '1', ha a bemenetre az '1 0' kombináció érkezik, de csak abban az esetben, ha az előző óraciklus által fogadott bemeneti bitkombináció '0 1' volt! A kezdeti állapothoz az X_1X_2 = '00' kombináció tartozik.

4.5.2.1 A feladat Mealy-modell szerinti megoldása

Megoldás:

1. *lépés: az állapotgráf felvétele.*

A 4.5.2.1.a ábrán láthatjuk a specifikáció alapján felrajzolt állapotgráfot:



4.5.2.1.a ábra

A 2.mintafeladat Mealy-modell szerinti állapotgráfja

2. *lépés: az előzetes szimbolikus állapottábla felvétele.*

A felrajzolt állapotgráf alapján megszerkesztjük az előzetes szimbolikus állapottáblát a Mealy-típusú tervezési eljárás sajátosságait figyelembe véve (4.5.2.1.b ábra).

aktuális állapot	**következő állapot/kimenet**			
	X_1X_2			
	00	*01*	*10*	*11*
a	$a/0$	$b/0$	$a/0$	$a/0$
b	$a/0$	$b/0$	$c/1$	$a/0$
c	$a/0$	$b/0$	$a/0$	$a/0$

4.5.2.1.b ábra

A 2.mintafeladat előzetes szimbolikus állapottáblája(Mealy-modell)

3. *lépés: az összevont szimbolikus állapottábla megszerkesztése.*

Az állapot összevonás szabályai alapján az előzetes szimbolikus állapottáblán az *a* és *c* állapotok nem megkülönböztethetőek, azaz összevonhatóak, így csupán két állapot marad az összevont szimbolikus állapottáblán: az *ac* (osztály) és a *b* (4.5.2.1.c ábra).

aktuális állapot	**következő állapot/kimenet**			
	X_1X_2			
	00	*01*	*10*	*11*
ac	$ac/0$	$b/0$	$ac/0$	$ac/0$
b	$ac/0$	$b/0$	$ac/1$	$ac/0$

4.5.2.1.c ábra

A 2. mintafeladat összevont szimbolikus állapottáblája(Mealy-modell)

4. *lépés: a kódolt állapottábla felvétele.*

Az összevont szimbolikus állapottábla alapján csupán két állapotot kell megkülönböztetni, ezért ehhez elég egyetlen szekunder változó(Q) bevezetése. Az *ac* állapothoz rendeljük a *Q=0* kódot, a *b*-hez pedig a *Q=1*-et. Ez alapján a kódolt állapottábla(4.5.2.1.d ábra):

aktuális állapot, Q	**következő állapot/kimenet**			
	X_1X_2			
	00	*01*	*10*	*11*
0	*0/0*	*1/0*	*0/0*	*0/0*
1	*0/0*	*1/0*	*0/1*	*0/0*

4.5.2.1.d ábra

A 2.mintafeladat kódolt állapottáblázata(Mealy-modell)

5. *lépés: a vezérlési tábla elkészítése.*

A feladat specifikációja D-MS flip-flopok alkalmazását írja elő, ez alapján tehát a vezérlési táblát az egyetlen szekunder változót reprezentáló D-MS tárolónkra kell adaptálni. A tároló saját bemeneti vezérlésének táblája alapján a 4.5.2.1.e ábrán látható a kódolt állapottáblából származtatott vezérlési tábla.

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

$\Rightarrow$

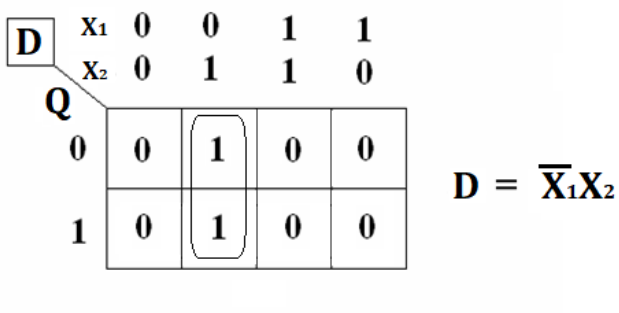
aktuális állapot, Q	D/Z			
	X_1X_2			
	00	*01*	*10*	*11*
0	*0/0*	*1/0*	*0/0*	*0/0*
1	*0/0*	*1/0*	*0/1*	*0/0*

4.5.2.1.e ábra

A 2.mintafeladat vezérlési táblája(Mealy-modell)

6. *lépés: a szekunder változó(D-MS adatkimenet) és a kimenet függvényeinek Karnaugh tábla segítségével történő megadása.*

Ebben a lépésben felrajzoljuk és kitöltjük a vezérlési tábla lapján a specifikációhoz illeszkedő Karnaugh táblákat(4.5.2.1.f és 4.5.2.1.g ábra):



4.5.2.1.f ábra[1]

*A 2.mintafeladat D-MS flip-flop **D** adatbemenetének Karnaugh táblája, és a táblázatból kiírt algebrai alakja(Mealy-modell)*

Z	X_1	0	0	1	1
	X_2	0	1	1	0
Q					
0		0	0	0	0
1		0	0	0	1

$$Z = X_1\overline{X_2}\,Q$$

4.5.2.1.g ábra[1]

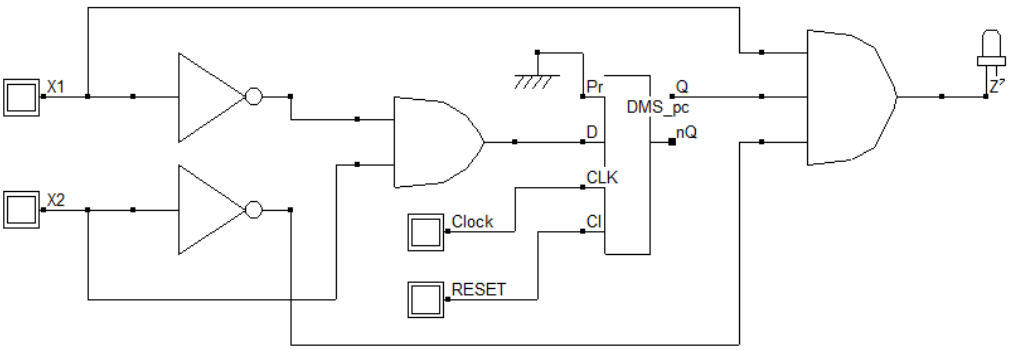
A 2.mintafeladat $\boldsymbol{Z}$ *kimenetének Karnaugh táblája, és a táblázatból kiírt algebrai alakja(Mealy-modell)*

7. *A kezdeti állapot beállításának ellenőrzése*

A stabil kezdeti *a* állapothoz tartozó $\boldsymbol{X_1X_2}$ *= '0 0'* kombináció mellett a $\boldsymbol{Z}$ kimeneten alacsony logikai szintnek kell megjelenni, ami a függvény szerint teljesül is. A kódolt állapottábla ide vonatkozó bejegyzése szerint a D-MS tárolónk kimenetén is ugyanezt az értéket kell biztosítanunk, ezért a 4.5.2.1.h ábrán látható realizációban az R('RESET') kezdőérték beállító jelünket a tároló törlő ('Cl') bemenetére kell kötnünk, a 'preset' segédbemenet ('Pr') pedig konstans alacsony szintet('ground') kap.

8. *Az áramköri realizáció elkészítése*

A feladat logikai kapuszintű realizációja a DSCH gyakorló programban elkészített sematikus áramköri rajzon (4.5.2.1.h ábra) látható:

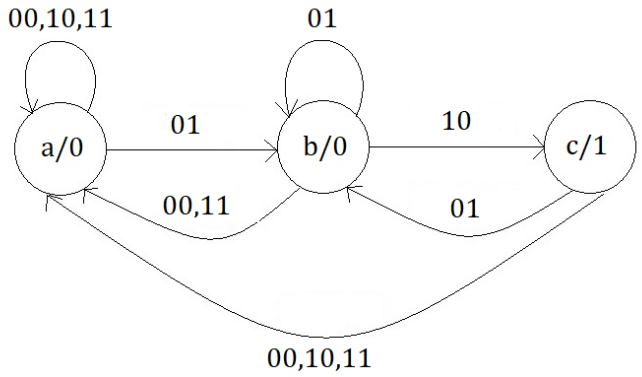


4.5.2.1.h ábra

A 2.mintafeladat Mealy-típusú áramköri realizációja a DSCH program segítségével

4.5.2.2 A feladat Moore-modell szerinti megoldása

A 2.mintafeladat Moore-típusú hálózattal történő megvalósításának első lépéseként rajzoljuk fel a specifikáció alapján az állapotgráfot (4.5.2.2.a ábra.)!



4.5.2.2.a ábra

A 2.mintafeladat Moore-modell szerinti állapotgráfja

Második lépésként szerkesszük meg a hozzá tartozó előzetes szimbolikus állapottáblát(4.5.2.2.b ábra)!

aktuális állapot/kimenet	**következő állapot**			
	X_1X_2			
	00	*01*	*10*	*11*
a/0	*a*	*b*	*a*	*a*
b/0	*a*	*b*	*c*	*a*
c/1	*a*	*b*	*a*	*a*

4.5.2.2.b ábra

A 2.mintafeladat előzetes szimbolikus állapottáblája(Moore-modell)

Következő lépésként vizsgáljuk meg, hogy az állapot összevonás szabályai alapján tudjuk-e az állapotok számát csökkenteni a táblán! Az elemzés ugyan első ránézésre azt mutatja, hogy a kimenet szempontjából az *a* és *b* állapotok összevonhatóak lennének, de sajnos a második feltétel nem teljesül, hiszen nem minden bemenő bitkombinációnként egyeznek meg a következő állapotok: $\boldsymbol{X_1X_2}$ = *'1 0'* esetében *a* aktuális állapotban maradunk stabilan, *b* aktuális állapotból viszont a *c* állapot felé indulunk. Mindez azt jelenti, hogy a kódolt állapottábla elkészítésének alapja ezúttal az előzetes szimbolikus állapottáblánk.

Mivel három állapotot kell megkülönböztetnünk, így két szekunder változó bevezetése szükséges a bináris kódoláshoz, mely legyen a következő:

	Q_1	Q_2
a :	0	0
b :	0	1
c :	1	0

Ebből pedig a 4.5.2.2.c ábrán látható kódolt állapottáblát készíthetjük el:

kódolt aktuális állapot/kimenet $Q_1^n Q_2^n / Z$	**kódolt következő állapot** $X_1 X_2$			
	00	*01*	*10*	*11*
	$Q_1^{n+1} Q_2^{n+1}$	$Q_1^{n+1} Q_2^{n+1}$	$Q_1^{n+1} Q_2^{n+1}$	$Q_1^{n+1} Q_2^{n+1}$
0 0/0	*0 0*	*0 1*	*0 0*	*0 0*
0 1/0	*0 0*	*0 1*	*1 0*	*0 0*
1 0/1	*0 0*	*0 1*	*0 0*	*0 0*
1 1/-	*- -*	*- -*	*- -*	*- -*

4.5.2.2.c ábra

Az 2.mintafeladat kódolt állapottáblája (Moore-modell)

A $\boldsymbol{Q_1 Q_2}$ *= '1 1'* kódkombinációt nem használjuk fel, így a táblázatban a hozzá tartozó bejegyzésekhez a 'don't care' ('-') jelölés kerül.

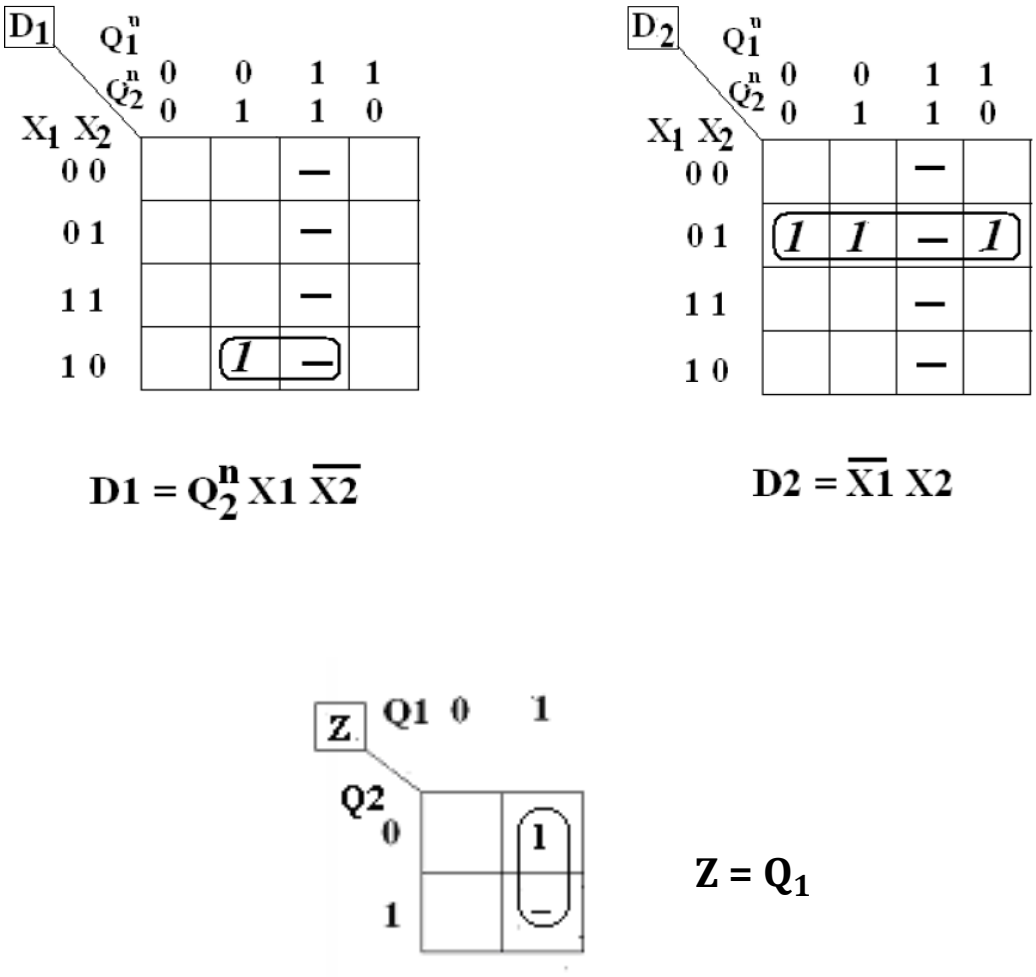
Mivel D-MS tároló felhasználásával kell megoldani a feladatot, ezért erre specifikáljuk a vezérlési táblát. Korábbi tanulmányainkból tudjuk, hogy a D-MS flip-flop alkalmazása esetén a vezérlési táblába a D bemenet vezérlési celláiba csak egyszerűen be kell másolni a kódolt állapottáblában megadott 'következő állapot'-kódot, így a 4.5.2.2.d ábrán látható táblázatot kapjuk. Magától értetődő, hogy a két szekunder változó két D-MS flip-flop (D_1 ; D_2) használatát feltételezi:

kódolt aktuális állapot/kimenet $Q_1^n\ Q_2^n/Z$	X_1X_2			
	00	*01*	*10*	*11*
	D_1D_2	D_1D_2	D_1D_2	D_1D_2
0 0/0	*0 0*	*0 1*	*0 0*	*0 0*
0 1/0	*0 0*	*0 1*	*1 0*	*0 0*
1 0/1	*0 0*	*0 1*	*0 0*	*0 0*
1 1/-	*- -*	*- -*	*- -*	*- -*

4.5.2.2.d ábra

A 2.mintafeladat vezérlési táblája(Moore-modell)

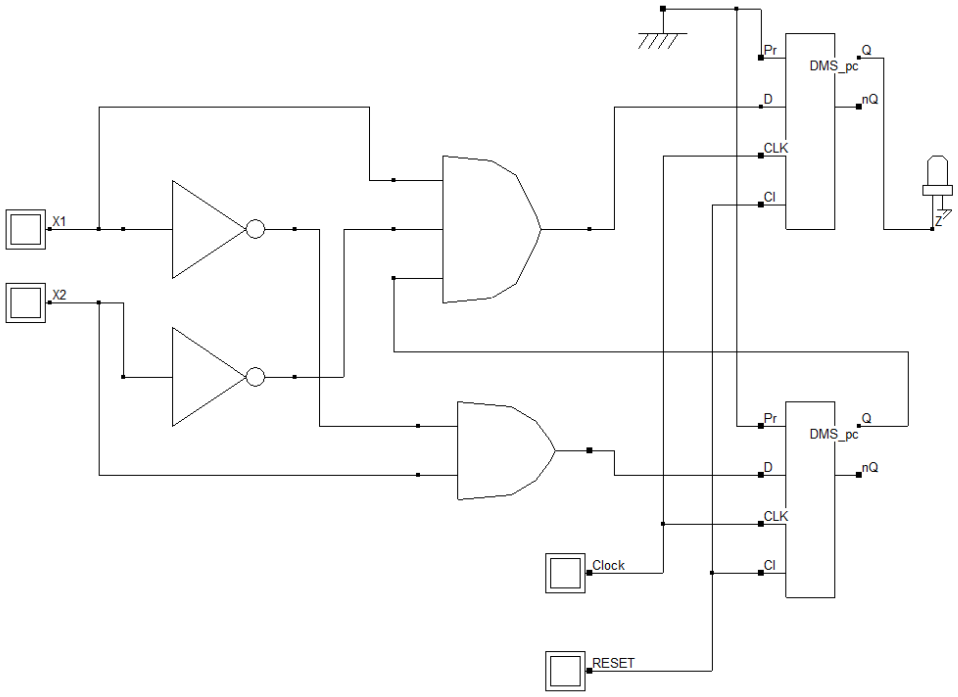
A D-MS tárolók és a kimenet függvényének felírásához szerkesszük meg a szükséges Karnaugh táblákat (4.5.2.2.e ábra):



4.5.2.2.e ábra[1]

*A 2.mintafeladat D-MS flip-flop **D** bemeneteinek, és a **Z** kimenetnek a Karnaugh tábla alapján kiírt algebrai függvényalakja(Moore-modell)*

A Moore-tipusú kapurealizáció a 4.5.2.2.f ábrán látható:



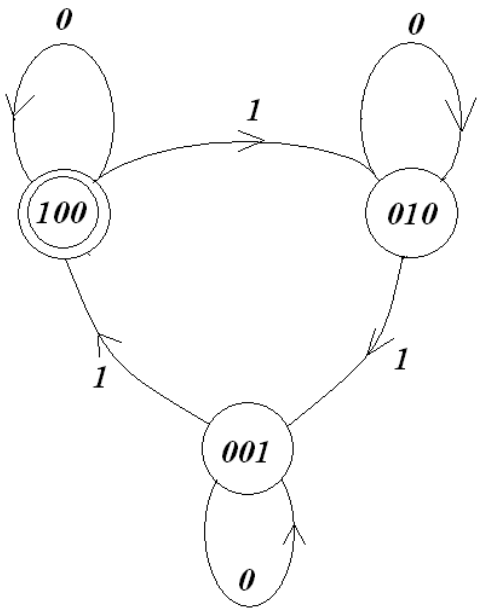
4.5.2.2.f ábra

A 2.mintafeladat Moore-típusú áramköri realizációja a DSCH program segítségével

4.5.3 Szinkron sorrendi hálózat tervezése: 3.mintafeladat

Példa:

tervezzük meg szinkron 'preset' és 'clear' bemenetekkel is rendelkező, élvezérelt D-MS tárolókkal a 4.5.3.a ábrán látható állapotgráf szerinti állapot-kimenetű szinkron sorrendi hálózatot a lehető legegyszerűbb változatban! A kezdeti állapotot a gráfon dupla kör jelzi.



4.5.3.a ábra[1]

A 3.mintafeladat állapotgráfja

4.5.3.1 A feladat megoldása Karnaugh táblás egyszerűsítés segítségével

A feladat specifikációjában egy új meghatározással találkozunk: az *állapot-kimenetű hálózat* fogalmával.

Állapot-kimenetű hálózat:

ha megnézzük a feladathoz tartozó állapotgráfot, akkor megállapíthatjuk, hogy a korábbi jelölésrendszerek (Mealy, Moore) alapján a kimenet értékére vonatkozó (külön) feljegyzést nem látunk. A gráfon most csak a bemenet (egy bit), illetve az egyes állapotokhoz rendelt hárombites kódok értékei vannak feltüntetve. Ilyenkor a kimenet értéke(i) maga az állapot, azaz maga az állapotot reprezentáló bináris kódszó. Ez egyben azt is jelenti, hogy az adott hálózatnak annyi kimenete van, ahány bit határozza meg az egyes állapotok kódját. Az ilyen struktúrájú sorrendi hálózatokat állapot-kimenetű hálózatoknak nevezzük.

Mivel a specifikációban a megadott állapotgráfon már a kódolt állapotokkal jellemzett hálózat egyes (adott pillanatbeli) működési fázisait láthatjuk, ezért kézenfekvő, hogy a feladat megoldását a kódolt állapottábla felvételével kezdjük (4.5.3.1.a ábra):

kódolt aktuális állapot,			kódolt következő állapot					
			X = 0			X = 1		
Q_1^n	Q_2^n	Q_3^n	Q_1^{n+1}	Q_2^{n+1}	Q_3^{n+1}	Q_1^{n+1}	Q_2^{n+1}	Q_3^{n+1}
1	0	0	1	0	0	0	1	0
0	1	0	0	1	0	0	0	1
0	0	1	0	0	1	1	0	0
0	0	0	-	-	-	-	-	-
0	1	1	-	-	-	-	-	-
1	0	1	-	-	-	-	-	-
1	1	0	-	-	-	-	-	-
1	1	1	-	-	-	-	-	-

4.5.3.1.a ábra

A 3.mintafeladat kódolt állapottáblája

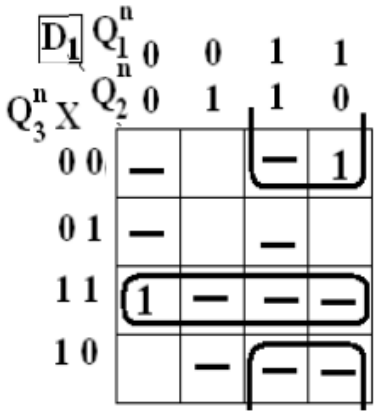
D-MS tárolók használata esetén a szekunder változókhoz rendelt flip-flopok vezérlési táblája a 4.5.3.1.b ábra szerint adódik:

kódolt aktuális állapot,			kódolt következő állapot					
			X = 0			X = 1		
Q_1^n	Q_2^n	Q_3^n	D_1	D_2	D_3	D_1	D_2	D_3
1	0	0	1	0	0	0	1	0
0	1	0	0	1	0	0	0	1
0	0	1	0	0	1	1	0	0
0	0	0	-	-	-	-	-	-
0	1	1	-	-	-	-	-	-
1	0	1	-	-	-	-	-	-
1	1	0	-	-	-	-	-	-
1	1	1	-	-	-	-	-	-

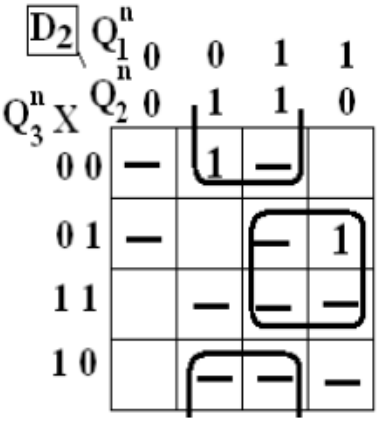
4.5.3.1.b ábra

A 3.mintafeladat vezérlési táblája

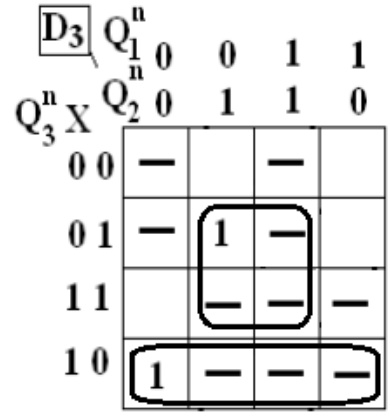
A vezérlési tábla adatai alapján megszerkesztett és kitöltött Karnaugh táblákon az egyszerűsítéseket elvégezve adódnak a D adatbemenetek logikai függvényei (4.5.3.1.c ábra):



$$D_1 = Q_1^n \overline{X} + Q_3^n X$$



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



$$D_3 = Q_2^n X + Q_3^n \overline{X}$$

4.5.3.1.c ábra[1]

*A 3.mintafeladat D-MS flip-flop **D** bemeneteinek*

Karnaugh táblái, és a táblázatokból kiírt algebrai függvényalakok

4.5.3.2 Egy másik megoldás: az ’1’-es súlyozású kódolás kihasználása

A feladat állapotgráfját tanulmányozva szembeötlő, hogy a három állapotot ezúttal három szekunder változóval különböztettük meg, holott elég lenne kettő is. Ugyanakkor azt is felfedezhetjük, hogy minden egyes kódszóban csak egyetlen ’1’-es szerepel. *Az olyan kódolást, melyben csak egyetlen bit helyén szerepel ’1’-es érték (a többi ’0’), 1-es súlyú kódolásnak (bit per state) nevezzük.*

Megjegyzés: szinkron hálózatok VLSI (Very Large Scale Integration= nagy integráltságú elemek) megvalósításakor gyakran igen gyorsan célravezető egy olyan állapotkód, amikor minden egyes szimbolikus állapothoz egy D–MS flip-flopot rendelünk. Ilyenkor ez a flip-flop ’aktív’, vagyis ’1’-es kimeneti értékű. Ahhoz azonban, hogy az állapotokat meg is különböztessük a hozzájuk rendelt kódszavakkal, éppen olyan hosszúságú (bitszámú) kódot kell alkalmazni, ahány állapotunk van. Ez biztosítja az 1-es súlyú kódolás helyes alkalmazását is.

A különböző állapotokhoz tetszőleges helyiérték szerint adhatjuk meg az egyetlen egy ’1’-esünk pozícióját, így többféle kódkiosztás is – a specifikáció által elvárt - helyes működési megoldáshoz vezet.

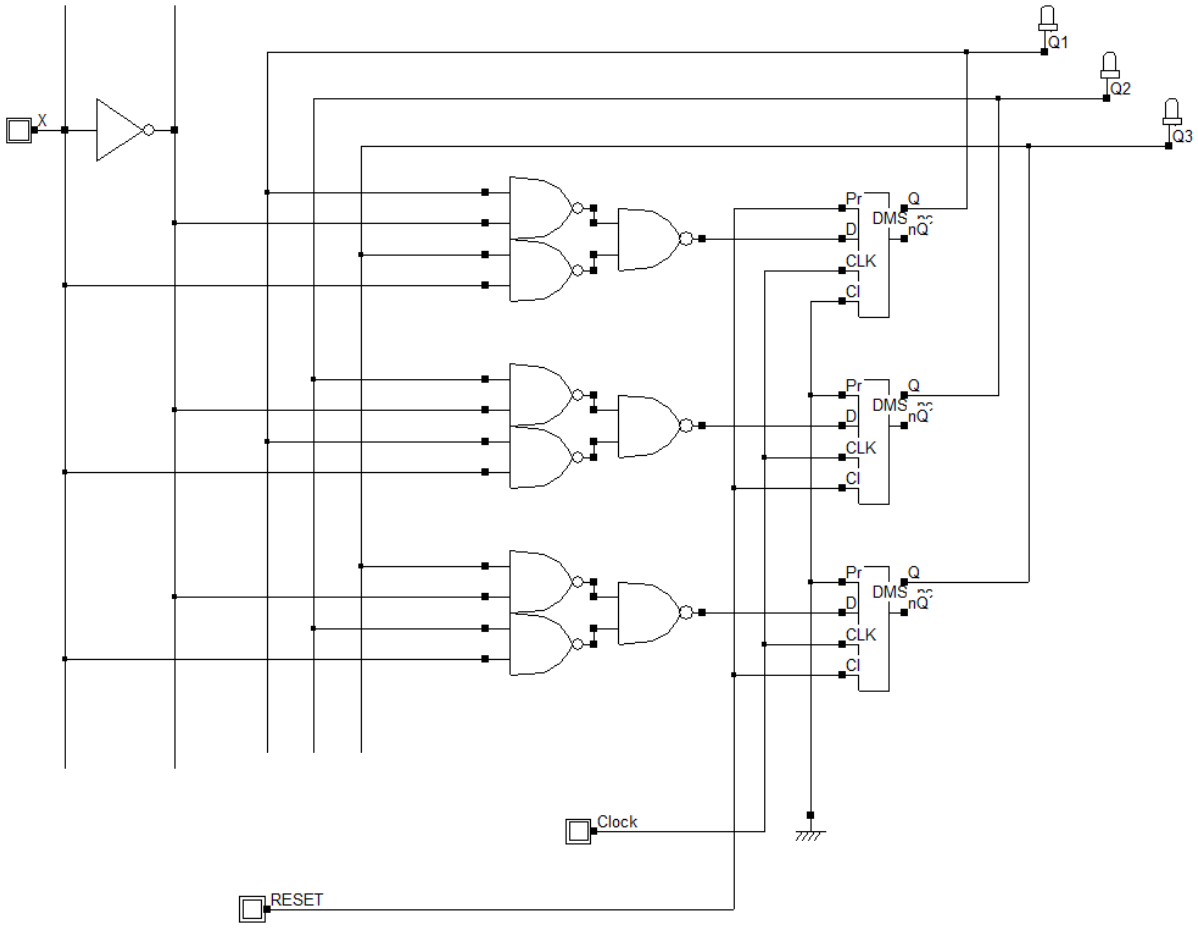
A D bemenetek vezérlésének megvalósítása ilyenkor rendkívül egyszerű, és az állapotgráf alapján elvégezhető: minden egyes D bemenetre akkor és csak akkor kell ’1’-et kapcsolni, amikor az általa reprezentált tároló kimenetnek ’0’-ról ’1’-re kell változnia, azaz amikor az adott flip-flop által reprezentált állapotnak be kell állnia. Ez pedig az állapotgráfból egyértelműen kiolvasható. Az így kapott algebrai függvényalak megegyezik a Karnaugh táblák alapján meghatározottakkal, azaz

$$D_1 = Q_1^n \overline{X} + Q_3^n X$$

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

$$D_3 = Q_2^n X + Q_3^n \overline{X}$$

Ne feledkezzünk meg a kezdeti állapot helyes beállításáról sem! Példánkban ez azt jelenti, hogy a kezdeti állapot ($D_1 D_2 D_3$ = ’100’) beállítását végző ’R’ jelünket ezúttal a D_1 tárolónk ’preset’ (Pr) segédbemenetére, a D_2 és D_3 flip-flop esetében pedig a ’clear’ (Cl) segédbemenetre kell kötni (4.5.3.2.a ábra):



4.5.3.2.a ábra

A 3.mintafeladat NAND-NAND architektúrájú áramköri realizációja a DSCH program segítségével

4.6 Szinkron sorrendi hálózatok tervezése mintapéldákon keresztül II.

4.6.1 Szinkron számláló alkalmazása a szinkron sorrendi hálózatokban

Mielőtt az első mintafeladatot ismertetnénk, tekintsük át a szinkron számlálók felépítését és működésük néhány fontos ismérvét!

A számlálók olyan összetett funkciójú digitális egységek, amelyek az eredeti, hagyományos számlálási funkción túl digitális egységek időzítő-vezérlő áramköreiben is felhasználásra kerülnek, mint rögzített funkciójú időzítők.

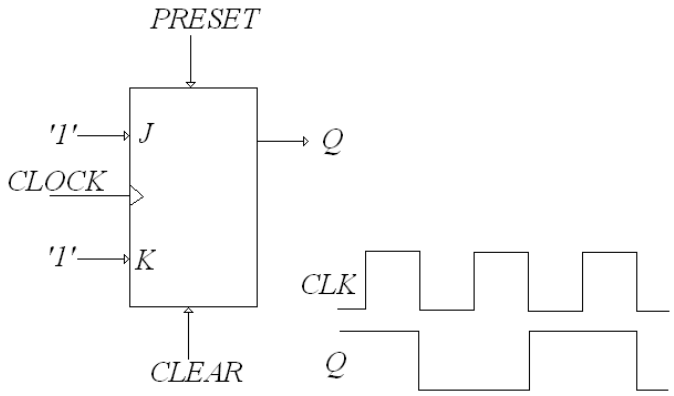
A számlálókat két csoportra osztjuk: szinkron és aszinkron számlálókra. Mindkét típusú számlálót a '*modulo*' értékkel jellemzünk. A *'modulo-m*' (m-modulusú) számláló számlálási tartománya a természetes számok '*m*'-mel való osztása után kapott lehetséges maradékok tartományán halad át. (*0, 1, 2, .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

Megjegyzés: a számlálókkal, mint összetett digitális alapegységekkel egy későbbi fejezetben még foglalkozunk, de most, a szinkron sorrendi hálózatok ismertetése kapcsán csak a szinkron számláló felépítését tárgyaljuk meg részletesebben.

4.6.1.1 A JK-MS flip-flop, mint a számlálók alapeleme

Az aszinkron 'preset' és 'clear' bemenetekkel ellátott JK-MS 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 4.6.1.1.a. 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 JK 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 közre-működése nélkül.



4.6.1.1.a ábra[1]

A JK MS flip-flop kettes-osztó funkciója

A szóban forgó flip-flop számlálókban történő alkalmazása esetén még kiegészül egy ún. engedélyező(E) bemeneti jellel, melynek szerepe, hogy csak akkor engedi a J és K bemeneten megjelenő logikai érték órajelciklus által történő mintavételezését, ha magas szinten van. Ellenkező esetben (alacsony szint) a tároló kimenetén őrzi az előző ciklus végén állandósult logikai értéket.

4.6.1.2 A szinkron számláló modellje

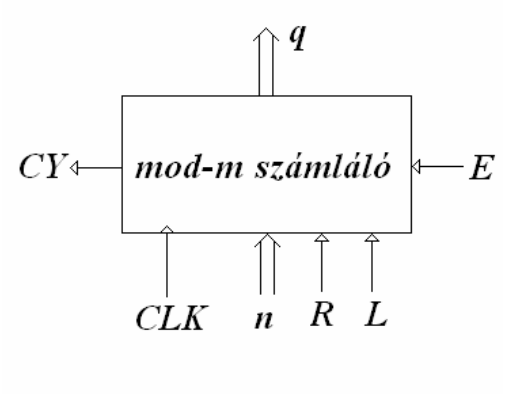
A szinkron számlálókat felépítő JK-MS vagy D-MS flip-flopok órajele azonos. A visszacsatoló hálózat a számláló állapot átmeneti gráfja alapján tervezhető meg azokkal a módszerekkel, amelyeket a szinkron hálózatok tervezésének tárgyalásakor elsajátítottunk. Hangsúlyozandó, hogy számlálók esetén az állapotkód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

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

- az *L* (LOAD) jel hatására - amennyiben nincs *R* - a számláló tartalma a következő órajel lefutó élére az *n* adatbemenet(ek)re kapcsolt bitvektor lesz.

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

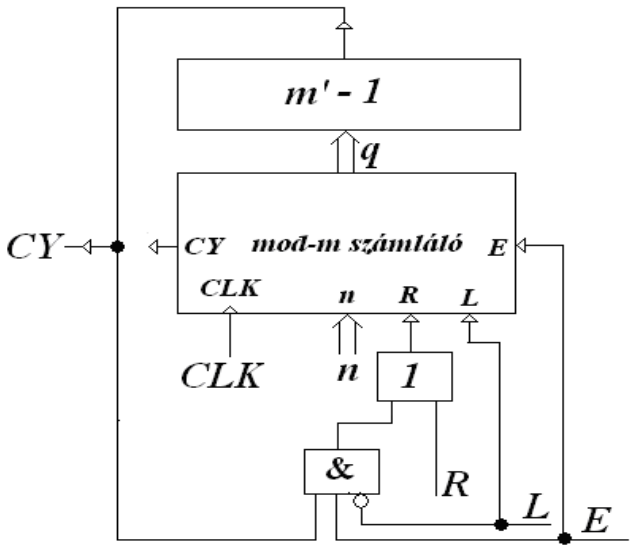
A 4.6.1.2.a. ábrán látható a szinkron számláló általánosan használt szimbóluma:



4.6.1.2.a. ábra[1]

Szinkron számláló általános szimbóluma

Ha egy *m*-modulusú szinkron számlálót át kívánunk alakítani *m' < m* modulusúvá, akkor új 'carry' (*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. A vezérlő jelek közötti prioritási viszony csak akkor örökíthető át az átalakított számlálóra, ha beiktatjuk az $\bar{L}$ és *R* bemenetű ÉS kaput. Az *L* jel *E* feletti prioritásának ugyanis akkor is érvényesülnie kell, ha a *CY* = 1! (4.6.1.2.b ábra)

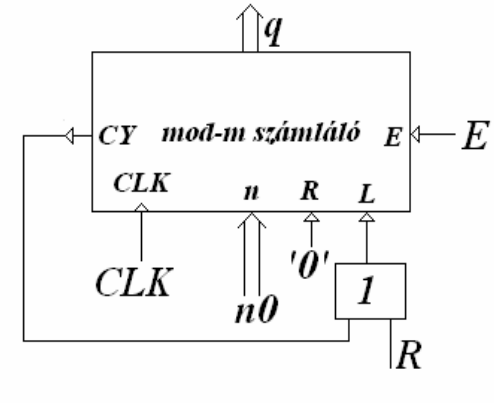


4.6.1.2.b ábra[1]

Egy m-modulusú szinkron számláló átalakítása m' < m modulusú szinkron számlálóvá

Megjegyzés: az átalakított m' < m modulusú számlálónk 'n' bitvektor bemenetén nem íródhat be m'-1-nél nagyobb érték: ezt vagy feltételként szabjuk a felhasználónak, vagy gondoskodni kell egy kiegészítő, ún. szűrőáramkörről (kombinációs hálózat), mely megakadályozza az m'-1-nél nagyobb értékek megjelenését az 'n' adatbemeneteken!

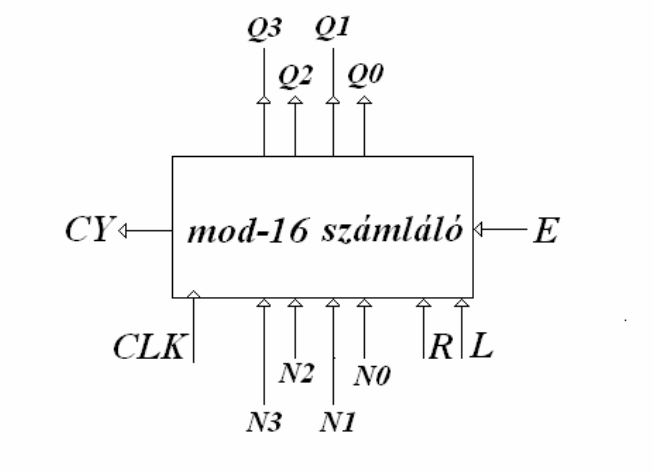
Amennyiben nullától különböző ('n0'), de a modulusnál kisebb kezdőértékkel szeretnénk ciklikusan indítani a számlálást, akkor az eredeti számláló *L* jelének felemelésével, a *CY* =1 feltétellel írjuk az adatbemeneteken keresztül a számlálóba ciklikusan az új kezdőértéket. Az új *R* bemenet ugyancsak az *L* bemenetre hat. Az eredeti *R* bemenetet nem használjuk, azt konstans logikai '0' értékre kötjük (4.6.1.2.c ábra).



4.6.1.2.c ábra[1]

Nullától különböző kezdeti érték beállítása szinkron számlálón

A különböző modulusú számlálók közül a leggyakrabban használt *mod-16*-os számláló szimbólumát a 4.6.1.2.d ábrán látjuk:

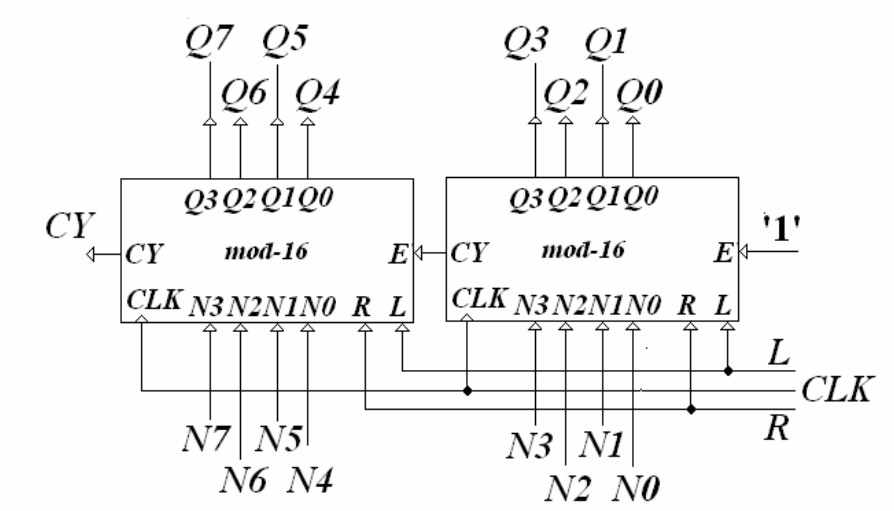


4.6.1.2.d ábra[1]

4-bites mod-16-os szinkron számláló

Megjegyzés: a 4.6.1.2.d ábrán látható alapkiépítés mellet még több funkciót is megvalósítanak egy egységben. Például a felfelé('up') történő számlálás mellett lefelé('down') való számlálást is (UP-DOWN COUNTERS).

Ha a bemutatott mod-16-os számlálót 15-ig ('1111') felszámoltattuk, akkor a *CY* (CARRY) kimeneten magas szint jelenik meg. A következő órajel lefutó élére a számláló ismét '0'-ra áll ('0000'), és a *CY* kimenet is alacsony szintre kerül. Ez a *CY* arra alkalmas, hogy egy magasabb helyiértékre helyezett számláló *E* bemenetét magas szintre állítsa, így lehetővé téve nagyobb modulusú számlálók kialakítását (kaszkádosítás: 4.6.1.2.e ábra).



4.6.1.2.e ábra[1]

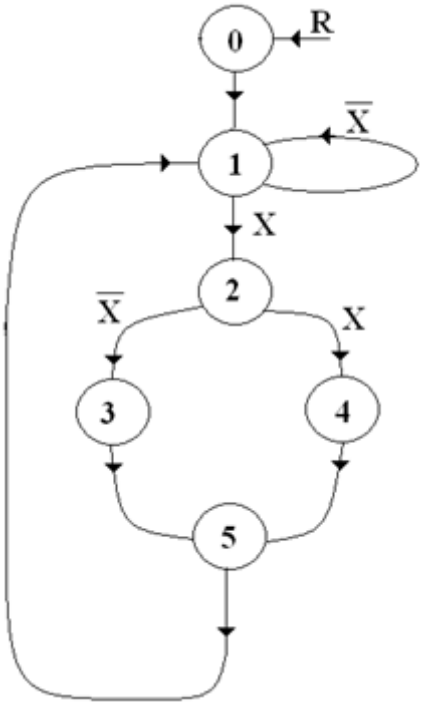
Mod-256-os számláló kialakítása mod-16 modulokból.

A törölhető, engedélyezhető, betölthető szinkron számlálókat szinkron sorrendi hálózatok állapot-regiszteréként alkalmazhatjuk. Ez azt jelenti, hogy a hálózat állapotait a számláló kimenetei kódolják. Ha egy állapotgráf lehetséges átmeneteit számba vesszük, belátható, 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* (enable) jelet emeljük fel. Ha az átmenet egy nem önmagára és nem az inkrementált kódú következő állapothoz vezet, akkor az *L*(load) 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, és ezeket programozott logikai hálózatként alkalmazzuk.
Nézzük most meg az első mintapéldán keresztül, hogy melyek a számlálóval kialakított szinkron sorrendi hálózat tervezésének fő lépései!

4.6.2 Szinkron sorrendi hálózat tervezése szinkron számlálóval: 1. mintafeladat

Példa:

valósítsuk meg törölhető (R), betölthető (L) és engedélyezhető (E) számlálóval és multiplexerekkel az alábbi kódolt állapotgráf szerinti szinkron sorrendi hálózatot!(4.6.2.a ábra)



4.6.2.a ábra[1]

Az 1.mintafeladat szinkron számlálóval kialakított sorrendi hálózatának állapotgráfja

A 4.6.2.a ábrán látható az állapot-kimenetű szinkron sorrendi hálózat kódolt állapotgráfja.

Első lépésként szerkesszünk meg az állapotgráf alapján két táblázatot: az egyik táblázat a *lépések('E')*, azaz az inkrementálások logikai feltételeit, a másik az *ugrások('L')* logikai feltételeit és következő állapotait tartalmazza (4.6.2.b ábra).

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'			

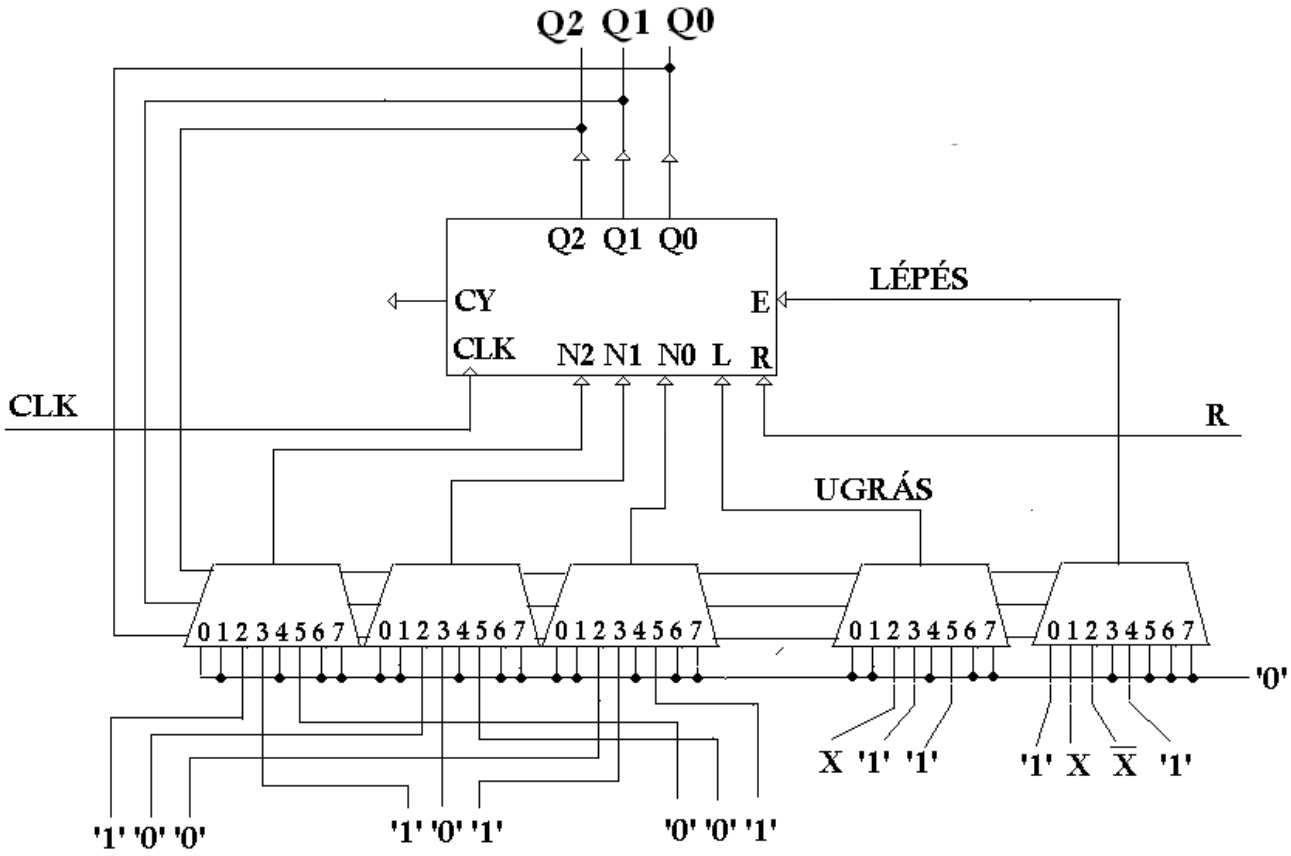
4.6.2.b ábra[1]

Az 1.mintafeladat 'lépés' és 'ugrás' táblázatai

A feladat megoldásának következő lépése a multiplexerek programozása:

- a léptető (*E*) vezérlőbemenetre csatlakozó multiplexert a „LÉPÉS” táblázatban meghatározott feltételek („feltétel” oszlop) szerint kell programozni,
- az ugrást vezérlő bemenetre kapcsolódó multiplexert az „UGRÁS” táblázatban felvett feltételek („feltétel” oszlop) szerint kell programozni,
- az ’n’ bitvektor bemenetekre csatlakozó multiplexerek adatbemeneteit pedig szintén az „UGRÁS” táblázatban lefektetett, következő állapot („köv.áll.” oszlop) kódjának megfelelő bináris értékekkel kell felprogramozni.

A 4.6.2.c ábrán látható, speciálisan erre a feladatra felrajzolt célarchitektúra mutatja, hogyan kell az aktuális állapot kódja által megcímzett multiplexer bemenetekre a táblázatok szerinti feltételeket, illetve a betöltendő következő állapotkódokat rákapcsolni.



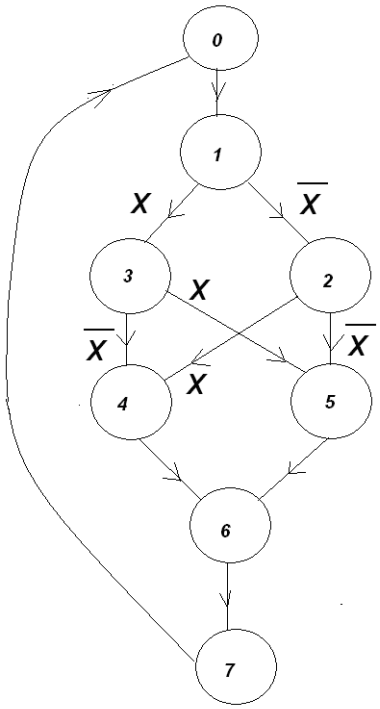
4.6.2.c ábra[1]

Az 1.mintafeladat realizációja specifikált célarchitektúrával, mod-8-as számlálóval és mpx8-1 multiplexerekkel

4.6.3 Szinkron sorrendi hálózatok tervezése szinkron számlálóval: 2.mintafeladat

Példa:

valósítsuk meg törölhető (R), betölthető (L) és engedélyezhető (E) számlálóval és multiplexerekkel az alábbi kódolt állapotgráf szerinti szinkron sorrendi hálózatot!(4.6.3.a ábra)



4.6.3.a ábra[1]

Az 2.mintafeladat állapotgráfja

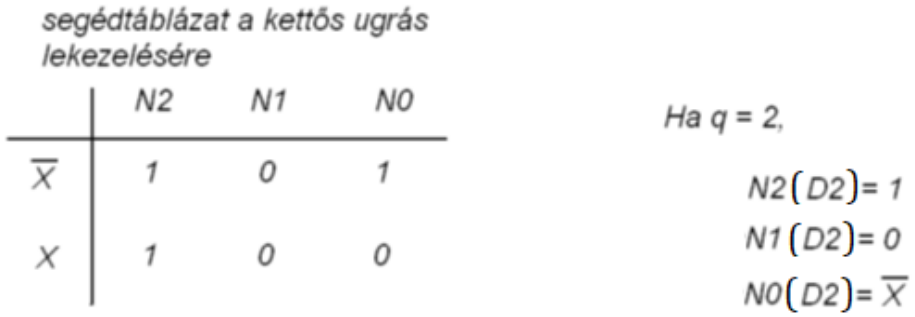
A feladat megoldását az 1. mintapéldában ismertetettek szerint kezdjük a „LÉPÉS” és „UGRÁS” táblázatok elkészítésével (4.6.3.b ábra):

lépések		*ugrások*		
akt. áll	*feltétel*	*akt. állapot*	*feltétel*	*köv. állapot*
0	*'1'*	*1*	*X*	*3*
1	$\overline{X}$	*2*	$\overline{X}$	*5*
3	$\overline{X}$	*2*	*X*	*4*
5	*'1'*	*3*	*X*	*5*
6	*'1'*	*4*	*'1'*	*6*
7	*'1'*			

4.6.3.b ábra[1]

A 2.mintafeladat állapotgráfja alapján felvett 'lépések' és 'ugrások' táblázatai

Az elkészített táblázatokat tanulmányozva egy érdekes jelenségre figyelhetünk fel: az „ugrások” táblázatában azt látjuk, hogy a számlálónk ’2’-es aktuális állapotában (*q=2*) $\overline{X}$ feltétel megléte esetén következő állapotként az 5-ös számlálóállásba kell „ugratnunk” a számlálónkat, míg *X* esetén a 4-es számlálóállásba. Ez azt jelenti, hogy az *X* feltétel, mint logikai változó akár ponált, akár negált értéket vesz fel, a ’2’-es számlálóállás (*q=2*) esetén a következő órajelciklus végén mindenképpen ugranunk kell: vagyis az *L* bemenetre csatlakozó multiplexer 2-es lábára konstans magas szintet kell kapcsolni. Azt, hogy ugyanakkor az *X* feltétel függvényében melyik új számértéket olvastassuk be az ’*n*’ bitvektor bemenetére csatlakozó multiplexereken keresztül - esetünkben a 5-öt (’*101*’) vagy a 4-et (’*100*’) – a számlálónkba, azt egy ún. segédtáblázat elkészítésével tudjuk meghatározni (4.6.3.c ábra).



segédtáblázat a kettős ugrás lekezelésére

	N2	N1	N0
$\overline{X}$	1	0	1
X	1	0	0

Ha q = 2,

N2(D2) = 1
N1(D2) = 0
N0(D2) = $\overline{X}$

4.6.3.c ábra[1]

A 2.mintafeladat segédtáblázata a „kettős ugrás” kezeléséhez

Mivel a beolvasandó szám értéke *X*-től, mint logikai változótól függ, ezért a táblázat soraihoz *X* lehetséges értékeit rendeljük, az oszlopokhoz pedig az egyes adatbemeneteket (MSB szerint N_2 , N_1 , N_0). Következő lépésként a táblázaton belül beírjuk az egyes *X* feltételek esetén a bemeneteken megjelenítendő számértékeket bináris kód szerint. Vagyis egy olyan táblázatot alkottunk, amiből egyértelműen ki lehet olvasni, hogy *X* függvényében milyen logikai értékeket kell az adott számlálóállás (példánkban *q*=2) esetén felprogramozni a multiplexerek megfelelő adatbemeneteire.

A táblázat N_2 bemenethez tartozó oszlopában azt látjuk, hogy *X* bármely logikai értéke esetén ezen a bemeneten keresztül magas szintet kell beolvasnunk a számlálónkba, így az N_2 bitvektor bemenethez csatlakozó multiplexerünk 2-es adatbemeneti lábára (D_2) konstans magas szintet kell kötnünk.

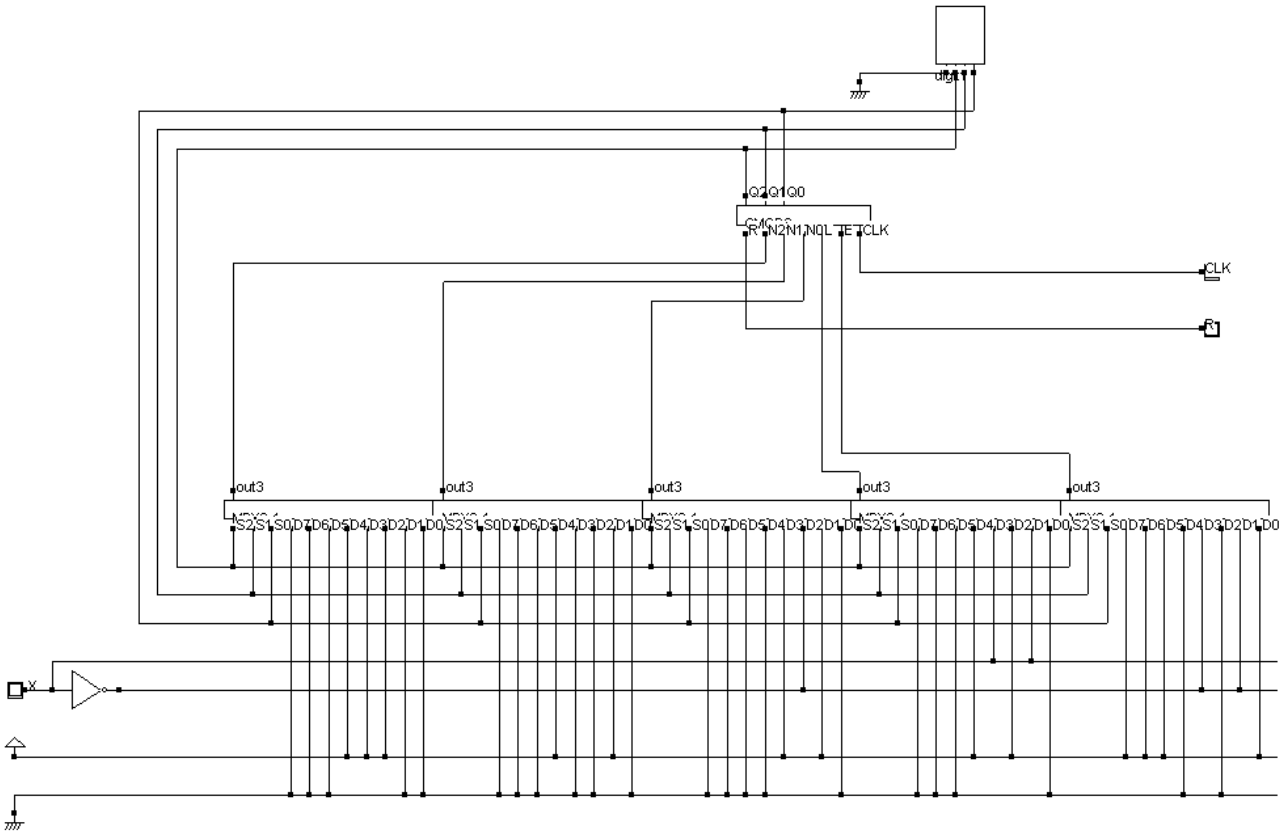
A táblázat N_1 bemenethez tartozó oszlopában pedig azt látjuk, hogy *X* bármely logikai értéke esetén ezen a bemeneten keresztül alacsony szintet kell beolvasnunk a

számlálónkba, így az N_1 bitvektor bemenethez csatlakozó multiplexerünk 2-es adatbemeneti lábára (D_2) konstans alacsony szintet kell kötnünk.

A táblázat N_0 bemenethez tartozó oszlopában található bejegyzések a következő logikát diktálják: ha *X* alacsony szintű, akkor az N_0 bitvektor bemenethez csatlakozó multiplexerünk 2-es adatbemeneti lábára (D_2) magas szintet kell kapcsolnunk, abban az esetben pedig, amikor *X* magas szinten van, ugyanerre a bemenetre alacsony szintet kell biztosítani. A feladatot csak úgy tudjuk megoldani, ha ezt a bemenetet *X* feltételnek a táblázatból adódó függvényeként vezéreljük, vagyis az $\bar{X}$ -at kötjük rá.

Általánosságban kimondhatjuk azt a szabályt, hogy kettős ugrás esetén mindig az adott feltételt szolgáltató változó függvényeként kell meghatározni az érintett adatbemeneti értéket.

A konkrét realizációhoz *mod-8*-as szinkron számlálóra és *mpx8-1* multiplexerekre, valamint egy inverterre lesz szükségünk. A DSCH szimulátor programban megszerkesztett architektúra a 4.6.5.d ábrán látható:



4.6.3.d ábra[1]

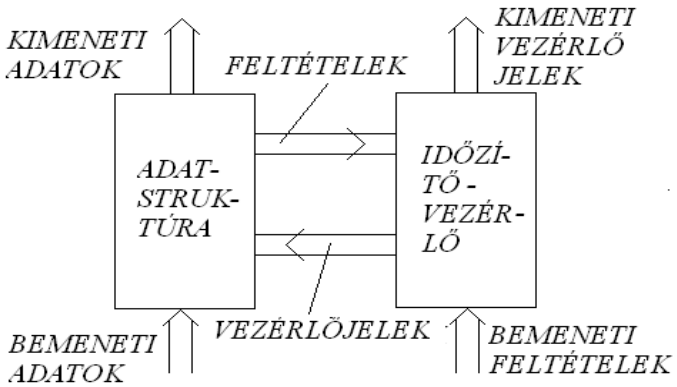
A 2.mintafeladat DSCH szimulátorban előállított realizációja

4.6.4 Szinkron sorrendi hálózatok tervezése szinkron számlálóval: 3.mintafeladat

4.6.4.1 Szinkron számlálók alkalmazása időzítő-vezérlő egységekben

A harmadik mintafeladat ismertetése előtt tekintsük át röviden, mik azok az időzítő-vezérlők, és milyen helyet foglalnak el a digitális architektúrákban.

Digitális berendezések tervezésekor célszerű dekompozícióval meghatározni az egyes hálózati működési funkciókhoz tartozó részstruktúrákat. Ezen elv alapján célszerű elkülöníteni az adatutakat az azok kijelölését és időbeli mozgatását meghatározó időzítő-vezérlő egységtől (4.6.4.1.a ábra).



4.6.4.1.a ábra[1]

Digitális egység dekompozíciója

Az adatstruktúrára és időzítő-vezérlő egységre való felbontás szükségessé teszi a két egység közötti kapcsolódási pontok pontos meghatározását. Az adatstruktúra lényegében regiszterekből, multiplexerekből és funkciós egységekből áll, bemeneti adatokat fogad, és kimeneti adatokat szolgáltat. Ezek az egységek egymással összekapcsolódva különféle adatpályákat 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 műveletvezérlő kódjait, és felemeli, illetve lebocsátja a regiszterek beíró jeleit. Ezeket nevezzük vezérlőjeleknek. 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. A következőkben az időzítő-vezérlő egységet röviden vezérlő egységnek fogjuk nevezni.

A számláló típusú vezérlő alapegysége egy törölhető(R), engedélyezhető(E), betölthető(L) szinkron számláló, amely a kapcsolódó multiplexerekkel együtt egy állapot-kimenetű szinkron sorrendi hálózatot valósít meg. A 4.6.4.1.b ábra szerinti sematikus felépítést tanulmányozva - a számláló vezérlőbemenetei között fennálló elsőbbségi viszonyokat figyelembe véve a működés a következőképpen írható le:

9. ha az R bemenetet felemeljük, a számláló tartalma a fennálló helyzettől függetlenül nullázódik. Ezzel szemben, ha

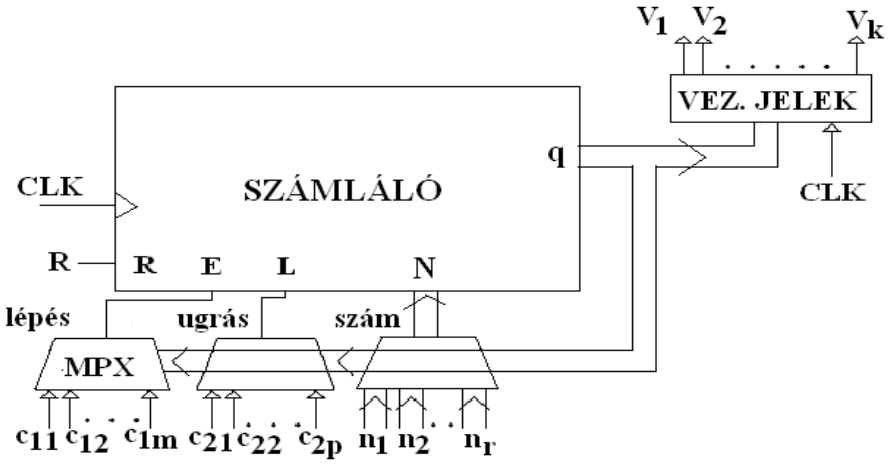
- az R bemenet szintje alacsony, akkor két eset van:

    1. 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,

    2. ha a betöltésre kapcsolódó feltétel szintje alacsony. Ekkor ismét két eset van:

        (a) egyik, ha a kiválasztott, az engedélyező bemenetre kapcsolódó feltétel magas. Ilyenkor a számláló tartalma inkrementálódik, ezzel szemben

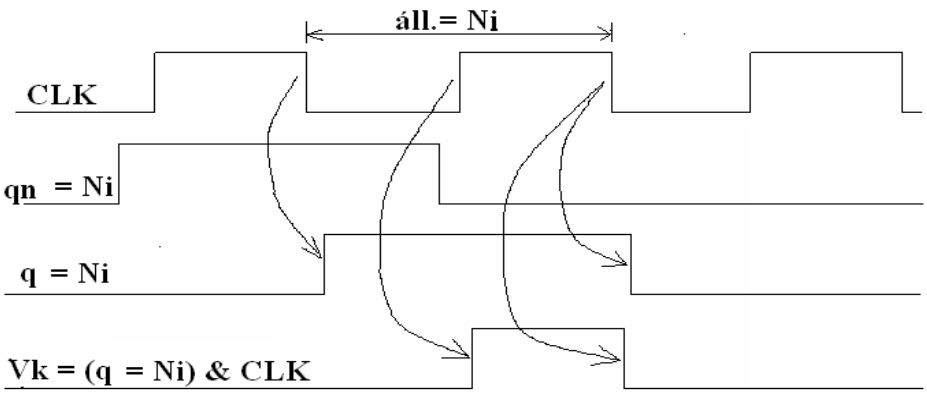
        (b) ha az engedélyező bemenetre kapcsolódó feltétel szintje alacsony, akkor a számláló értéke nem változik.



4.6.4.1.b ábra[1]

Vezérlőegység kialakítása szinkron számláló felhasználásával

A fenti működési fázisokat az előző fejezetben már részleteztük, de a vezérlő másik egységét ott nem tárgyaltuk: ez az egység a környezetnek és az adat-struktúrának szóló vezérlőjeleket generálja. A vezérlőjelek generálásának alapproblémája a hazárdmentesség. 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 vezérlőjel aktivitását magával az órajellel szinkronizáljuk, akkor a 4.6.4.1.c. ábra szerinti idődiagram igazolja a hazárdmentességet:



4.6.4.1.c. ábra[1]

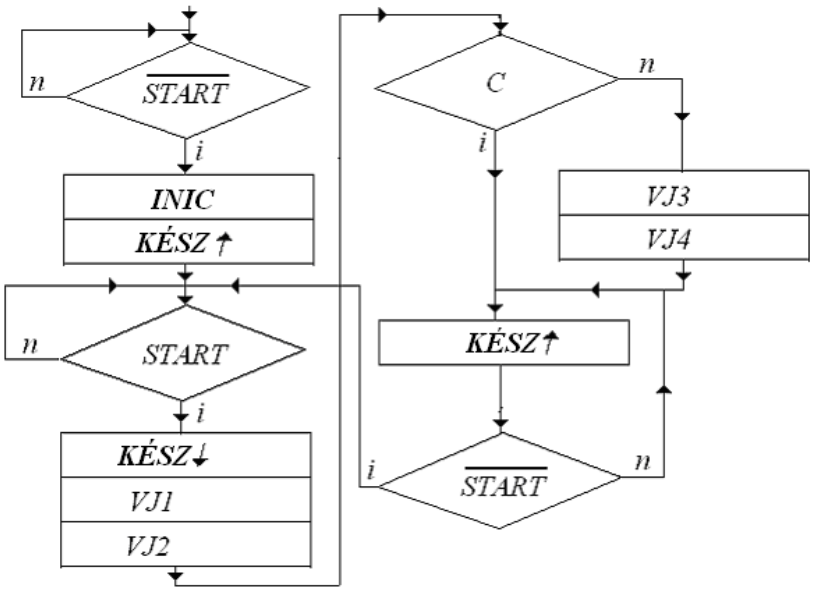
Hazárdmentes vezérlés megvalósítása vezérlőegységben

Legyen q_n 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 vezérlőjel aktivitását kiváltani, az órajellel szinkronban. Belátható, hogy a vezérlőjel bekövetkezése után következő órajel már az időzítési feltétel biztos elmúlása után jelenik meg! Ezzel biztosított a hazárdmentes vezérlés.

4.6.4.2 A 3.mintafeladat

Példa:

tervezzünk időzítő-vezérlő egységet szinkron számláló felhasználásával az alábbi (4.6.4.2.a ábra) folyamatábra alapján!



4.6.4.2.a ábra[1]

A 3.mintafeladat folyamatábrája

A folyamatábrán látható működés szerint az adatstruktúra számára szolgáltatni kell egy *INIC* nevű jelet, valamint a *VJ1, VJ2, VJ3, VJ4* vezérlőjeleket, a környezetből fogadni kell egy *START* jelet, és értelmezzük a *C* nevű jelet az adatstruktúrából eredő feltételnek. Ezeken felül a környezet számára még szolgáltatni kell egy *KÉSZ* jelet is.

A folyamatábra pontos értelmezéséhez a szimbólumok jelentése a következő:

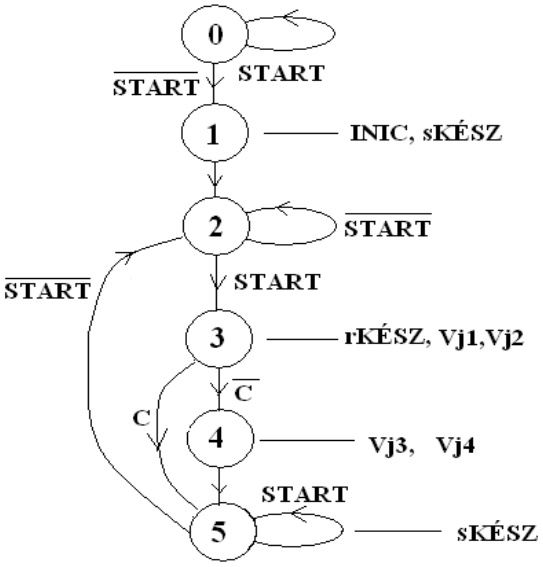
- rombusz, logikai feltétellel: ha igaz, az 'i' kimeneten távozunk
- négyszög, jelnév-bejegyzéssel: órajellel szinkronizált pozitív impulzus a megnevezett jelen
- négyszög jelnév-bejegyzéssel és felfelé vagy lefelé mutató nyíllal: az órajel felfutó élére szinkronizált felfutás vagy lefutás a megnevezett jelen

Megoldás:

Először is értelmezzük és határozzuk meg a hálózat működését a megadott folyamatábra alapján:

az ábra szerint a vezérlőegység a kezdeti állapotból elindulva - amennyiben a *START* alacsony - produkál egy *INIC* impulzust, és a *KÉSZ* felemelésével jelzi a készenlétet. Ezután vár a S*TART* 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 viszont magas, akkor azok kimaradnak. Ezután megint fel kell emelni a *KÉSZ* jelet, és a *START* szintje szerint visszatérni.

Következő lépésként célszerű a vezérlés folyamatának ütemezését, azaz egy vezérlési szekvencia leírását egy számláló-állásokkal jelölt állapotgráfban elkészíteni (4.6.4.2.b ábra):



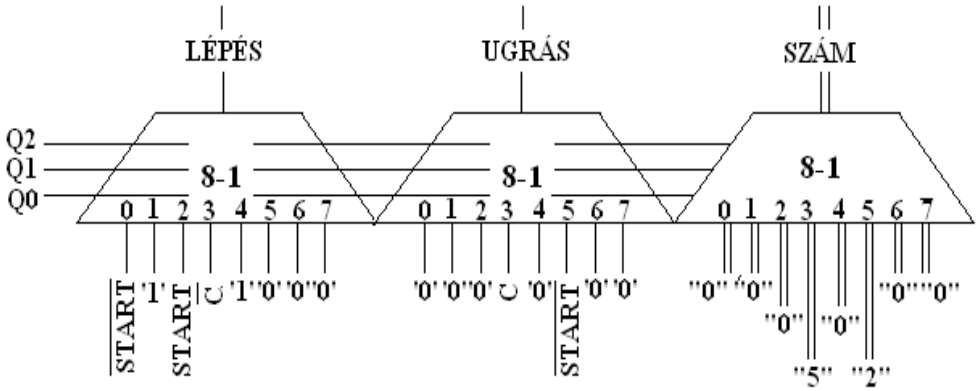
4.6.4.2.a ábra[1]

A 3.mintafeladat állapotgráffal megadott vezérlési szekvenciája

A számlálóállásokhoz rendelt akciókat a gráf jobboldalán soroljuk fel. Az akciórész lehet üres. Ilyen például az első ütem, amikor a *START* értékétől függően várakozunk, vagy egy ütemmel továbblépünk. Figyeljük meg, hogy itt a jel emeléseket és ejtéseket '*s*' vagy '*r*' előtagokkal jelöljük, érzékeltetve ezzel, hogy az ilyen típusú jeleket *SR* bistabil tárolókkal 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.

A teljes szekvencia leírásához szükséges állapotok számából adódik, hogy a realizáláshoz mod-8-as számlálót kell használni, ami azt is jelenti, hogy 3 darab mpx8-1 multiplexerre lesz szükségünk.

A multiplexerek programozását is a vezérlési szekvencia leírásából lehet meghatározni. Az ehhez szükséges táblázatok elkészítése már rutinmunka: a „*LÉPÉS*" multiplexer megtervezésekor azokat az ütemeket gyűjtjük össze az ide tartozó táblázatban, amelyekben inkrementálás található. Az ütemek az inkrementáláshoz tartozó feltételeket címzik. Feltétel nélküli inkrementálási ütemhez nyilvánvalóan konstans logikai '1' tartozik. A másik szükséges táblázat felrajzolásánál hasonlóan gyűjtjük össze az „*UGRÁS*" multiplexer címeit és bemeneteit is. Ugyanezekhez a címekhez rendeljük a „*SZÁM*" multiplexer bemeneteit is, amelyekhez az ugrásokhoz beírt ütem-számokat rendeljük. Az így meghatározott multiplexer-hálózat programozása a 4.6.4.2.b ábrán látható, ahol a címzőbemenetekként megjelenő Q_1, Q_2 *és* Q_3 a mod-8-as számlálónk számláló(adat) kimeneti bitjei:



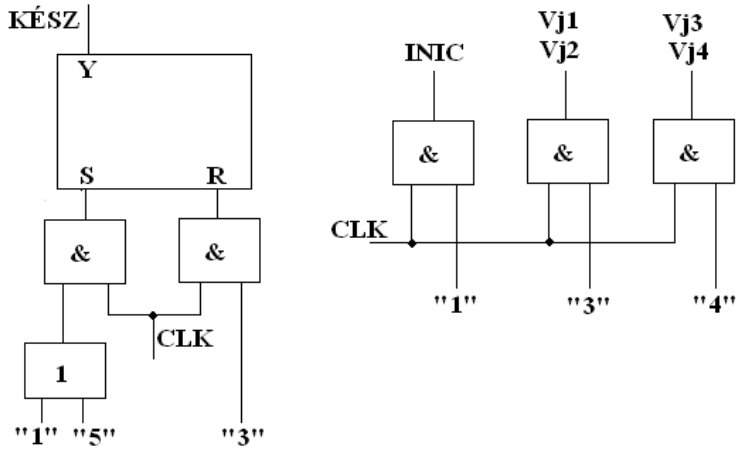
4.6.4.2.b ábra[1]

A 3.mintafeladat multiplexereinek programozása

A komplett realizáció elkészítéséhez még szükség van a vezérlőjel-generátorok megtervezésére.

Az *INIC* valamint a *VJ1, VJ2, VJ3, VJ4* vezérlőjelek előállítása NEM-VAGY kapukkal történhet, a számlálónk kimenetéről leolvasott 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 SR tárolón keresztül történik. Látható, hogy az alkalmazott tárolón is az órajellel szinkronban történik meg a változás! Az egység kapuszintű sematikus vázlata a 4.6.4.2.c

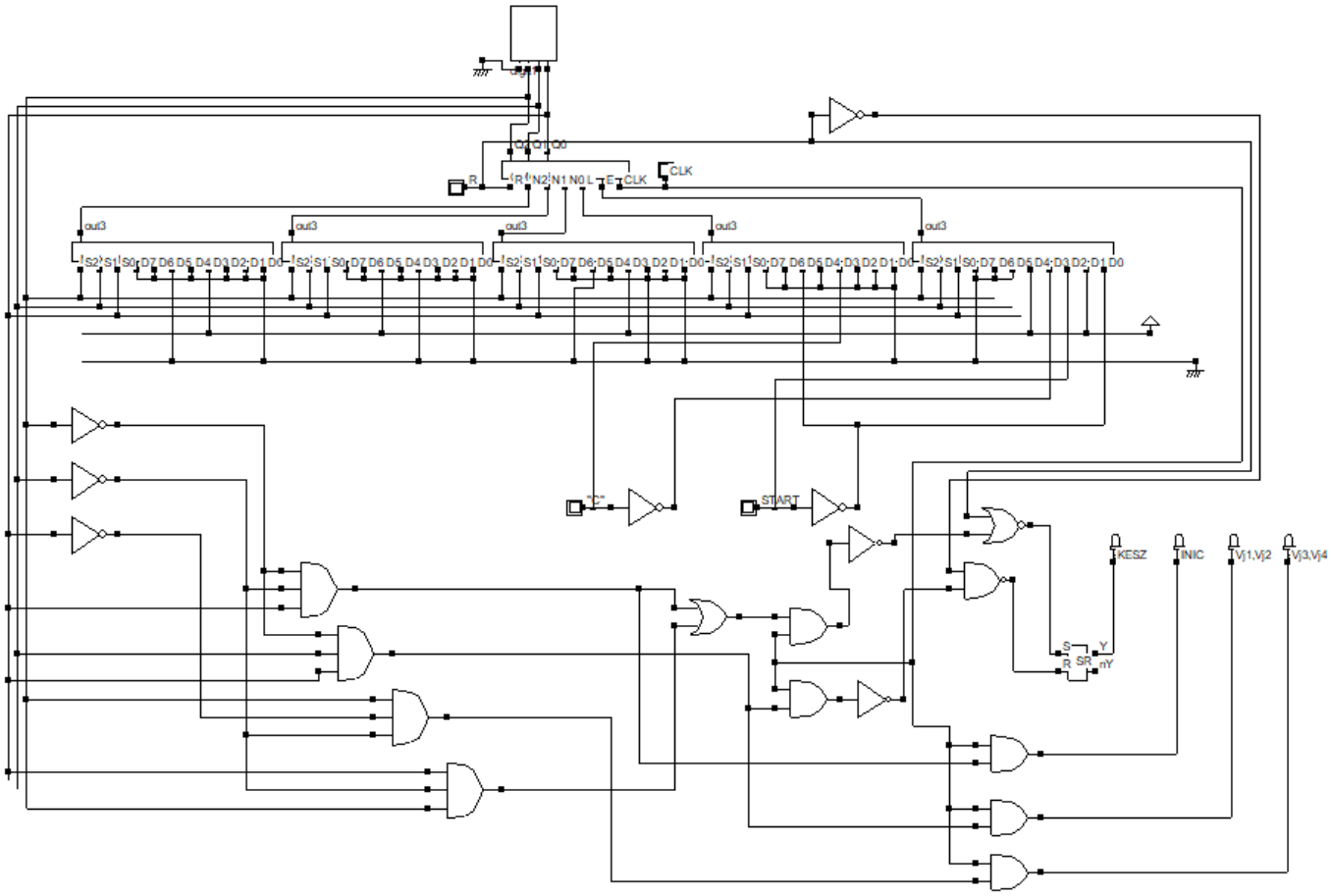
ábrán látható (az egyes ’számlálóállások’ dekódolt formában, egyetlen bittel kerültek feltüntetésre az ábrán):



4.6.4.2.c ábra[1]

A 3.mintafeladat vezérlőjel-generátorainak kapuszintű sematikus vázlata

A feladat komplex kapuszintű realizációja (DSCH) a 4.6.4.2.d ábrán látható:



4.6.4.2.d ábra

A 3.mintafeladat realizációja (DSCH)

4.7 Aszinkron sorrendi hálózatok tervezése mintapéldákon keresztül: kritikus versenyhelyzet és lényeges hazárdok aszinkron sorrendi hálózatokban.

Ebben a fejezetben mintapéldákon keresztül mutatjuk be az aszinkron hálózatok tervezési eljárásait, megismerjük az egyes tervezési lépések sorrendjét, valamint a megtervezett hálózatok logikai kapuszintű megvalósítását.

4.7.1 Aszinkron sorrendi hálózat tervezése: 1.mintafeladat - a kritikus versenyhelyzet felismerése és kiküszöbölése aszinkron sorrendi hálózatokban

Példa:

közvetlenül visszacsatolt kombinációs hálózattal tervezzünk olyan egybemenetű (X) és egykimenetű (Z) aszinkron sorrendi hálózatot, amelynek kimenetén a szint mindannyiszor ellenkezőjére vált, ahányszor a bemenet magas szintről alacsony szintre vált. Bekapcsolás után a hálózat az X=0 bemenetnél Z = 0 kimenetet szolgáltasson!

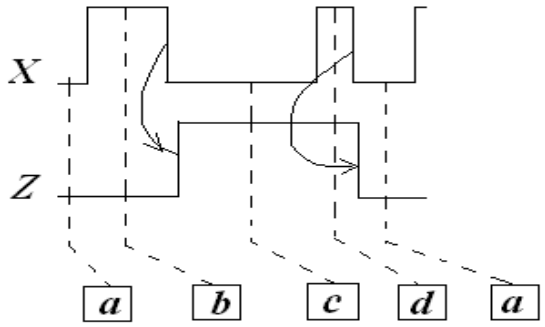
Megoldás

Aszinkron sorrendi hálózatokban nincs szinkronizáló jel, azaz a következő állapot (*n+1*) értelmezését nem egy órajel ciklus lezajlása után determináljuk. Ezekben a hálózatokban általánosságban azt feltételezhetjük, hogy a bemenet minden egyes változása egy új állapotot generál. Természetesen ez nem feltétlenül van így minden esetben, hiszen egy következő (*n+1*) állapot gyakran megegyezik az aktuális állapottal (*n*), amire már az előző feladatok során láttunk példákat. 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 megoldásának menete – a szinkron hálózatoknál már megismert szisztematikus eljárási módszer alapján - a következő:

1. *Idődiagram (állapot-átmeneti gráf) felrajzolása.*
2. *Előzetes szimbolikus állapottábla felvétele.*
3. *Összevont szimbolikus állapottábla megszerkesztése.*
4. *Kódolt állapottábla elkészítése a kritikus versenyhelyzet elemzésével*
5. *- közvetlen visszacsatolt kombinációs hálózat esetében a szekunder változó(k) és a kimenet(ek) függvényeinek Karnaugh tábla segítségével történő statikus hazárdmentesített megadása algebrai alakban;*
   - SR tárolók alkalmazása esetén a vezérlési tábla, valamint a szükséges Karnaugh táblák alapján az SR tároló(k) és a kimenet(ek) statikusan hazárdmentesített algebrai függvényalakjainak megadása.
6. *Kezdeti állapotról történő gondoskodás.*
7. *Lényeges hazárdok vizsgálata (szükség esetén késleltetések beiktatása).*
8. *Realizáció logikai kapukkal (és tárolókkal).*

Ahogyan a szinkron hálózattervezés kezdeti lépéseként az állapotgráf felvétele ajánlott, úgy ajánlható aszinkron hálózat tervezésének első lépéséül egy ún. *idődiagram* felvétele. Az idődiagram (vagy más néven ütemdiagram) felvételekor figyelembe kell venni, hogy az aszinkron hálózat minden új stabil állapotba való elindulása egy bemeneti jel változására indul meg, és működtetési szabály, hogy egyidejűleg csak egyetlen bemeneti jel változhat. Az idődiagram jól mutatja a bemeneti jelváltozások és a kimeneti kombináció-változások közötti ok-okozati összefüggéseket, és segítséget nyújt az előzetes, szimbolikus állapottábla felvételéhez(4.7.1.a ábra):



4.7.1.a ábra[1]

Az 1.mintafeladat idődiagramja

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

Következő lépésként meg kell adni a feladathoz tartozó előzetes szimbolikus állapottáblát. Az aszinkron hálózat szimbolikus előzetes állapottáblájának felvétele ugyanúgy intuitív módon oldandó meg, mint a szinkron hálózatok esetében, de általában nehezebb feladat annál. Fontos tervezői döntés, hogy Mealy-, vagy Moore-típusú hálózatot akarunk-e tervezni, hiszen az előzetes állapottábla felépítése ettől jelentősen függ. 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. A táblázatban 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.

Példánkban a felrajzolt idődiagram alapján szerkesztett előzetes szimbolikus állapottábla a 4.7.1.b ábrán látható:

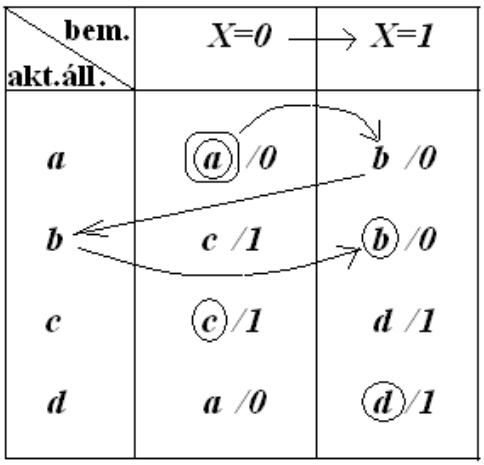
bem. / akt.áll.	*X=0*	*X=1*
a	(*a*)/*0*	*b* /*0*
b	*c* /*1*	(*b*)/*0*
c	(*c*)/*1*	*d* /*1*
d	*a* /*0*	(*d*)/*1*

4.7.1.b ábra[1]

Az 1.mintafeladat előzetes szimbolikus állapottáblája

Az állapottáblán körbe-foglalással jelöljük a stabil állapotokat. Tudjuk, hogy aszinkron állapottáblán stabil következő állapot az, amelynek szimbóluma azonos az aktuális állapot szimbólumával. A működést az állapottáblán is követhetjük. A kezdeti *a* aktuális állapotban az *X=0* bemenet az *a* állapotot stabilizálja. Az állapot mellett „/" jellel elválasztva látjuk a kezdeti kimeneti értéket.

Induljunk most el ebből a stabil *a* állapotból az *X=1* változás hatására a következő *b* stabil állapotba! Ez az ún. *stabil-stabil állapot átmenet* a következőképpen értelmezhető (4.7.1.c ábra): a bekarikázott *a* állapot *X=0* oszlopából átlépünk ugyanezen sor *X=1* oszlopába, ahol a *b,* mint *tranziens* (nem stabil, azaz átmeneti) következő állapotkódot találjuk, változatlan *Z* értékkel. Ha ez visszajut a bemenetre, akkor átlépünk a *b* aktuális állapot sorára, ahol ebben az oszlopban *(X=1*) azt látjuk, hogy a *b* állapot stabilizálódik. A táblázatban szereplő többi stabil-stabil állapot átmenet is (*b→c*, *c→d*, *d→a*) hasonlóképpen követhető.



4.7.1.c ábra[1]

Stabil-stabil állapot átmenet (a→b) szemléltetése az 1.mintafeladat előzetes szimbolikus állapottáblájában

Miután megszerkesztettük az előzetes szimbolikus állapottáblát, következő lépésként vizsgáljuk meg, lehetséges-e az állapotok számának csökkentése. Az összevonhatóság feltételei alapján sajnos ebben az esetben nem találunk nem megkülönböztethető állapotokat, így a kódolt állapottábla felvételénél ebből a táblázatból kell kiindulnunk.

Mivel négy állapotot kell bináris kóddal megkülönböztetni, ezért ehhez két szekunder változót (Y_1; Y_2) vezetünk be. Az egyes állapotokhoz rendeljük hozzá az alábbi (szabadon választott) kódolást:

	Y_1	Y_2
a :	0	0
b :	0	1
c :	1	0
d:	1	1

A kódolt állapotokat a 4.7.1.d ábrán látható kódolt állapottábla tartalmazza:

kódolt aktuális állapot $Y_1^v Y_2^v$	kódolt következő állapot/kimenet	
	X=0 $Y_1^v Y_2^v$/Z	X=1 $Y_1^v Y_2^v$/Z
0 0	(0 0)/0	0 1/0
0 1	1 0/1	(0 1)/0
1 0	(1 0)/1	1 1/1
1 1	0 0/0	(1 1)/1

4.7.1.d ábra

Az 1.mintafeladat kódolt állapottáblája egy szabadon választott kódolás alapján

<u>A feladat megoldásában ennél a pontnál egy nagyon fontos vizsgálatot kell elvégeznünk!</u>

Ennek oka pedig a következő: eddigi tanulmányaink során a szinkron hálózatok állapotkódolásánál szabad kezünk volt abban, hogy a megfelelő hosszúságú szavakból álló kódkészlet szavait hogyan rendeljük hozzá a szimbolikus állapotokhoz, ugyanis minden választás a specifikációnak megfelelő megoldáshoz vezetett. 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 jelkésleltetési 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. (Egy későbbi fejezetben egy szisztematikus állapotkódolási módszert is megismerünk majd.)

Vizsgáljuk meg a 4.7.1.e ábrán látható táblázat segítségével a *'01'→'10'* stabil-stabil állapot átmenetet!

kódolt aktuális állapot $Y_1^v Y_2^v$	**kódolt következő állapot/kimenet** X=0 $Y_1^v Y_2^v$/Z	X=1 $Y_1^v Y_2^v$/Z
0 0	*0 0/0*	*0 1/0*
0 1	*1 0/1*	*0 1/0*
1 0	*1 0/1*	*1 1/1*
1 1	*0 0/0*	*1 1/1*

4.7.1.e ábra

Egy ideális, '01'→'10' stabil-stabil állapot átmenetet

Tegyük fel, hogy az *X=1*-hez tartozó *'01'* kódú *b* állapotban vagyunk. Ha most *X* bemenetet *'0'*-ra kapcsoljuk, akkor tranziens állapotként az *'10'* (*c*) állapot beállását várjuk, amely a bemenetre visszajutva stabilizálódik, és ezzel megtörténik az elvárt *'01'→ '10'* átmenet. *Azonban ha a hálózatot ezzel az állapotkóddal megvalósítjuk, hibás lehet a valóságos működés!* A 4.7.1.f ábrán szemléltetjük, mi lesz annak a következménye, ha a késleltetési idők különbözősége miatt egy másik tranziens állapot, a *'00'* (*a*) áll be.

Ebben az esetben az átmenet kétféleképpen is megvalósulhat:

1. először Y_2^v vált *'1'*-ről *'0'*-ra ($Y_1^v Y_2^v$ *='01'→'00')* az *X=1→0* hatására. Ekkor a *'00'* kódú sorban az *X=0* oszlopában következő állapotként a *'00'* kódot látjuk, ami azt jelenti, hogy ebben a kódú állapotban stabilizálódik a hálózatunk. Ez helytelen, nem előírásszerű, vagyis HIBÁS működést eredményez, hiszen a kiindulásbeli, *'01'→'10'* vezérlés (4.7.1.e ábra) az *'10'* állapotkódba jutást irányozza elő!

2. először Y_1^v vált ’0’ról ’1’-re ($Y_1^v Y_2^v$ =’01’→’11’) az *X=1→0* hatására. Ez azt jelenti, hogy az ’11’ kódú állapot sorába ugrunk, az *X=0* oszlopban. Itt megint csak egy tranziens kódot látunk: ’*00*’. Vagyis következő lépésként a ’*00*’ állapotkód sorába ugrunk, maradva természetesen ebben az oszlopban. Itt viszont most a következő állapot „önmaga”, vagyis stabil ’*00*’. Tehát ez esetben is helytelen a működés, és nem teljesül az eredetileg ’*10*’ állapotba jutás és stabilizálódás.

kódolt aktuális állapot $Y_1^v Y_2^v$	kódolt következő állapot/kimenet	
	X=0 $Y_1^v Y_2^v$/Z	X=1 $Y_1^v Y_2^v$/Z
0 0	0 0/0	0 1/0
0 1	1 0/1	1. 0 1/0
1 0	1 0/1	2. 1 1/1
1 1	0 0/0	1 1/1

A 4.7.1.f ábra
Kritikus versenyhelyzet az $Y_1^v Y_2^v$ =’01’→’10’ állapot átmenet esetében

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

Ezen elvet követve kódoljuk újra a megkülönböztetendő állapotokat a következők szerint:

	Y_1	Y_2
a:	0	0
b:	0	1
c:	1	1
d:	1	0

Készítsük el ez alapján újból a kódolt állapottáblát, és vizsgáljuk meg, valóban kritikus versenyhelyzet mentes-e a kódolásunk (4.7.1.g ábra)!

kódolt aktuális állapot $Y_1^v Y_2^v$	kódolt következő állapot/kimenet	
	X=0 $Y_1^v Y_2^v$/Z	X=1 $Y_1^v Y_2^v$/Z
0 0	0 0/0	0 1/0
0 1	1 1/1	0 1/0
1 1	1 1/1	1 0/1
1 0	0 0/0	1 0/1

4.7.1.g ábra

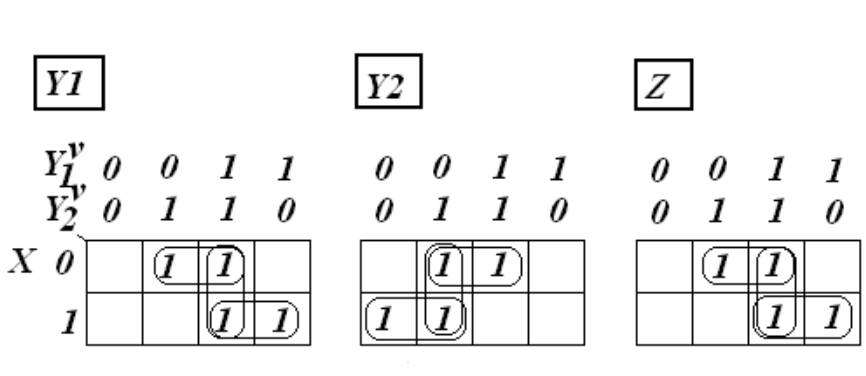
Az 1.mintafeladat kritikus versenyhelyzetektől mentes állapotkódolása

Láthatjuk, hogy egyetlen stabil-stabil állapot átmenet esetében sem merül fel kritikus versenyhelyzet, tehát a hálózatunk a specifikáció szerint fog működni.

A megfelelően kódolt állapottábla megszerkesztése után adjuk meg Karnaugh táblás egyszerűsítés segítségével a szükséges algebrai függvényalakokat! Ehhez készítsük el a szekunder változók és a kimenet kódolt állapottáblájából származtatott Karnaugh táblák kitöltését, és írjuk fel a minimalizáláshoz a lehetséges prímimplikánsokat.

Figyelem! Valamennyi szekunder változóhoz tartozó függvényt statikus hazárdoktól mentesen kell lefedni: az f_y hálózat kimenetein jelentkező statikus hazárd ugyancsak a specifikációtól eltérő, hibás működéshez vezethet, és általában követelmény a kimenet hazárdmentessége is!

Mindezek figyelembe vételével a 4.7.1.h ábra szerinti grafikus függvényalakokat kapjuk:



4.7.1.h ábra[1]

Az 1.mintafeladat megoldásához tartozó Karnaugh táblák

A Karnaugh táblákon felrajzolt prímimplikánsok algebrai alakjai:

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

<u>Megjegyzés:</u> *az eredményként kapott függvényeket vizsgálva a következő megállapítást tehetjük: Z nem függ a bemenettől, csakis egyetlen szekunder változótól (Y_1). Mealy-típusú hálózat tervezésébe fogtunk, mégis annak speciális eseteként, Moore-típusú hálózatot kaptunk.*

A feladat megoldásának utolsó lépéseként el kell készíteni a kapuszintű realizációt. Ezúttal azonban még egy fontos vizsgálat meg kell, hogy előzze az áramkör felrajzolását: *a helyes kezdeti állapot beállásának ellenőrzése*.

Ennek meghatározásához először is meg kell állapítani a kezdeti állapothoz tartozó elsődleges és másodlagos változók, valamint a kimenet(ek) értékét. Példánkban az *X* bemenő változó értékét a kezdeti állapotban ’0’-nak tekintettük. Ehhez a kezdeti *X=0* értékhez az *a* (kezdeti stabil állapotnak tekintett) állapotban a *Z=0* kimeneti értéket rendeltük. Az *a* állapothoz pedig kódoláskor az $Y_1^v Y_2^v$*=00* szekunder változó kódot

rendeltük. Ezekből kiindulva pedig egyértelműen meg lehet és kell tudni állapítani, hogy szükség van-e külön kezdeti állapot beállító segédbemenetre.

Az ellenőrzéshez nem kell mást tennünk, mint a megoldásként kapott függvényekbe, mint egyenletekbe be kell helyettesíteni a kezdeti állapothoz tartozó ismert és előírt értékét. Ha egyértelműek az eredmények, és teljesül az egyenlőség, akkor nem kell külön kezdő állapot beállító jelről gondoskodni.

Példánkban az egyenletekbe történő behelyettesítés nem hoz egyértelmű eredményt, hiszen pl. Y_1 és Y_2 esetében sem látjuk biztosítva az egyenletek jobboldalán történő behelyettesítések eredményeképpen a baloldalon meghatározott '0' értékek megjelenését, mivel a bekapcsolás pillanatában a visszacsatolások miatt nem specifikált az Y_1^v és Y_2^v értéke, így a szorzatok eredménye is bizonytalan:

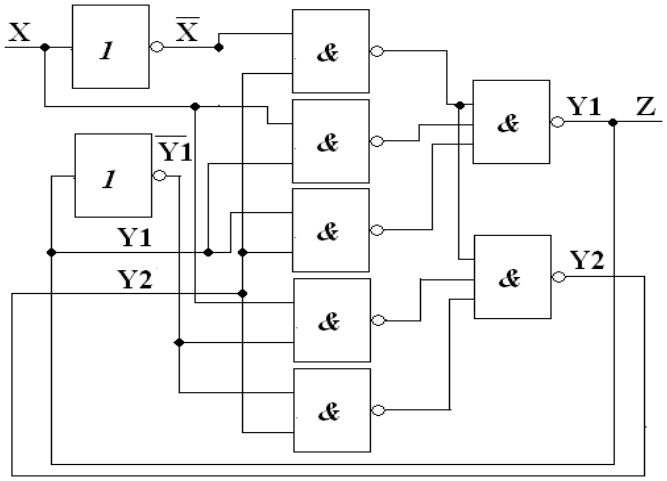
$$0 = Y_2^v \cdot 1 + Y_1^v \cdot 0 + Y_1^v Y_2^v$$
$$0 = \overline{Y_1^v} \cdot 0 + \overline{Y_1^v} Y_2^v + Y_2^v \cdot 1$$
$$0 = Y_1$$

Ebben az esetben mindenképpen célszerű a kezdeti állapot beállító jel ('*R*'=reset) alkalmazása. Az állapottáblából világosan megállapítható, hogy a tábla szerinti kezdeti állapot beállításának érdekében három feltételt kell teljesíteni:

1. a kezdeti állapot kódját rá kell kényszeríteni az f_y hálózatra, a visszacsatolásoktól függetlenítve ezeket a bemeneteket. Ezt a helyzetet legalább addig kell fenntartani, amíg az f_y kimenetein kialakul az kezdeti állapot kódja (illetve, ha SR tárolókkal történik a visszacsatolás, akkor azok kimenetén alakul ki ez a kód),
2. rá kell kapcsolni a hálózatra azt a bemeneti kombinációt, amely a kezdeti állapothoz tartozik
3. végül meg kell szüntetni a kényszerített visszacsatoló ágat (állapotot), és helyre kell állítani az eredeti visszacsatolást, ezzel a hálózat a kezdeti állapotban stabilizálódik.

(Megjegyzés: az aszinkron sorrendi hálózatok kezdeti állapotba kényszerítésének egyéb módszerei és lehetőségei egy későbbi modulban külön kerülnek megtárgyalásra.)

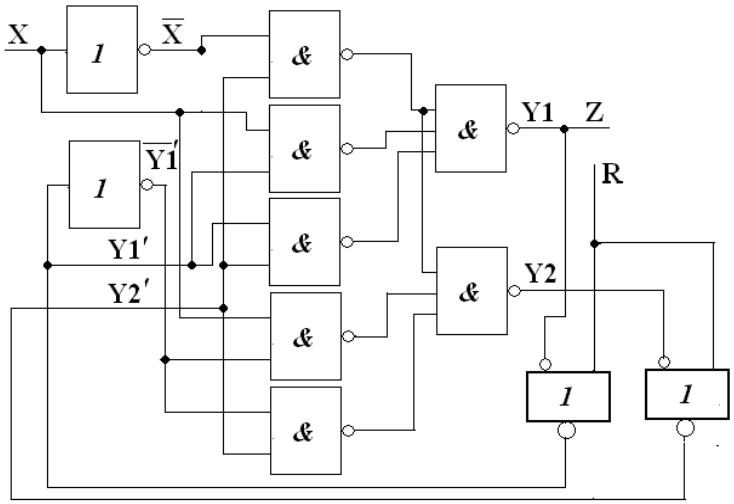
A kezdeti állapot beállító jel ('*R*') nélküli realizáció a 4.7.1.i ábrán látható:



4.7.1.i ábra[1]

Az 1.mintafeladat realizációja kezdő állapot beállítás nélkül

A 4.7.1.j ábrán látható a már felhasított visszacsatoló ágba beillesztett '*R*' kezdő állapotot beállító jel:



4.7.1.j ábra[1]

Az 1. mintafeladat realizációja a kiegészítő 'reset'(R) logikával

A ’*R*’ jel áramköri beiktatásának elve a következő: hasítsuk fel a visszacsatolást, és illesszünk be a visszacsatoló körbe 2 darab két-bemenetű logikát. Az egyik bemenetük közös, a kezdeti állapotba kényszerítő ’*R*’ (reset) jel, a másik bemenetük a visszacsatolandó szekunder változókra ($Y_1^v Y_2^v$) kapcsolandó. A kimeneteket kapcsoljuk az f_y hálózat bemeneteire. Mindkét *R*-logika az *R* = ’1’ esetben ’0’-át ad tovább, ez pedig a kezdeti állapot kódja. Ha *R=0*, a logikák kimenetére a megfelelő szekunder változó kerül, tehát él a visszacsatolás.

4.7.2 Aszinkron sorrendi hálózatok tervezése: 2.mintafeladat

Példa:

*tervezzünk kétbemenetű (**X_1,X_2**) ún. „sorrendi ÉS” áramkört, amelynek **Z** kimenete akkor és csakis akkor ad magas szintet, ha az **X_1** bemenet előbb áll ’**1**’-re, mint az **X_2**. Kezdeti állapotban az **X_1X_2=00** bemeneti bitkombináció esetén **Z=0**. A tervezést végezzük el a következő állapotot előállító hálózat közvetlen visszacsatolásával, és SR tárolókkal történő visszacsatolással is !*

Megoldás:

Korábbi gyakorlatunktól eltérően – kellő rutin birtokában - a megoldást ezúttal rögtön az előzetes szimbolikus állapottábla felvételével kezdhetjük (ebben az esetben az idődiagram nem feltétlenül olyan szemléletes kiinduló alap, és az állapottábla anélkül is megszerkeszthető.) A tábla (4.7.2.a ábra) kitöltést célszerű a stabil kezdőállapot felvételével indítani: esetünkben ez a specifikáció szerint az **X_1X_2=00**-nál a **Z=0,** és rendeljük hozzá az ***a*** szimbólumot. (A kimenet egyes állapotokhoz meghatározandó értékénél a feladat specifikációjából egyértelmű, hogy **Z** magas szintjei csak az ’**11**’ oszlopban lesznek, de csak azoknál az állapotoknál, amelyek az ’***10***’-ban levő állapotokat követik.)

Induljuk el tehát a bekapcsolás utáni kezdeti ***a*** állapotunkból, és következő lépésként vegyük számba a lehetséges bemeneti változásokat. Az ***a*** állapotban az ’**11**’ bemeneti kombinációra való áttérés nem megengedett, hiszen ebben az esetben két bemeneti változó is értéket váltana, ezt pedig megtiltjuk. Így az ’**11**’-hez tartozó következő állapot és a kimenet értéke közömbös(’**-/-**’). A ’***01***’ és az ’***10***’ 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 ’***00***’ jelentkezik, az ***a*** állapotba mehetünk vissza. Ha ’***11***’ jön, akkor egy új, ***d*** állapotot vesszük fel, és **Z**-t továbbra is alacsonyan tartjuk. A ***c*** állapotból ’***00***’-ra ***a***-ba mehetünk a **Z=0**-val, ’***11***’-re viszont az új ***e*** állapot beálltához a **Z=1** tartozik, hiszen teljesült a speciális ’ÉS’ feltétel,

X_2 az X_1-t követően emelkedett magasra. Most a két legutóbb felvett állapotról, ***d***-ről és ***e***-ről kell gondoskodni. Egyikből sem kapcsolhatunk '***00***'-ra, de a '***01***'-re a ***b / 0***, '***10***'-ra a ***c / 0*** jó választás, hiszen az előbbi esetben X_1 lefut, így a speciális 'ÉS' feltétel teljesülésének lehetősége távolabbra kerül, a második esetben viszont fennmarad.

Az előzetes szimbolikus állapottábla a 4.7.2.a ábrán látható.

aktuális állapot	következő állapot/kimenet			
	X_1X_2			
	00	*01*	*10*	*11*
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

4.7.2.a ábra

A 2.mintafeladat előzetes szimbolikus állapottáblája

A következő lépésként az állapotok számának csökkentésével próbálkozunk, az állapot összevonás kettős feltételének figyelembe vételével. Ezt a szabályt most némi kiegészítéssel élve alkalmazhatjuk, ugyanis a közömbös bejegyzések lehetőséget adnak erre. Ez esetünkben azt jelenti, hogy <u>két állapot összevonható, ha bemenő kombinációnként megegyeznek a specifikált kimeneti kombinációk, és a specifikált következő állapotok.</u> Ennek alapján a következő párok vonhatók össze: ***ab , ad , bd , ce***. Az összevont állapotok tehát : (***abd***), (***ce***). Jelöljük az (***abd***) összevont állapotot s_1-gyel, a (***ce***)-t s_2-vel. Az összevont állapottábla sorainak kitöltésénél az eredeti tábla közömbös bejegyzései okoznak gondot. Könnyen belátjuk azonban, hogy mindig azt az összevont állapotot kell beírnunk, amelyhez tartozó állapot az adott oszlopban, az összevont állapot eredeti állapotainak sorában szerepelt. Például: s_1 sorában az '***11***' oszlopban s_1-et írunk, mivel az s_1-hez tartozó állapotok specifikált következő állapota a ***d***, ami az s_1-ben szerepel.

(*Megjegyzés: általában 'sx' szimbólumokkal szokták jelölni az egyes állapotokat az ún. állapotgépes (state-machine) tervezéseknél, melyeket gyakran használnak elsősorban az ipari gyártási folyamatok szekvenciális működésének leírására. Az 's' szimbólum a 'state' szóból származik.)*

Az ezek alapján elkészített összevont szimbolikus állapottábla a 4.7.2.b ábrán látható:

aktuális állapot,	**következő állapot/kimenet**			
	X_1X_2			
	00	*01*	*10*	*11*
s_1	$s_1/0$	$s_1/0$	$s_2/0$	$s_1/0$
s_2	$s_1/0$	$s_1/0$	$s_2/0$	$s_2/1$

4.7.2.b ábra

A 2.mintafeladat összevont szimbolikus állapottáblája

A két állapot kódolásához egyetlen szekunder változó (***Y***) bevezetése elegendő: rendeljük az s_1 állapothoz ennek '0' értékét, az s_2-höz pedig az '***1***'-et. Nyilvánvaló, hogy ebben az esetben az egy szekunder változó miatt a kritikus versenyhelyzet problémája fel sem merül.

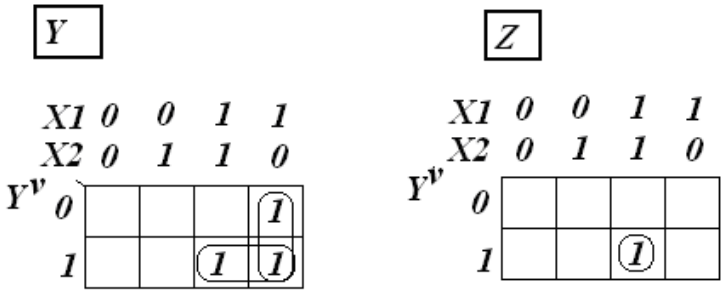
Az így kialakult kódolt állapottábla a 4.7.2.c ábrán látható:

aktuális állapot, *Y*	**következő állapot/kimenet**			
	X_1X_2			
	00	*01*	*10*	*11*
0	*0/0*	*0/0*	*1/0*	*0/0*
1	*0/0*	*0/0*	*1/0*	*1/1*

4.7.2.c ábra

A 2.mintafeladat kódolt állapottáblája

A kódolt állapottábla szerint vegyük fel és töltsük ki a Karnaugh táblákat. A lefedések alapján felírhatjuk a realizáció alapjaként szolgáló függvényalakokat (4.7.2.d ábra):



4.7.2.d ábra[1]

A 2.mintafeladat Karnaugh táblái

A szekunder változóra és a kimentre kapott algebrai kifejezések a következők:

$$Y = X_1\overline{X_2} + X_1Y^v$$
$$Z = X_1X_2Y^v$$

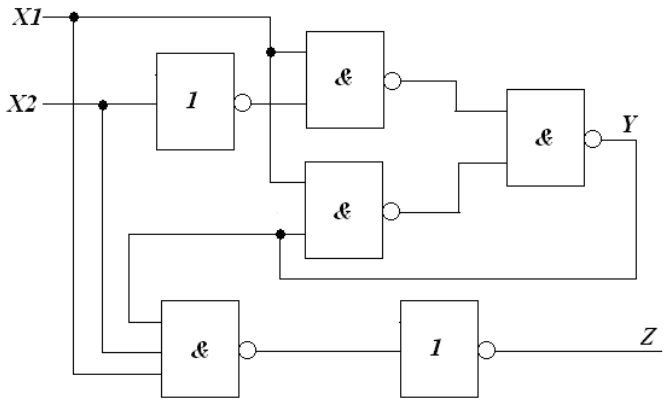
A realizáció elkészítése előtt meg kell még vizsgálnunk a kezdeti állapot beállását. Ezúttal könnyű dolgunk van ennek megállapításában, hiszen a függvényekbe behelyettesítve a megfelelő értékeket egyértelműen látszik, hogy a kezdeti állapotban (*a*) mind a másodlagos változó (*Y*), mind pedig a kimenet (*Z*) az előírt logikai értéket veszi fel:

$$0(Y) = 0\cdot 1 + 0\cdot Y^v$$

$$0(Z) = 0\cdot 0\cdot Y^v$$

vagyis nem kell külön ’*R*’ jel beiktatásáról gondoskodnunk.

A kapuszintű realizáció NEM-ÉS logikával a 4.7.2.e ábrán látható:



4.7.2.e ábra[1]

A 2.mintafeladat NAND-NAND realizációval megvalósított áramköre

A realizációt a feladat kiírása szerint el kell végezni *SR* tárolók felhasználásával is. Ehhez az előzőekben megszerkesztett kódolt állapottábla alapján fel kell venni a feladatra specifikált *SR* vezérlési táblát is. Mivel egyetlen szekunder változónk van, ezért elegendő egyetlen tároló vezérlését megadni. Az *SR* tároló saját vezérlési táblája –a már ismert módon – az összetett igazságtáblájából származtatható a 4.7.2.f ábrán látható táblázat szerint:

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

4.7.2.f ábra[1]

Az SR tároló összetett igazságtáblájából származtatott vezérlési táblája

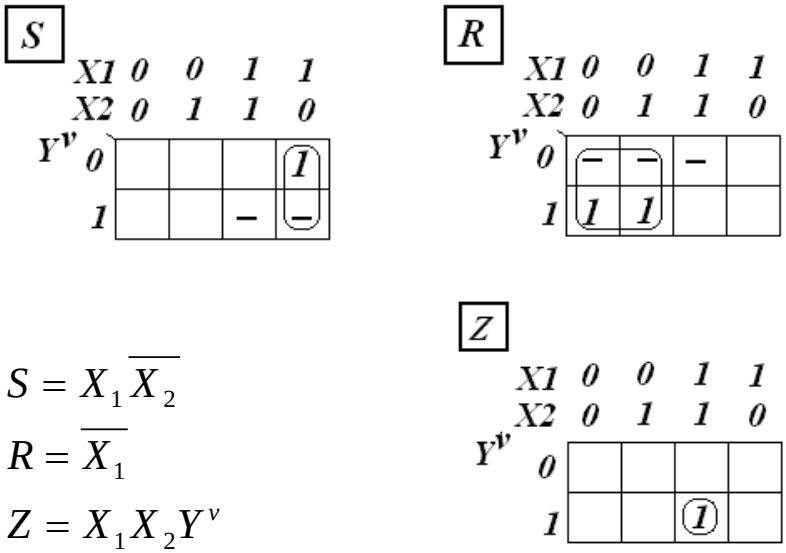
A feladatra specifikált *SR* vezérlés táblája a 4.7.2.g ábrán látható:

kódolt aktuális állapot	**kódolt következő állapot**			
	X_1X_2			
	00	*01*	*10*	*11*
Q	*S R*	*S R*	*S R*	*S R*
0	*0 -*	*0 -*	*1 0*	*0 -*
1	*0 1*	*0 1*	*- 0*	*- 0*

4.7.2.g ábra

A 2.mintafeladat szekunder változóját reprezentáló SR tároló vezérlése

A származtatott Karnaugh táblák az *S, R* és *Z* függvényekkel a 4.7.2.h ábrán látható:



$$S = X_1\overline{X_2}$$

$$R = \overline{X_1}$$

$$Z = X_1X_2Y^v$$

4.7.2.h ábra[1]

A 2.mintafeladat Karnaugh táblái és az azokból felírt S, R, Z függvényalakok

A realizáció megkezdése előtt ellenőrizzük a kezdeti állapot beállását! Ezúttal arra kell figyelnünk, hogy kialakul-e az *SR* tárolónk kimenetén a szükséges alacsony szint. Ehhez az *S* bemenetre '0'-át, az *R* bemenetre mindenképpen '1'-et kell biztosítani. Végezzük el a kapott függvényekbe a behelyettesítést! Ennek eredménye a következő:

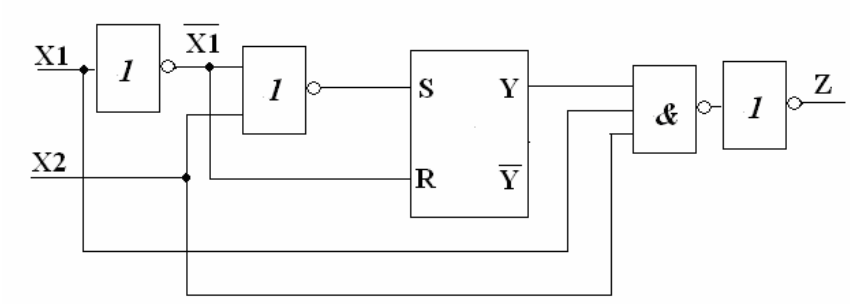
$$0(S) = 0 \cdot 1$$

$$1(R) = 1$$

$$0(Z) = 0 \cdot 0 \cdot Y^{v}$$

tehát nem kell külön '*R*' kezdeti állapot beállító jel beiktatásáról gondoskodni.

A kapuszintű realizáció NEM-ÉS logikával a 4.7.2.i ábrán látható:



4.7.2.i ábra[1]

A 2.mintafeladat SR tárolós realizációval megvalósított áramköre(NAND-NAND)

4.7.3 Lényeges hazárdok aszinkron sorrendi hálózatokban.

A hazárdjelenségek tárgyalása során osztályoztuk a hazárdokat viselkedésük szerint. Akkor a következő felsorolással éltünk:

- *statikus hazárdok*
- *dinamikus hazárdok*
- *funkcionális hazárdok.*

Az első három hazárdtípust már korábban specifikáltuk, létezik azonban egy negyedik típus is, a *lényeges hazárdok* családja. Mivel ezek a hazárdok aszinkron sorrendi hálózatokban fordulnak elő, ezért ebben a fejezetben értelmezzük és határozzuk meg pontosan, mit is takar ez a meghatározás.

Az eddigi aszinkron hálózat tervezési példáink megoldása során csak a szekunder változók versengése miatt kialakuló hibákkal, és azok kiküszöbölésével foglalkoztunk. Ez csak akkor tekinthető korrekt eljárásnak, ha garantálni tudjuk azt, hogy a bemeneti jelek változása okozta események a szekunder változók értékeinek megváltozása 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.*

Megjegyzés: a fent említett időkésleltetésekkel a feladatok megoldása során nem foglalkoztunk, mert feltételeztük a bemeneti értékek/kombinációk időben konstans jellegét a stabil-stabil állapot átmenetek vizsgálata alatt.

4.8 Sorrendi hálózatok kezdeti állapotának beállítási lehetőségei

4.8.1 Szinkron szekvenciális hálózatok kezdeti állapotának beállítása

Általánosságban elmondható, hogy a kezdeti állapotkódok 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.

Szinkron sorrendi hálózatokat mindig ’mester-szolga’ tárolókkal valósítunk meg, így a kezdeti állapot beállítása a flip-flopok kezdeti állapotainak beállítását jelenti. Ilyenkor - az adott *MS* tároló kialakításától függően – két lehetőség közül választhatunk:

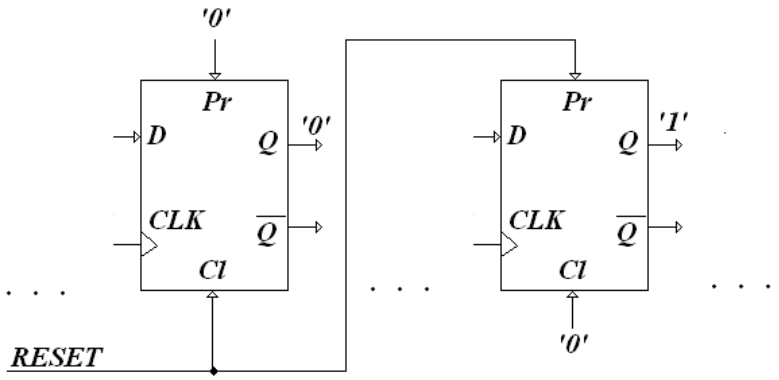
1. a kezdeti állapot beállítása a ’preset’ (Pr) és a ’clear’ (Cl) segédbemenetek kihasználásával,

2. a kezdeti állapot beállítása az f_y hálózat kiegészítésével.

4.8.1.1 Kezdeti állapot beállítása a 'preset' és 'clear' segédbemenetek felhasználásával

Szinkron hálózattervezési mintapéldáink megoldásában a kezdeti állapot beállítását lehetővé tevő kiegészítések megtervezésekor kihasználtuk a flip-flopok 'preset' és 'clear' segédbemeneteit. A kezdeti állapot kódja példáinkban mindig csupa '0'-ból állt, így a flip-flopok 'preset' bemenetét '0'-ra kapcsoltuk, és valamennyi 'clear' bemenetére rákapcsoltuk a kezdeti állapotot kikényszerítő '*R*' ('reset') bemeneti jelet.

Ezt a módszert általánosítsuk tetszőleges kezdeti állapot kódra! A 4.8.1.1.a ábra egy fiktív *D-MS* flip-flop sorban (állapot-regiszterben) két különböző kezdeti értékű flip-flopot mutat. Nyilvánvaló, hogy a '0' kezdeti értékűek 'clear', míg az '1' kezdeti értékűek 'preset' bemenetét aktivizáljuk a '*RESET*' jellel.



4.8.1.1.a ábra[1]

Példa a 'preset'(Pr) és 'clear'(Cl) segédbemenetek felhasználása a kezdeti állapot beállításához

4.8.1.2 Kezdeti állapot beállítása az f_y hálózat kiegészítésével

Amennyiben az alkalmazott tárolónk nem tartalmaz külön 'preset' és 'clear' segédbemenetet, akkor egy kiegészítő hálózat megtervezésével és beiktatásával lehet megvalósítani a szükséges kezdeti állapotot beállító 'Pr' és 'Cl' funkciót.

D-MS flip-flop kiegészítése 'Pr' és 'Cl' segédbemenettel:

A *D* flip-flopok esetén alkalmazandó kiegészítő hálózatok megtervezéséhez a szükséges igazságtáblák a 4.8.1.2.a ábrán láthatók:

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

4.8.1.2.a ábra[1]

Igazságtáblák a D-MS tároló 'Pr' és 'Cl' segédbemeneteinek tervezéséhez

A baloldali tábla azt az esetet mutatja, amikor a 'clear' bemenethez tartozó RESET jel, valamint a *D* adatbemenetre (D_i) eredetileg csatlakozó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni a 'clear' funkció elérése érdekében a *D* flip-flopunk adatbemenetére (D_i').

A jobboldali táblázat pedig azt a helyzetet mutatja, amikor a 'preset' bemenethez tartozó RESET jel, valamint a *D* adatbemenetre (D_j) eredetileg csatlakozó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni a 'preset' funkció elérése érdekében a *D* flip-flopunk adatbemenetére (D_j').

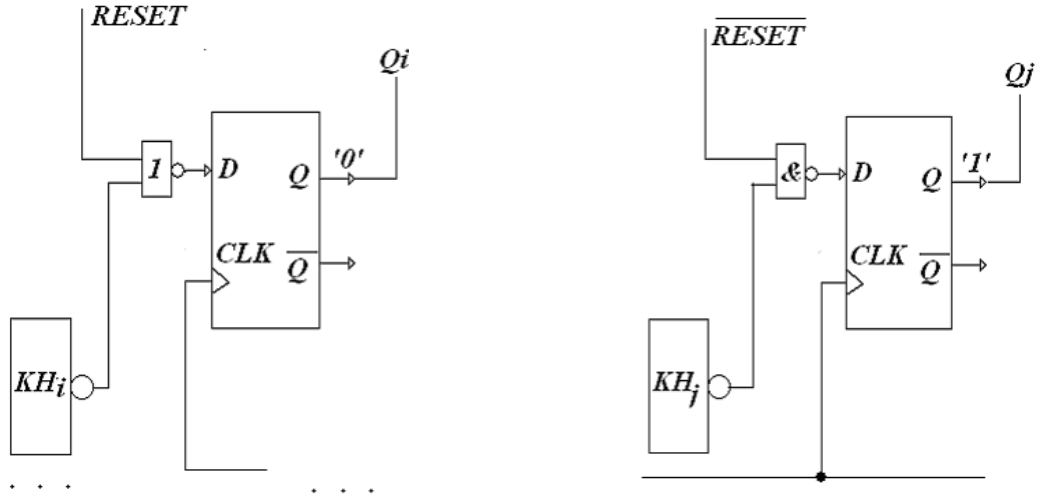
A táblázatból kiolvasott algebrai alakok:

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

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

Megjegyzés: D_j' felírása esetében érdemes a szokásos mintermes kanonikus alakot „átfordítani' az egyszerűbb formulává, vagyis a kimenet egyetlen '0' értékéhez megadni a formula negáltját, természetesen a vonatkozó De-Morgan azonosságok felhasználása alapján.

A szükséges átalakításokat tartalmazó realizációk a 4.8.1.2.b ábrán láthatók:



4.8.1.2.b ábra[1]

A 'clear' (bal oldali séma) és a 'preset' (jobb oldali séma) kiegészítő hálózatok beillesztése D flip-flopok kezdeti állapotainak beállítására. A $\overline{D}_i$ és $\overline{D}_j$ logikai szintek előállításának érdekében a KHi és KHj hálózatok végére invertereket kell tennünk, ezeket a körök szimbolizálják.

JK-MS flip-flop kiegészítése 'Pr' és 'Cl' segédbemenettel:

A *JK* flip-flopok esetén alkalmazandó kiegészítő hálózatok megtervezéséhez a szükséges igazságtábla párok a 4.8.1.2.c ábrán láthatók:

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

4.8.1.2.c ábra[1]

Igazságtáblák a JK-MS tároló 'Pr' és 'Cl' segédbemeneteinek tervezéséhez

A felső táblapár azt az esetet mutatja, amikor a 'clear' bemenethez tartozó RESET jel, valamint a *J* és *K* bemenetekre (J_i; K_i) eredetileg csatlakozó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni a 'clear' funkció elérése érdekében a *JK* flip-flopunk *J* és *K* bemenetére (J'_i; K'_i).

Az alsó táblázatpár pedig azt a helyzetet mutatja, amikor a 'preset' bemenethez tartozó RESET jel, valamint a *J* és *K* bemenetre (J_j; K_j) eredetileg csatlakozó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni a 'preset' funkció elérése érdekében a *JK* flip-flopunk *J* és *K* bemenetére (J'_j; K'_j).

A *D* flip-flop esetében alkalmazott függvényfelírási gondolatmenet alapján az egyes algebrai alakok a következőképpen adódnak:

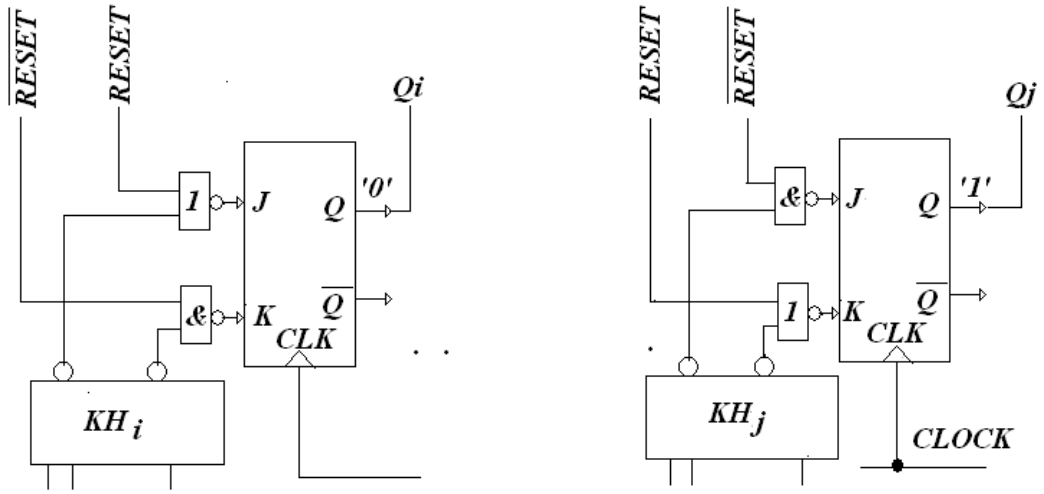
$$J'_i = \overline{\overline{J_\iota} + RESET}$$

$$K'_i = \overline{\overline{K_\iota}\,\overline{RESET}}$$

$$J'_j = \overline{\overline{J_J}\,\overline{RESET}}$$

$$K'_j = \overline{\overline{K_J} + RESET}$$

A megvalósított kapuszintű realizáció a 4.8.1.2.d ábrán látható:



4.8.1.2.d ábra[1]

A 'clear' (bal oldali séma) és a 'preset' (jobb oldali séma) kiegészítő hálózatok beillesztése JK flip-flopok kezdeti állapotainak beállítására. A $\bar{J}_i$ és $\overline{K}_i$ valamint $\bar{J}_j$ és $\overline{K}_j$ logikai szintek előállításának érdekében a KHi és KHj hálózatok végeire invertereket kell tennünk, ezeket a körök szimbolizálják.

Megjegyés: a fent bemutatott módszer minden esetben biztosítja a kezdeti állapot beállását a szekunder változók és a bemenetek aktuális állapotától függetlenül. Elvi megközelítés szerint a kezdeti állapotot a szekunder változók aktuális állapotának a módosításával is elő lehet állítani, ami ugyan egy egyszerűbb módszert jelent, de fontos megjyegyezni, hogy nem biztosítja minden esetben a bemenetektől függetlenül a kezdeti állapot beállítását, ezért fejezetünkben ezt nem is tárgyaljuk.

4.8.2 Aszinkron szekvenciális hálózatok kezdeti állapotának beállítása

Aszinkron sorrendi hálózatok kedzeti állapotának beállításánál alapvetően a már megismert kétféle visszacsatolási séma alapján kell a szükséges kiegészítő hálózatot megtervezni illetve alkalmazni:

1. a kezdeti állapot beállítása közvetlen visszacsatolású aszinkron sorrendi hálózatokban,

2. a kezdeti állapot beállítása *SR* tárolóval visszacsatolt aszinkron sorrendi hálózatokban.

4.8.2.1 A kezdeti állapot beállítása közvetlen visszacsatolású aszinkron sorrendi hálózatokban

Ezzel a módszerrel már találkoztunk a korábbi mintafeladatok megoldása során. Tekintsük most újra át a megvalósítás lényegét: szekunder változónként fel kell hasítanunk a visszacsatolást, és be kell illesztenünk egy-egy olyan kiegészítő hálózatot, amely RESET='0' esetén a kiszámolt szekunder változó (eredeti) értéket, RESET='1' esetén pedig a kívánt kezdő értéket ('0'-át vagy '1'-et) helyezi a megfelelő kimenetre. Jelöljük most ezt a kimenetet *Y'*-vel. Ezek alapján a kezdeti állapotot beállító segéd hálózatok igazságtábláit és logikai megoldásait a 4.8.2.1.a ábrán látható táblázatpár mutatja:

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

4.8.2.1.a ábra[1]

A 'RESET' logika értelmezése közvetlenül visszacsatolt aszinkron sorrendi hálózat kezdeti állapotainak beállításához

A baloldali tábla azt az esetet mutatja, amikor a ’0’ szükséges kezdeti állapotot beállító RESET jel, valamint az eredetileg visszacsatolódó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni a ’0’ logikai értékkel bíró kezdeti állapot elérése érdekében a visszacsatoló jelet fogadó hálózat bemenetére.

A jobboldali tábla pedig azt az esetet mutatja, amikor az ’1’ szükséges kezdeti állapotot beállító RESET jel, valamint az eredetileg visszacsatolódó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni az ’1’ logikai értékkel bíró kezdeti állapot elérése érdekében a visszacsatoló jelet fogadó hálózat bemenetére.

Természetesen a fent bemutatott gondolatmenet alapján adódó logikai implementáció mindig az adott feladat specifikációja szerint illesztendő a megvalósítandó realizációba.

Megjegyzés: a kész realizáció ellenőrzésekor (pl. szimuláció) arra is kell ügyelni, 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.

4.8.2.2 Kezdeti állapot beállítása SR tárolókkal visszacsatolt aszinkron sorrendi hálózatokban

Az *SR* tárolókkal visszacsatolt aszinkron sorrendi hálózatok esetében is a kezdeti állapot biztosításához sokszor elengedhetetlen feltétel, hogy a tárolók kimeneteit is a megfelelő logikai értékre állítsuk. ’Preset’ és ’clear’ segédbemenetekkel nem rendelkező *SR* tárolók esetében a kezdeti értéket beállító jelünket fogadó kiegészítő hálózat megtervezéséhez nyújt segítséget a RESET logikán felépülő igazságtábla pár, amely a 4.8.2.2.a ábrán látható:

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

4.8.2.2.a ábra[1]

Igazságtáblák az SR tároló 'Pr' és 'Cl' segédbemeneteinek tervezéséhez

A felső táblapár azt az esetet mutatja, amikor a 'clear' bemenethez tartozó RESET jel, valamint az *S* és *R* bemenetekre ($S_i; R_i$) eredetileg csatlakozó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni a 'clear' funkció elérése érdekében az *SR* tárolónk *S* és *R* bemenetére (S_i'; R_i').

Az alsó táblázatpár pedig azt a helyzetet mutatja, amikor a 'preset' bemenethez tartozó RESET jel, valamint az *S* és *R* bemenetre (S_j; R_j) eredetileg csatlakozó f_y hálózati által megadott értékkombinációk feltüntetésével meghatározzuk, hogy milyen szükséges logikai szintet kell kapcsolni a 'preset' funkció elérése érdekében az *SR* tárolónk *S* és *R* bemenetére (S_j'; R_j').

A *D* flip-flop esetében alkalmazott függvényfelírási gondolatmenet alapján az egyes *S* és *R* algebrai alakok – a *JK* flip-flop módosult *J* és *K* bementére kapott kifejezésekkel hasonlóságot mutatva - a következőképpen adódnak:

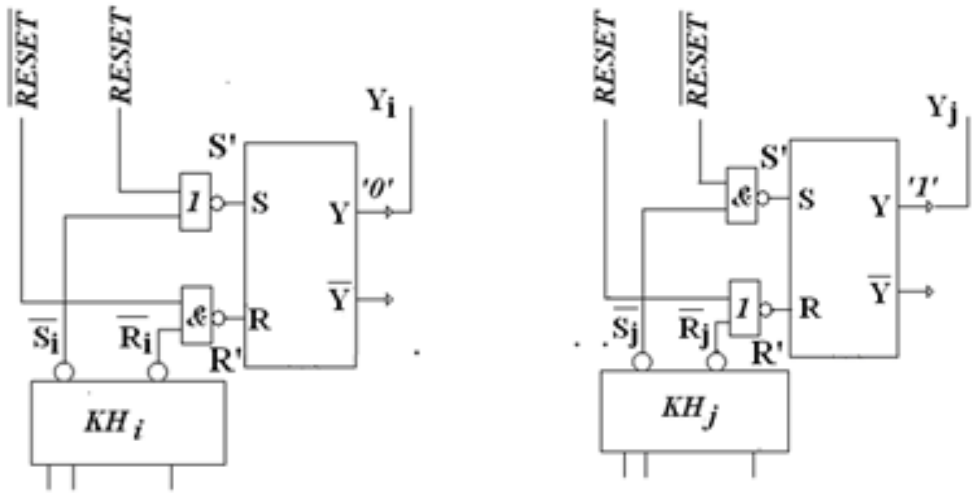
$$S_i' = \overline{\overline{S_\iota} + RESET}$$

$$R_i' = \overline{\overline{R_\iota}\,\overline{RESET}}$$

$$S_j' = \overline{\overline{S_J}\,\overline{RESET}}$$

$$R_j' = \overline{\overline{R_J} + RESET}$$

A realizáció a 4.8.2.2.b ábrán látható:



4.8.2.2.b ábra[1]

A 'clear' (bal oldali séma) és a 'preset' (jobb oldali séma) kiegészítő hálózatok beillesztése SR tároló kezdeti állapotainak beállítására. A $\bar{S}_i$ és $\bar{R}_i$ valamint $\bar{S}_j$ és $\bar{R}_j$ logikai szintek előállításának érdekében a KHi és KHj hálózatok végeire invertereket kell tennünk, ezeket a körök szimbolizálják.

Megjegyzés: ez a módszer csak a kezdeti állapothoz tartozó bemeneti érték/kombináció biztosítása mellett garantálja a stabil kezdeti állapot beállítását! Elvi lehetőség van még a szekunder változók módosítása által beállítani a szükséges kezdeti állapotot az SR tárolókkal visszacsatolt aszinkron sorrendi hálózatok esetében, de ennek hátránya, hogy a kialakítás a bemenetekre megadott kezdeti bemeneti kombinációval együtt sem biztosítja mindig a kezdeti állapot beállítását.

5. Állapot összevonási módszerek

A digitális hálózatok tervezése során elsődleges szempont a minimalizálás, vagyis a lehető legkevesebb alkatrész felhasználása, illetve legkisebb (legkedvezőbb) hálózati struktúra megalkotása. A hálózattervezési lépések során többször is alkalmunk nyílik erre: többek között ide tartozik az előzetes szimbolikus állapottáblán felvett állapotok számának csökkentése. Ezt állapot összevonással érhetjük el. Az állapotok

összevonhatóságával kapcsolatban lefektettünk egy általános szabályt, mely szerint *az előzetes állapottábla két állapotát nem kell megkülönböztetni, ezért azok összevonhatók, ha*

1) *bemeneti kombinációnként egyeznek a hozzájuk rendelt kimeneti kombinációk,*

2) *és bemenő kombinációnként ugyanarra a következő állapotra vezetnek.*

Ezt az általános szabályt terjesszük ki a hálózatok specifikus mélységére vonatkozóan, azaz - feltételrendszerét tekintve - válasszuk szét a teljesen specifikált előzetes szimbolikus állapottáblával leírható hálózatok csoportját a nem teljesen specifikált előzetes szimbolikus állapottáblával jellemzett hálózatok csoportjától.

5.1 Állapot összevonás teljesen specifikált előzetes szimbolikus állapottáblán

5.1.1 Az ekvivalencia-típusú reláció

Az előzetes szimbolikus állapottáblát akkor tekintjük teljesen specifikáltnak, ha nem tartalmaz egyetlen „közömbös" bejegyzést sem. Ez alapján az állapot összevonás feltétele teljesen specifikált előzetes szimbolikus állapottáblán az alábbiak szerint fogalmazható meg:

két állapot a teljesen specifikált hálózat állapottábláján nem megkülönböztethető, ha a két állapotból elindulva bármely bemeneti sorozatra ugyanazt a kimeneti sorozatot látjuk.

Ha a teljesen specifikált hálózat állapotpárjait megvizsgáljuk, azt tapasztaljuk, hogy a „nem megkülönböztethető" pár-alkotást, mint *relációt* a következő tulajdonságok jellemzik:

1) *reflexív,*
2) *szimmetrikus*
3) *tranzitív.*

Az egyes tulajdonságok jelentése a következő:

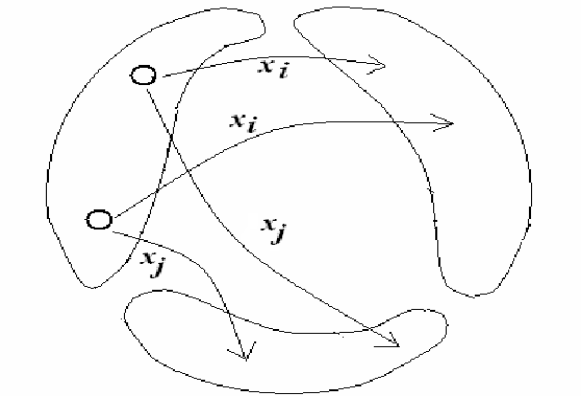
1) a *reflexivitás* az a trivialitás, hogy egy szimbolikus állapot saját magától nem különböztethető meg, azaz $\boldsymbol{a \equiv a}$.

2) *szimmetrikusak* azok a bináris relációk, amelyekre igaz, hogy amennyiben $\boldsymbol{a \equiv b}$, akkor bizonyosan fennáll a $\boldsymbol{b \equiv a}$ reláció is.

3) *tranzitív* relációk esetén igaz, hogy amennyiben $\boldsymbol{a \equiv b}$ és $\boldsymbol{b \equiv c}$, akkor teljesül az $\boldsymbol{a \equiv c}$ is.

Magától értetődő, hogy *a teljesen specifikált állapottáblákra definiált nem megkülönböztethetőség reflexív, szimmetrikus és tranzitív*. Az ilyen relációkat ***ekvivalencia-típusú relációknak*** nevezzük. Szokás a teljesen specifikált hálózatnak ezt a tulajdonságát ***állapot ekvivalenciának*** is nevezni, azaz a nem megkülönböztethető állapotpárok tagjait ekvivalenseknek mondjuk.

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 állapot ekvivalencia a teljesen specifikált hálózat állapothalmazát olyan, közös elemeket nem tartalmazó részhalmazokra bontja fel, amelyek az összevont állapothalmazt alkotják. Ezt szemléletesen mutatja az 5.1.a 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.



5.1.a ábra[1]

Teljesen specifikált hálózat diszjunkt részhalmazai: példa két állapot x_i és x_j hatására képződő utódállapotainak azonos új osztályba történő származtatására

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

- $\boldsymbol{a \equiv b}$: $\boldsymbol{a}$ és $\boldsymbol{b}$ állapotok ekvivalensek,
- $\boldsymbol{a <> b}$: $\boldsymbol{a}$ és $\boldsymbol{b}$ állapotok antivalensek

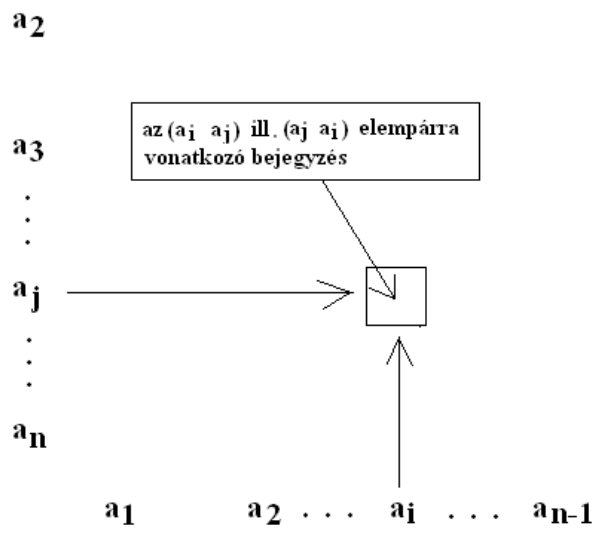
Az állapottábla analízise során sokszor nem lehet eldönteni azonnal az ekvivalencia vagy az antivalencia fennállását. Ezért, ha nem látjuk két állapotról azonnal, hogy antivalensek, akkor feljegyezzük azokat a feltételeket, amelyek fennállása esetén a két állapot ekvivalens.

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 $a \equiv b$ és $c \equiv d$.

Megjegyzés: 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.

5.1.2 A lépcsős tábla

Ha egy halmaz *n* elemből áll, akkor egy $n_x n$ méretű négyzetrács négyzeteibe bejegyezhetjük egy bináris reláció teljesülését, nem teljesülését, vagy a teljesülés feltételeit. Ha a reláció szimmetrikus, elég ehhez az átló mentén felezett négyzetrács tábla, amit jellegzetes alakjáról *lépcsős táblának* nevezünk. A lépcsős tábla formája egy $\boldsymbol{a_1, a_2, \ldots a_n}$ elemhalmaz esetén az 5.1.2.a ábrán látható. A lépcsős táblás állapot összevonás természetesen nem áll meg a páronként értelmezett ekvivalencia megállapításánál. Ha a tranzitivitást a megállapított állapotpárok között érvényesítjük, akkor kialakulnak a maximális ekvivalencia osztályok, azaz azok a diszjunkt állapot-halmazok, amelyeknél nagyobbakat már nem lehet találni. Így valamennyi állapot bekerül egy ekvivalencia osztályba, és nincs egyetlen olyan állapot sem, amely egynél több osztályba bekerülne.



5.1.2.a ábra[1]

A lépcsős tábla struktúrája

Példa:

keressünk állapot összevonási lehetőségeket az 5.1.2.b ábrán látható előzetes szimbolikus állapottáblán!

aktuális állapot	következő állapot/kimenet	
	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**

5.1.2.b ábra

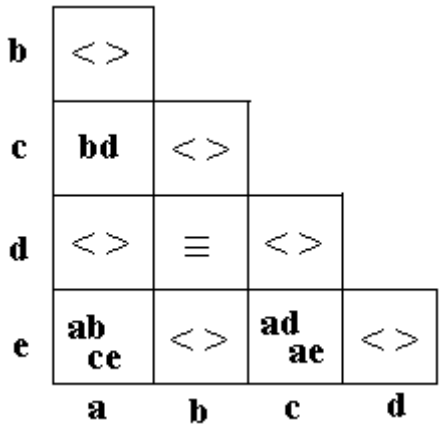
Mintapélda: egy teljesen specifikált hálózat előzetes szimbolikus állapottáblája

A fenti táblázat alapján készül el az *'elsőgenerációs' lépcsős tábla*, amely az 5.1.2.c ábrán látható. Kitöltésekor a következő módszert alkalmazzuk:

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

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

- ha a kimeneti értékek bemeneti kombinációnként megegyeznek, de a következő állapotok legalább egy bemeneti kombinációnál eltérnek, akkor a 'feltétel' bejegyzés kerül a keresztezési cellába. Egynél több eltérés esetén a részfeltételek 'ÉS' kapcsolatba kerülnek.



5.1.2.c ábra[1]

A mintafeladathoz tartozó 'elsőgenerációs' lépcsős tábla

Példaképpen nézzünk meg néhány bejegyzést:

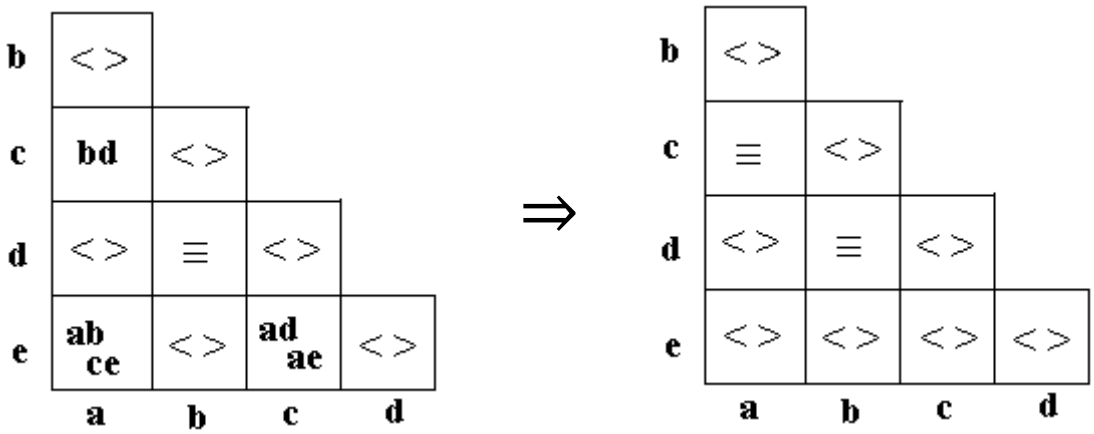
- az elsőként vizsgált *(***a b**) párról azonnal megállapítható az antivalencia, hiszen mind az *X = 0*-ra, mind az *X = 1*- re más és más kimeneti szintet ad a tábla, vagyis a keresztezési cellába a **<>** jelölés kerül,
- ugyanakkor az (**a c**) pár vizsgálatakor a kimenetekkel nincs baj, de a következő 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 példákban bemutatott vizsgálatok és konklúzióik alapján kitöltjük a táblázat többi celláját is, így teljessé válik az összes lehetséges állapotpárra vonatkoztatott összehasonlítás sor.

A második-generációs lépcsős táblát az elsőből kiindulva alkotjuk meg a következő szabályok alapján (az 5.1.2.d. ábra baloldala az elsőgenerációs lépcsős tábla, a jobboldali az ebből előállítható második-generációs lépcsős tábla):

1) 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,

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

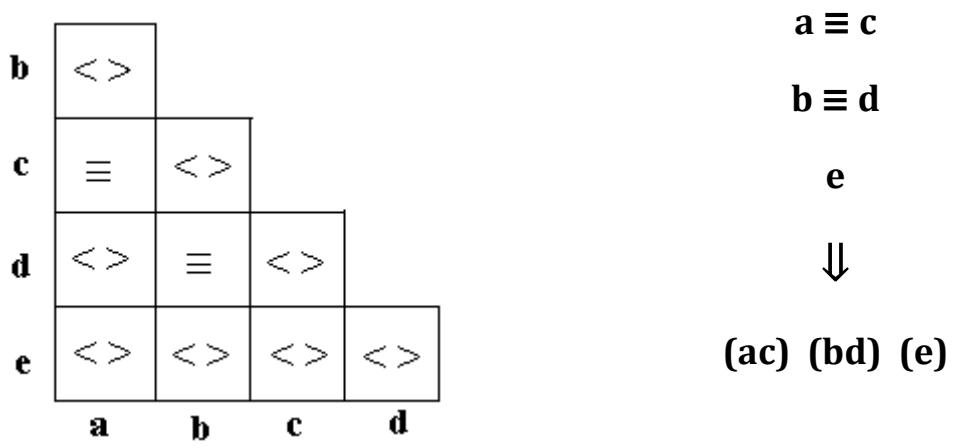


5.1.2.d ábra[1]

Az első- és második-generációs lépcsős tábla

A második-generációs lépcsős táblában az elsőgenerációs feltétele érvényesülése a következő: az (**a b**) antivalenciájának következménye az (**a e**) pár antivalenciája, az (**a d**) antivalenciájé pedig az (**e c**) antivalencia. A (**b d**) ekvivalencia lehetővé teszi az (**a c**) ekvivalenciát.

Ez a lépcsős tábla már kizárólag ekvivalencia és antivalencia bejegyzéseket tartalmaz, így az ekvivalens párok és a tranzitivitás figyelembe vételével a maximális ekvivalencia osztályok könnyen kialakíthatók (5.1.2.e ábra):



5.1.2.e ábra

A mintafeladat maximális ekvivalencia osztályai illetve állapotai

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

Legyen az összevont állapotok jelölése:

$$(\text{ac}) \rightarrow s_1$$

$$(\text{bd}) \rightarrow s_2$$

$$\text{e} \rightarrow s_3$$

Ez alapján az összevont szimbolikus állapottábla a 5.1.2.f ábrán látható:

aktuális állapot	következő állapot/kimenet	
	X=0	X=1
s_1	$s_1/0$	$s_2/1$
s_2	$s_2/1$	$s_3/0$
s_3	$s_3/0$	$s_1/1$

5.1.2.f ábra

A mintafeladat összevont szimbolikus állapottáblája

Ettől a ponttól kezdve a feladat megoldásának további menete a már jól ismert módszer alapján folytatható.

Összefoglalva az állapot összevonás fő lépései teljesen specifikált előzetes állapottáblából:

1. *az ekvivalens és antivalens állapotpárok megkeresése lépcsős tábla segítségével,*
2. *a maximális ekvivalencia osztályok meghatározása,*
3. *a maximális ekvivalencia osztályoknak megfelelő állapotokkal az*
4. *összevont állapottábla elkészítése.*

5.2 Állapot összevonás nem teljesen specifikált előzetes szimbolikus állapottáblán

5.2.1 A kompatibilitás-típusú reláció

Az előzetes szimbolikus állapottáblát akkor tekintjük nem teljesen specifikáltnak, ha tartalmaz (akár egyetlen) „közömbös" bejegyzést vagy a kimenet(ek), vagy a következő állapotokat illetően.

A táblázat egyes bejegyzéseire a következő meghatározásokkal élhetünk: egy nem teljesen specifikált szimbolikus előzetes állapottáblával megadott hálózat adott állapotához tartozó specifikációs bemeneti sorozat az, amelyre a hálózat minden állapotátmenete és kimenete specifikálva van. Két szimbolikus állapot a nem teljesen specifikált hálózat állapottábláján csak akkor megkülönböztethető, 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 nem megkülönböztethető. Továbbá 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.

A nem teljesen specifikált hálózat állapotpárjaira érvényes nem megkülönböztethetőség, mint bináris reláció a következő tulajdonságokat mutatja:

1) reflexív
2) szimmetrikus
3) nem tranzitív

Az ilyen relációkat ***kompatibilitás-típusú relációknak*** nevezzük A nem teljesen specifikált hálózat állapotpárjaira fennálló 'nem megkülönböztethetőség' tulajdonságot röviden *kompatibilitásnak*, illetve a pár tagjait *kompatibilis állapotoknak* fogjuk nevezni.

A kompatibilitás elégséges feltételei a következők:

- 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~ b : **a** és **b** állapotok kompatibilisek

a /~b : **a** és **b** állapotok nem kompatibilisek

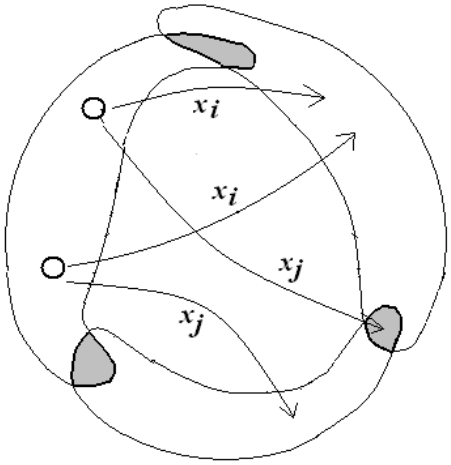
Feltételes kompatibilitás: *ab, cd …* az a két állapot, melyekre ez a bejegyzés vonatkozik, feltételesen kompatibilis, azaz csak akkor kompatibilis, ha *a~b* és *c~d …*

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:

1) az egyik a teljes lefedettség, azaz valamennyi állapotnak legalább egy osztályban szerepelnie kell,
2) 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. Az 5.2.1.a. ábra a zártságot szemlélteti.



5.2.1.a ábra[1]

A kompatibilitási reláció jellegzetessége: a nem diszjunkt részhalmazok zártsága

Példa:

a bal felső osztályból két állapotot jelöltünk ki: ezek közül az egyik utódállapota az x_j bemeneti kombinációra két másik osztály közös részében van, de ezek közül az egyik osztály azonos azzal az osztállyal, amelyben a másik állapot x_j-re adott utódja helyezkedik el.

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

a) vagy kevesebb kompatibilitási osztályból álló,

b) vagy az egyes osztályokban kevesebb állapotból álló osztályhalmazt 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. A fentiekből következik, 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.

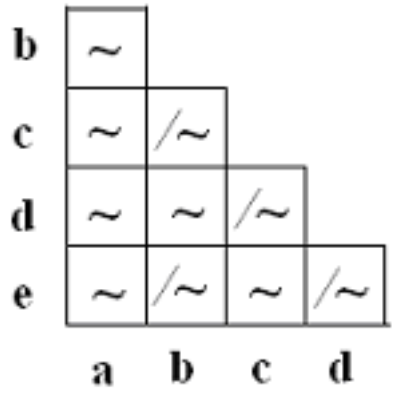
Kövessük egy korábbi, 'sorrendi ÉS' hálózatot reprezentáló mintapéldán (4.7.2) keresztül az elméletben megfogalmazott módszer gyakorlati alkalmazását az 5.2.1.b ábrán látható nem teljesen specifikált hálózat állapottáblájából kiindulva:

aktuális állapot	**következő állapot/kimenet**			
	X_1X_2			
	00	*01*	*10*	*11*
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*

5.2.1.b ábra

Mintapélda: egy nem teljesen specifikált hálózat előzetes szimbolikus állapottáblája

Készítsük el a tábla alapján a szükséges jelöléseket alkalmazva az elsőgenerációs lépcsős táblát (5.2.1.c ábra):

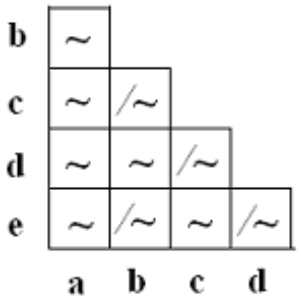


5.2.1.c ábra[1]

A mintafeladat elsőgenerációs lépcsőst táblája

Az eredményt vizsgálva megállapíthatjuk, hogy táblázatunk nem tartalmaz feltételes bejegyzéseket, így további táblázatot már nem tudunk ebből származtatni.

A táblázatból olvassuk ki a lehetséges kompatibilitási párokat, és írjuk fel a maximális kompatibilitási osztályokat is (5.2.1.d ábra):



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

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

5.2.1.d ábra[1]

A mintafeladat lehetséges kompatibilitási párjai, és maximális kompatibilitási osztályai

A megszerkesztett lépcsős tábla alapján 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:

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

2) és az (**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.

A kétféle összevonási lehetőségből adódó megoldásokra mutatnak egyenként példákat az alábbi alternatívák:

1) (**a b d**) $\rightarrow S_1$
(**c e**) $\rightarrow S_2$

2) (**a c e**) $\rightarrow S_1$
(**b d**) $\rightarrow S_2$

és a hozzájuk tartozó összevont szimbolikus állapottáblák (5.2.1.e és 5.2.1.f ábrák):

1. csoportosítás:

aktuális állapot	következő állapot/kimenet			
	X_1X_2			
	00	*01*	*10*	*11*
S_1	$S_1/0$	$S_1/0$	$S_2/0$	$S_1/0$
S_2	$S_1/0$	$S_1/0$	$S_2/0$	$S_2/1$

5.2.1.e ábra

Állapot összevonási lehetőség a mintafeladatban: 1.csoportosítás

2. csoportosítás

aktuális állapot	következő állapot/kimenet			
	X_1X_2			
	00	*01*	*10*	*11*
$S_1/0$	$S_1/0$	$S_2/0$	$S_1/0$	$S_1/1$
$S_2/0$	$S_1/0$	$S_2/0$	$S_1/0$	$S_2/0$

5.2.1.f ábra

Állapot összevonási lehetőség a mintafeladatban: 2.csoportosítás

Utolsó lépésként a kódkiosztással, valamint a Karnaugh táblás függvény kiolvasással elvégezhető a kapuszintű realizáció.

Megjegyzés: az idézett példában a korábban bemutatottól (4.7.2) eltérő állapot összevonási lehetőséget is találtunk, de természetesen mindkettő helyes megoldáshoz vezet.

<u>Összefoglalva</u> az állapot összevonás fő lépései nem teljesen specifikált előzetes állapottáblából:

1. *valamennyi kompatibilis és inkompatibilis állapotpár megkeresése a lépcsős tábla segítségével,*
2. *a lépcsős tábla alapján a maximális kompatibilitási osztályok megkeresése,*
3. *a kompatibilitási osztályok legkedvezőbb zárt halmazainak megkeresése,*
4. *a legkedvezőbb zárt halmaz osztályaihoz egy-egy állapotot rendelve az összevont állapottábla megszerkesztése.*

6. Állapotkódolási módszerek

6.1 Szinkron sorrendi hálózatok állapotkódolási módszerei

Szinkron sorrendi hálózatok tervezése során az állapotok megkülönböztetésére generált kódok esetében nincs versenyhelyzet veszély, így az állapotkódolás arra irányul, hogy a

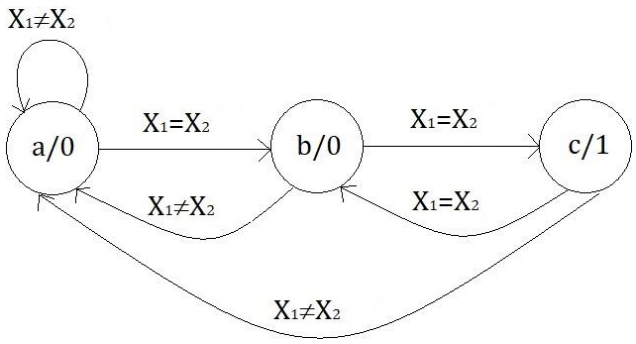
legegyszerűbb struktúrát alakítsuk ki. Ebben a fejezetben néhány gyakrabban használt kódolási eljárás kerül bemutatásra.

6.1.1 ’1’-es súlyú állapotkódolás elve ’(bit per state’)

Korábbi szinkron sorrendi hálózattervezési példáink során már találkoztunk azzal a speciális kódolási eljárással (4.5.3.2.), melynek során az egyes állapotokhoz rendelt kódszóban csak egyetlen bit helyén szerepel ’1’-es. Ezt a kódolási eljárást neveztük ’1’-es súlyú kódolásnak (’bit per state’, one hot encoding’). Azt is megemlítettük, hogy elsősorban szinkron sorrendi hálózatok VLSI megvalósításakor alkalmazzák ezt a gyorsan célravezető módszert, hiszen ebben az esetben minden egyes szimbolikus állapothoz egy D–MS flip-flopot rendelünk.

Megjegyzés: elsősorban FPGA-k (Field-Programmable Gate Array - programozható logikai kapukat tartalmazó hálózat) esetében szokásos tervezési módszer az ’1’-es súlyozású kódolás használata, hiszen ezekben az áramkörökben nagyszámú regiszter(’egybites’ D-latch) található, de viszonylag kevés bemenetszámú a hozzá tartozó kombinációs hálózat. Ezért itt sokkal fontosabb, hogy egyszerű legyen a vezérlő függvény, aminek felírását mutattuk be egy korábbi mintapéldán (4.5.3.2.) keresztül.

A 4.5.1.2 pontban bemutatott mintafeladat állapotainak megkülönböztetésére ’1’-es súlyozású kódokat is alkalmazhatunk. A feladat Moore-típusú gráfja a 6.1.1.a ábrán látható:



6.1.1.a ábra

Az első szinkron sorrendi hálózati mintafeladat Moore-típusú állapotgráfja

Alkalmazzunk 3 darab D flip-flopokat a realizációhoz, és kódoljuk az egyes állapotokat a következők szerint:

	Q_a	Q_b	Q_c
a :	1	0	0
b :	0	1	0
c :	0	0	1

A D bemenetek vezérlésének megvalósítása ilyenkor rendkívül egyszerű, és az állapotgráf alapján – a korábban már megismert módon – 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 a kezdeti (a) alapállapotot beállító ’R’ jelet is azonnal beilleszthetjük a függvényekbe.

Ezek alapján az egyes tároló vezérlések a következők:

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

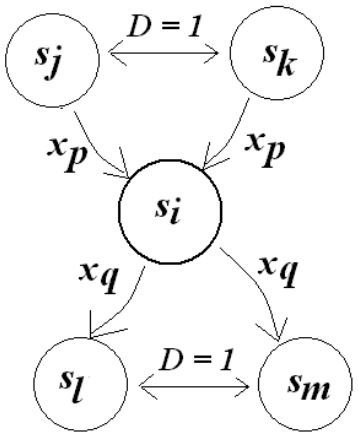
ahol ’E’ a két bemenet(x_1;x_2) EXNOR függvénye.

6.1.2 Szomszédos kódolás elve

A szomszédos állapotok kódolásának elvét más néven heurisztikus módszernek is nevezik, mert a gyakorlati tervezés során tapasztalható, hogy az f_y hálózat egyszerűsítésére jótékony hatással vannak bizonyos állapotkód viszonyok. Egyszerűbb lesz a hálózat, ha egy adott állapot következő állapotainak kódjai bemenő kombinációnként szomszédosak, illetve 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.

Példaként tekintsük a 6.1.2.a ábrán szereplő állapotgráffal és egy lehetséges előzetes szimbolikus állapottáblával megadott szinkron sorrendi hálózatot:



aktuális állapot	következő állapot	
	X=0	*X=1*
a	*c*	*d*
b	*a*	*e*
c	*a*	*b*
d	*c*	*e*
e	*a*	*b*

6.1.2.a ábra[1]

Szomszédos állapotkódok és egy előzetes szimbolikus állapottábla

Az ábra bemutatja a szomszédos kódolást kifejező állapotgráfon a leírt előnyös kódolási helyzeteket. Az ábra jobb oldalán egy előzetes szimbolikus 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 Karnaugh táblán a szomszédosság előnyösen szemléltethető, a szekunder változók számának megfelelő méretű Karnaugh táblán ábrázolhatjuk a feltételek szerint valamilyen mértékben teljesített követelményeket, és az állapotkódokat egyszerűen kiolvashatjuk.

A példa szerinti állapottáblát a 6.1.2.b ábrán látható formába dolgozzuk át, azaz elkészítjük az állapotok utódjait és az állapotok elődeit bemutató táblázatokat:

állapotok	állapotok: „utódok”	
	X=0	*X=1*
a	*c*	*d*
b	*a*	*e*
c	*a*	*b*
d	*c*	*e*
e	*a*	*b*

állapotok	állapotok: „elődök”	
	X=0	*X=1*
a	*b,c,e*	
b		*c,e*
c	*a,d*	
d	*a*	
e		*b,d*

6.1.2.b ábra

Utódok és elődök a 6.1.2.a ábra állapottáblája alapján

Az állapotkódok optimális elhelyezését a 6.1.2.c ábrán szereplő táblázatok mutatják:

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	

6.1.2.c ábra[1]

Az állapotok elhelyezése Karnaugh táblán, az előnyös szomszédosságok legnagyobb arányú biztosítására

Kódoljuk be az állapotokat az alábbiak szerint!

	Q_1	Q_2	Q_3
a :	1	1	1
b :	0	1	0
c :	0	0	1
d :	0	1	1
e :	0	0	0
-	1	0	0
-	1	1	0
-	1	0	1

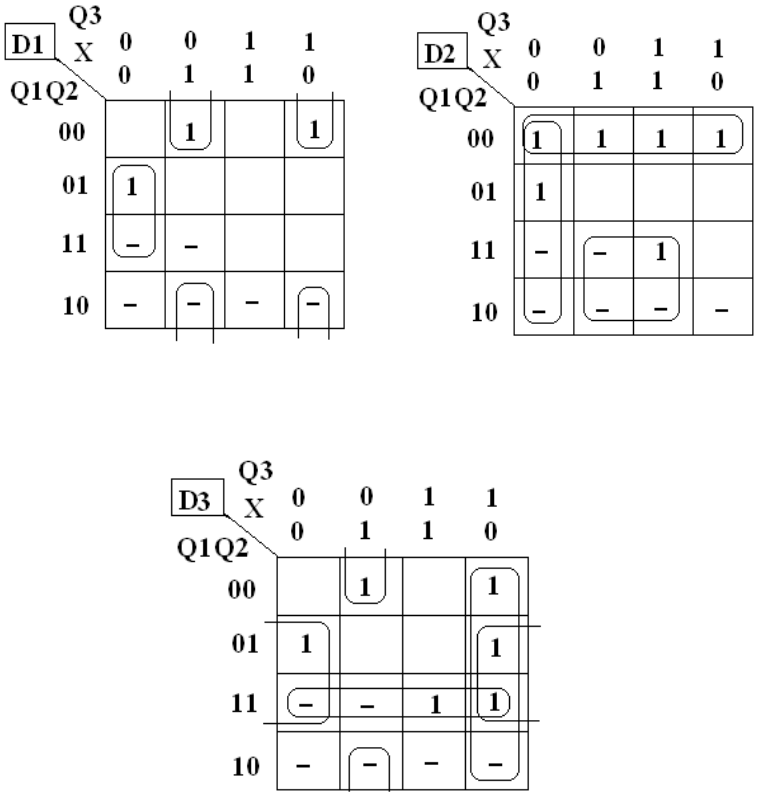
Ebből írjuk fel a kódolt állapottáblát (6.1.2.d ábra):

kódolt aktuális állapot,			kódolt következő állapot					
			X = 0			X = 1		
Q_1^n	Q_2^n	Q_3^n	D_1	D_2	D_3	D_1	D_2	D_3
1	1	1	0	0	1	0	1	1
0	1	0	1	1	1	0	0	0
0	0	1	1	1	1	0	1	0
0	1	1	0	0	1	0	0	0
0	0	0	0	1	0	1	1	1
1	0	0	-	-	-	-	-	-
1	1	0	-	-	-	-	-	-
1	0	1	-	-	-	-	-	-

6.1.2.d ábra

A kódolt állapottábla

A kódolt állapottáblából felírt Karnaugh táblák a 6.1.2.e ábrán láthatók:



6.1.2.e ábra[1]

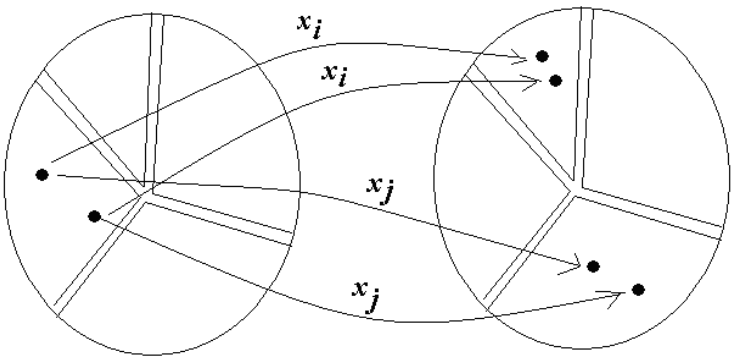
A kódolt állapottábla alapján megszerkesztett Karnaugh táblák, és a bennük létrehozott prímimplikánsok

6.1.3 A helyettesítési tulajdonságú (HT) partíciók alapján végzett kódolás elve: önfüggő szekunder változó csoportok keresése és kialakítása

Ez a módszer az állapothalmazon értelmezett partíciópárok elméletén alapul. A 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 partíciópárt 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 partíciópárnak. A komponens partíciók metszete a legfinomabb partíciót eredményezi. Ha egy komponens bemeneteire nem kapcsolódnak más komponensek állapotváltozói, akkor a komponens partíció önmagával alkot partíciópárt (6.1.3.a ábra). Az ilyen partíciót helyettesítési tulajdonságú (HT) partíciónak nevezik.



6.1.3.a ábra[1]

A HT partíció önmagával partíció-párt 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.

Megjegyzés: ebben a leckerészben az önfüggő szekunder változó csoport keresés alapján végzett kódolási eljárás elméleti hátterének bemutatására kerül sor, gyakorlati példán keresztüli szemléltetését az alábbi linken lehet megtekinteni: http://www.sze.hu/~somi/Digit%e1lis%20h%e1l%f3zatok/

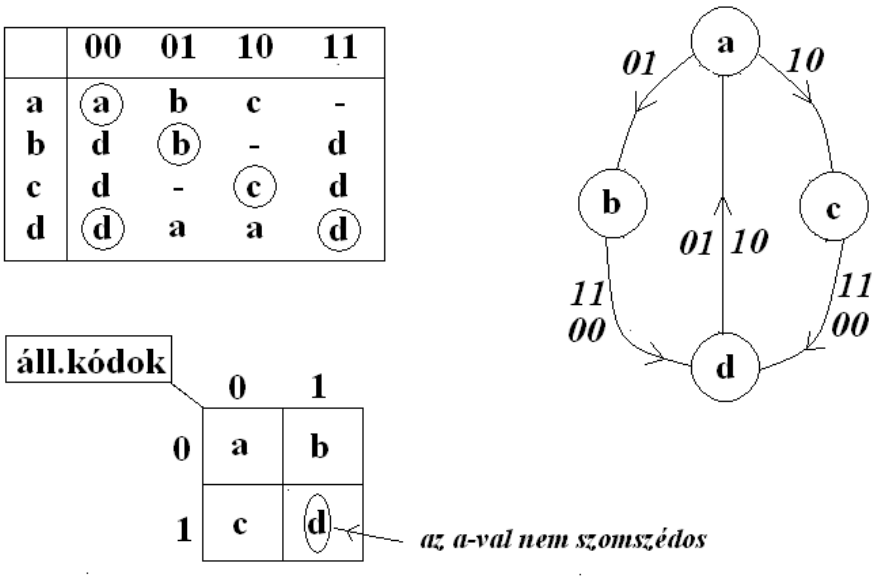
6.2 Aszinkron sorrendi hálózatok állapotkódolása

Aszinkron sorrendi hálózatok állapotkódolásakor a legfontosabb feladat a kritikus versenyhelyzetek elkerülése. Ennek eléréséhez kétféle állapotkódolási módszert lehet alkalmazni. Az egyik inkább próbálgatásos, ún. intuitív módszer(6.2.1), a másik egy elméletileg megalapozott, szisztematikus eljárás(6.2.2.).

6.2.1 Instabil állapotok beillesztésének elve (intuitív módszer)

Az eddigi feladatmegoldásoknál – aszinkron sorrendi hálózatok esetében – azt a módszert alkalmaztuk az egymást követő stabil állapotok kódjának megválasztásakor, hogy minden állapot átmenetre biztosítottuk a kódok egyetlen szekunder változó értékében való változását, hiszen a kritikus versenyhelyzet forrása éppen a többszörös változás. 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.

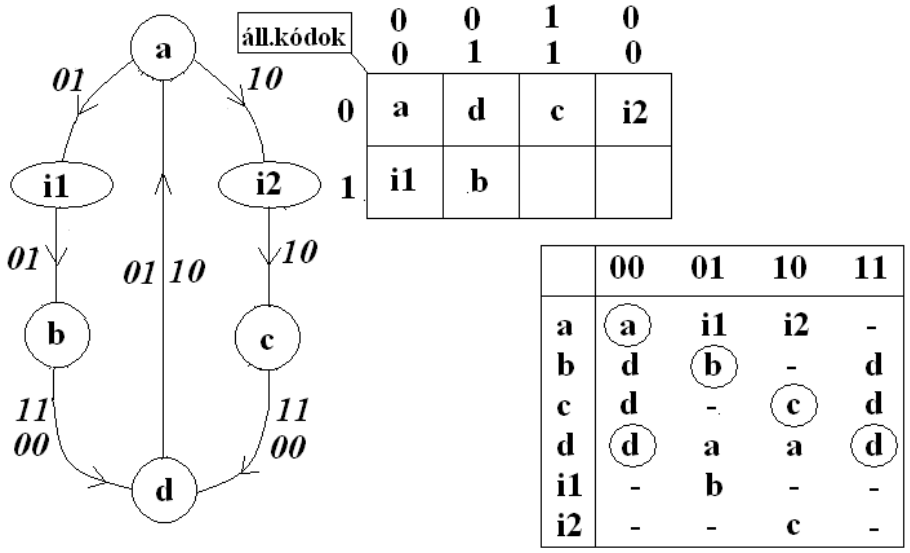
A fentiek szemléltetéseként nézzük a 6.2.1.a ábrán látható példát:



6.2.1.a ábra[1]

Egy aszinkron sorrendi hálózat előzetes szimbolikus állapottáblája és a hozzá tartozó állapotgráfja

Be kell szúrnunk két instabil állapotot (i_1 és i_2), 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. Lényeges tulajdonság, hogy sem az i_1, sem az i_2 állapot soha sem stabilizálódik, de áthidaló tranziens szerepet töltenek be. Ez jól követhető az új állapot átmeneti táblázat segítségével is (6.2.1.b ábra):



6.2.1.b ábra[1]

Bővítés szomszédos kódú instabil állapotokkal

6.2.2 A Tracey-Unger módszer elve (szisztematikus módszer)

Aszinkron sorrendi hálózatokban a szekunder változók között 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 megegyezzenek, és ebben a 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.

Példa:

adott egy kétbemenetű (DC) aszinkron sorrendi hálózat összevont szimbolikus állapottáblája(6.2.2.a ábra):

köv.áll / z

akt.áll \ D C	0 0	0 1	1 0	1 1
s1	(s1) / 0	(s1) / 0	s2 / 0	(s1) / 0
s2	s1 / 0	- / -	(s2) / 0	s3 / 1
s3	s4 / 1	(s3) / 1	(s3) / 1	(s3) / 1
s4	(s4) / 1	s1 / 0	s3 / 1	- / -

6.2.2.a ábra[1]

Egy kétbemenetű (DC) aszinkron sorrendi hálózat összevont szimbolikus állapottáblája

A kódolási eljárást az összevont szimbolikus állapottábla analízisével kezdjük: első lépésként 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. Az analízis eredményeként kapott listás táblázat a 6.2.2.b ábrán látható:

00->01	00->10	01->00	01->11	10->00	10->11	11->01	11->10
s4->s1 s3 *1*	s1->s2 s3 *2*	s3->s4 s1 *4*	-	s2->s1 s4 *5*	s2->s3 s1 *6*	-	~~s1->s2~~ ~~s3~~
	s4->s3 s2 *3*			~~s3->s4~~ ~~s1~~			

6.2.2.b ábra[1]

Az összes lehetséges stabil-stabil állapotátmenet listázása a „leselkedők" (hazárd kódos állapotok) feltüntetésével

Láthatjuk például, hogy a '**00**' → **01**' megengedett bemeneti váltáshoz egyetlen stabil-stabil állapot átmenet olvasható ki, nevezetesen az s_4 állapotból az s_1 állapotba való átmenet. Ezt egyedül az s_3 állapot tranziens kódja veszélyezteti, tehát itt a leselkedő állapot az s_3. Az ennek megfelelő kódolási szabályt a 6.2.2.c ábrán látható kódolási táblázatban az Y_1 szekunder változó képviseli, mégpedig azzal, hogy az oszlopában feltüntetett állapotkódban mind az s_1, mind az s_4 '0'-val jelenik meg, a leselkedő s_3 viszont '1'-el! Az s_2 állapot Y_1 pozícióbeli kódrésze közömbös. Így szerkesztjük végig a táblázatot, ismétlésekbe nem bocsátkozunk, és a fordított irányú, de azonos leselkedőt mutató átmeneteket is értelemszerűen csak egyszer tüntetjük fel (ld. az áthúzott táblázat elemek: 6.2.2.b ábra).

	Y1	*Y2*	*Y3*	*Y4*	*Y5*	*Y6*
	1 2	*1 2*	*1 2*	*1 2*	*1 2*	*1 2*
s1	*0 1*	*0 1*	*- -*	*1 0*	*0 1*	*1 0*
s2	*- -*	*0 1*	*1 0*	*- -*	*0 1*	*0 1*
s3	*1 0*	*1 0*	*0 1*	*0 1*	*- -*	*0 1*
s4	*0 1*	*- -*	*0 1*	*0 1*	*1 0*	*- -*

6.2.2.c ábra[1]

A stabil-stabil átmenetek és a hozzájuk tartozó leselkedők Tracey-Unger módszer szerinti kódolási lehetőségei

Következő lépésként vizsgáljuk meg az összes szekunder változóra a szabály alapján felírt lehetséges kódkiosztásokat. 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. 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. A 6.2.2.d ábrán látható táblázatban egy lehetséges állapot összevonással kialakult csoportosítást végeztünk:

	Y1		Y2		Y3		Y4		Y5		Y6	
	1	2	1	2	1	2	1	2	1	2	1	2
s1	0	1	0	1	-	-	1	0	0	1	1	0
s2	-	-	0	1	1	0	-	-	0	1	0	1
s3	1	0	1	0	0	1	0	1	-	-	0	1
s4	0	1	-	-	0	1	0	1	1	0	-	-

6.2.2.d ábra[1]

Egy lehetséges szekunder változó csoport összevonás a kódolási lista alapján

Az összevont állapotváltozók indexei (Y_p: „piros” csoport; Y_z: „zöld” csoport) mutatják, mely oszlopokat sikerült összevonni.

Ezzel kialakult, hogy a versenyhelyzet-mentes kód végül is két szekunder változóval biztosítható (6.2.2.e ábra):

	Y_p	Y_z
s1	0	0
s2	0	1
s3	1	1
s4	1	0

6.2.2.e ábra[1]

A mintafeladat állapotainak kritikus versenyhelyzet-mentes kódolása két szekunder változó felhasználásával

Az összevont szimbolikus állapottáblából származtatott kódolt állapottábla a 6.2.2.f ábrán látható:

		$Y_p Y_z / Z$			
Y_p^v	Y_z^v	0 0	0 1	1 0	1 1
0	0	(0 0)/0	(0 0)/0	0 1/0	(0 0)/0
0	1	0 0/0	- -/-	(0 1)/0	1 1/1
1	1	1 0/1	(1 1)/1	(1 1)/1	(1 1)/1
1	0	(1 0)/1	0 0/0	1 1/1	- -/-

6.2.2.f ábra[1]

A mintafeladat kódolt állapottáblája

A feladat megoldásának további lépéseit a már megismert módon kell elvégezni.

7. Összetett digitális egységek

Ebben a fejezetben az előzőekben már megismert egyszerű kapuszintű logikai hálózati elemek segítségével felépített összetett digitális egységek alapelemeit ismerhetjük meg. Ezek a magasabb szintű egységek – a digitális hálózatokban betöltött funkciójuk szerint – három alapcsoportban sorolhatók:

- *multiplexerek, demultiplexerek*
- *regiszterek*
- *funkciós egységek*

A multiplexerekkel már egy korábbi fejezetben megismerkedtünk: ezek a hálózati elemek a demultiplexerekkel együtt adatút szakaszok kijelölésére szolgálnak.

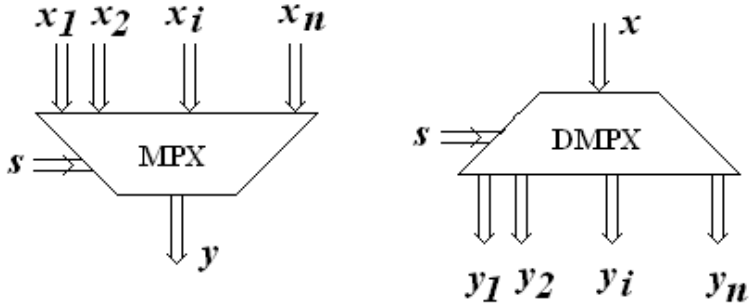
A regiszterek alapelemeivel, a tárolókkal szintén foglalkoztunk már: ezek adatokat tárolnak, és ezek elérését biztosítják.

A funkciós egységek alkalmazása egy digitális architektúrában adatok közötti műveletek elvégzését teszi lehetővé.

7.1 Multiplexerek, demultiplexerek

A multiplexerek és a demultiplexerek, mint adatút kijelölő egységek kerülnek felhasználásra a digitális hálózatokban. Felépítésük és funkciójuk alapján a kombinációs hálózatok csoportjába tartoznak, ezért már az első fejezetben pl. a multiplexerekkel részletesebben is foglalkoztunk, sőt a számláló bázisú szinkron sorrendi hálózatok tervezésekor, mint programozható logikai hálózati elemet is felhasználtuk egy speciális célarchitektúra építőelemeként.

Ismétlésként elevenítsük fel röviden a multiplexerek és demultiplexerek működésének és felépítésének lényegét: multiplexereknél(MPX) több forrás közül (x_1, x_2,.. x_n) kijelöljük azt, amelynek adata a kimenetre (y) kerül, míg a demultiplexerek(DMPX) esetén azt a kimenetet (y_1, y_2,.. y_n) címezzük meg, amelyen az egyetlen forrás (x) adatát meg kívánjuk jeleníteni. Az adatutak kijelölése vezérlőbemenetek (s) segítségével, címzéssel történik (7.1.a ábra):



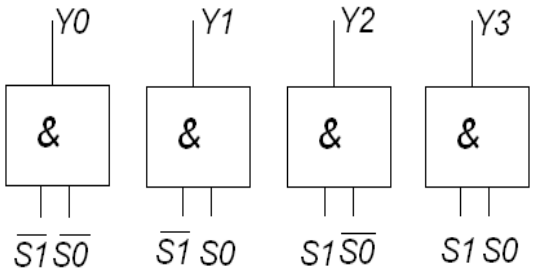
7.1.a ábra[1]

A multiplexerek és demultiplexerek szokásos szimbólum jelölése

Korábbi tanulmányaink során megismerhettük, hogyan lehet a multiplexert programozható logikai hálózati elemként alkalmazni: a mintermes kanonikus alakban megadott függvény mintermjeit a címző (vezérlő) bemenetekre adott címek képviselik, és a megcímzett adatbemenetre rá kell kapcsolnunk az adott mintermhez tartozó logikai értéket. Ezeknek a logikai konstansoknak a bemenetekre való kapcsolását tekinthetjük a multiplexerek programozásának.

A demultiplexereket gyakran használják dekóderként: ha egy demultiplexer adatbemenetét állandó, logikai '1' szintre kapcsoljuk, akkor ez egyenértékű azzal, hogy a hálózatában szereplő '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

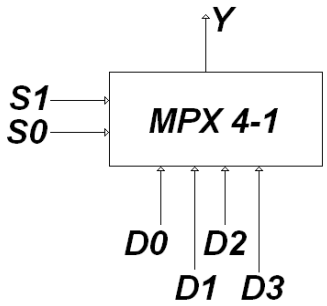
ilyen áramköröket nevezzük dekódereknek. Egy DMPX1-4 egység dekóderként történő felhasználására mutat egy realizációs példát a 7.1.b ábra:



7.1.b ábra[1]

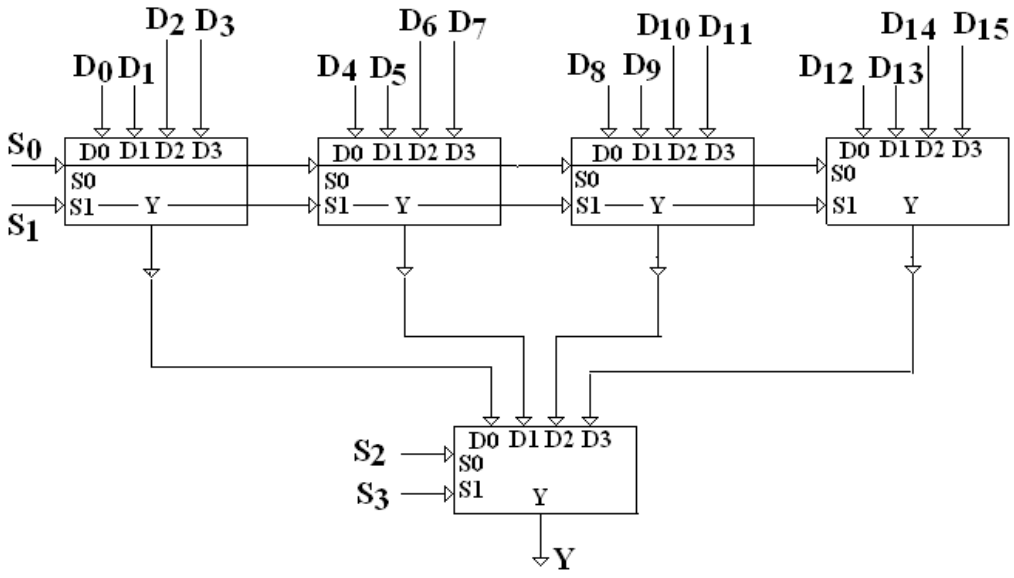
DMPX1-4 egységből kialakított kétbemenetű dekóder

Közepes integráltságú (MSI = Medium Scale Integration) áramkörökben gyakran használatos adatút választó eszköz az MPX4-1 multiplexer, mint alapegység (7.1.c ábra). Előfordul azonban, hogy esetenként a bemeneti adatunk négynél több bitből áll. Ebben az esetben célszerű a nagyszámban rendelkezésre álló alapegység felhasználásával elvégezni a bővítést. Erre mutat példát a 7.1.d ábra: itt MPX16-1 multiplexert alkottunk MPX4-1 egységekből. Látható, hogy nemcsak vízszintes irányban kellett bővíteni az egységet, hanem mélységében is: a felső sort „felfűző" címzőbemeneteken(s_0; s_1) megjelenő címvektor rész mind a négy egység bemenetei közül($D_0 - D_{15}$) ugyanazt a sorszámú bemenetet címzi meg, és a második sor egyetlen egységének címvektora(s_2; s_3) ebből a négy adatból választ ki egyet.



7.1.c ábra[1]

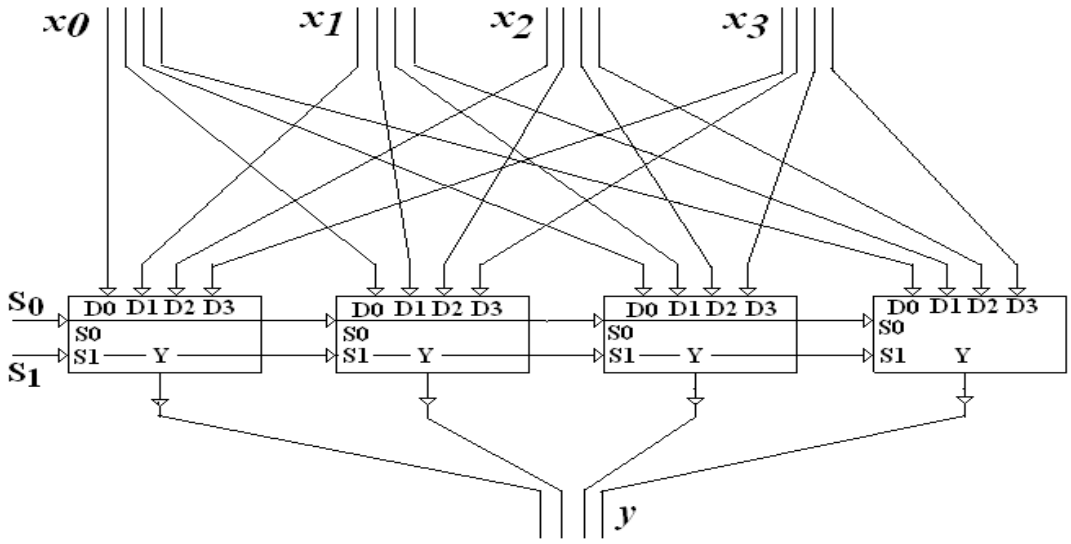
MPX4-1 egység



7.1.d ábra[1]

Bővítés a bemenetek számának növelésére MPX4-1 egységek felhasználásával

Amennyiben az a feladat, hogy 4 különböző adatbemenet($x_0 \dots x_3$) közül egynek az értékét kell egységesen továbbítani, akkor a sínek között választás céljából kialakított bővítésre mutat példát a 7.1.e ábra:



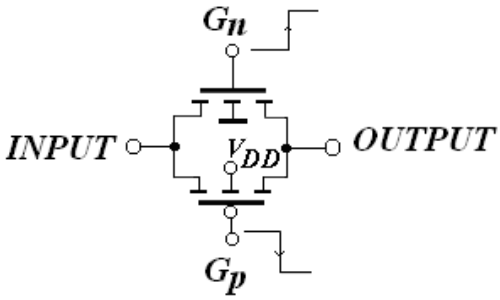
7.1.e ábra[1]

Négy, 4-bites sínből egyet kiválasztó multiplexer egység MPX4-1 alapegységekből

Az átvivő kapu (transmission gate) alkalmazása: a „lebegő szint" megvalósítása digitális hálózatokban

Eddigi tanulmányaink során egy hálózati struktúra adott pontján mindig kétféle logikai szintet határoztunk meg, illetve értelmeztünk: a magas ('1') és alacsony ('0') szinteket. Egy digitális rendszerben azonban sokszor szükség van az egyes hálózati részekben (elsősorban adatút átviteli pontokon) egy harmadik, ún. „lebegő" szint biztosítására. 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.

Ezt a megvalósítást biztosítja a CMOS átvivő kapu (transmission-gate, transfer-gate), amely két MOSFET eszközből álló kapcsoló (7.1.f ábra.):



7.1.f ábra[1]

Az átvivőkapu (transmission gate)

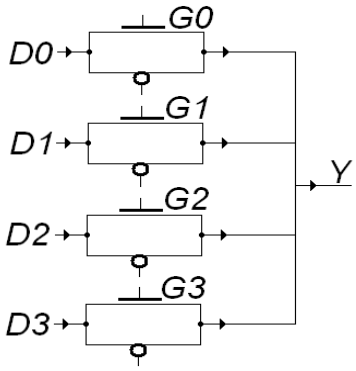
A „transmission gate" (TG) egy *n*-MOSFET és egy *p*-MOSFET párhuzamos összekapcsolása. Ahhoz, hogy bekapcsoljuk, az *n*-MOSFET G_n elektródájára magas, a *p*-MOSFET G_p elektródájára alacsony szintet kell adni.

A tranzisztorok párban történő alkalmazásának a szükségességét a következők indokolják:

a logikai áramkörökben szokásos, úgynevezett 'egy tápfeszültséges' rendszerben a logikai alacsony szinthez a '0', vagy 'föld' (GND) potenciált, a logikai magas szinthez ('1')

pedig a pozitív tápfeszültség értéket (V_{dd}) rendeljük. Megvizsgálhatjuk, hogy az n- illetve p-csatornás MOSFET hogyan viszik át egyik elektródájukról a másikra az egyes logikai szinteket. Azt fogjuk tapasztalni, hogy az n-MOSFET a logikai alacsony szintet jól átviszi, ezzel szemben a magas szintet csak „küszöbfeszültségnyi" hibával képes egyik oldaláról a másikra átjuttatni. (Küszöbfeszültségnek azt a feszültség értéket nevezzük, amelynek elérésénél(G) a tranzisztor „vezető" állapotba kerül, azaz „kinyit".) A p-csatornás MOSFET pedig éppen fordítva viselkedik, a magas szinttel boldogul hiba nélkül. Beláthatjuk, hogy a TG mind a magas, mind az alacsony szintet hibátlanul viszi át a bemenetről a kimenetre, hiszen egyik eszköz mindig kisegíti a másikat, és befejezi a folyamatot.

Egy átvivő kapukkal felépített MPX4-1 egységet láthatunk a 7.1.g ábrán:



7.1.g ábra[1]

MPX4-1 egység CMOS átvivő kapukkal

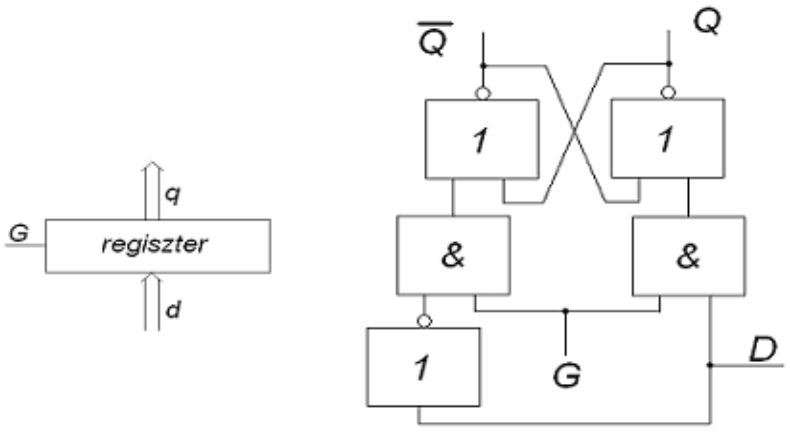
A vezérlőelektródákat bekapcsoláskor ellentétesen kell vezérelni, azaz a kis körökkel jelölt vezérlő bemenetek a G_0 - G_3 vezérlők negáltjai. Az átvivő kapukból felépített multiplexer jellegzetessége, hogy kimenete képes a logikai harmadik állapot felvételére, azaz ha egyik kapcsolót sem nyitjuk, a kimenet „lebeg".

7.2 Regiszterek

A digitális hálózatokban a regiszterek adatokat tárolnak, és ezek elérését biztosítják. A különböző típusokat felépítésük és működésük alapján külön alfejezetekben ismertetjük.

7.2.1 Szintvezérelt statikus regiszter

A szintvezérelt statikus regiszter általános sémája, és DG tárolóból kialakított egybites cellájának kapuszintű realizációja 7.2.1.a ábrán látható:



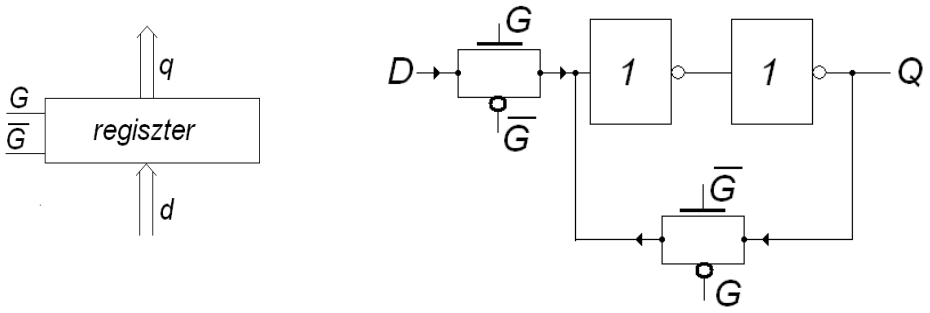
7.2.1.a ábra[1]

A szintvezérelt statikus regiszter általános szimbóluma, és DG tárolóból kialakított egybites cellájának kapuszintű realizációja

Erre a regiszterre egy bemeneti bitvektor (*d*) és egy logikai beírójel (*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.

7.2.2 Szintvezérelt statikus regiszter ponált és negált beíró jelekkel

A ponált és negált beíró jellel vezérelt CMOS regiszter szimbóluma és egybites cellájának kapuszintű struktúrája a 7.2.2.a ábrán látható:



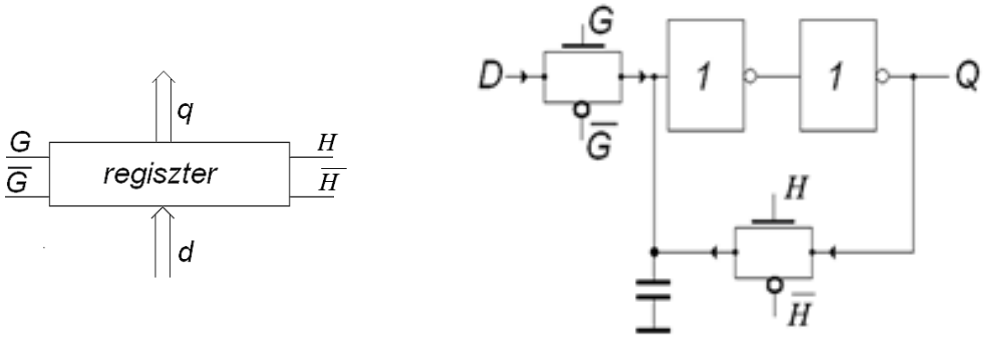
7.2.2.b ábra[1]

Ponált és negált beíró jellel vezérelt CMOS regiszter szimbóluma és egybites cellájának kapuszintű struktúrája

A CMOS technikában, különösen a VLSI (Very Large Scale Integration) 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.

7.2.3 Kvázistatikus regiszter

A kvázistatikus regiszter szimbólumát és egy-bitnyi NOR struktúráját a 7.2.3.a ábrán láthatjuk:



7.2.3.a ábra[1]

A kvázistatikus regiszter szimbóluma és egy bites cellájának struktúrája

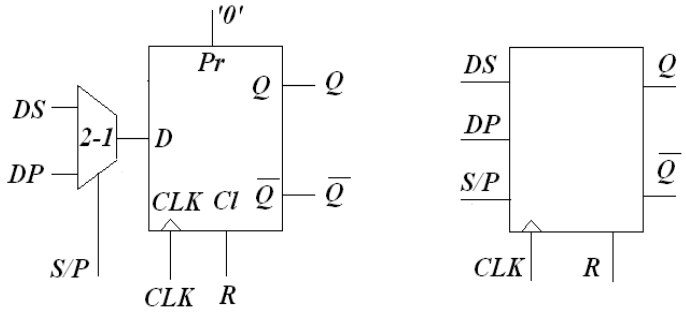
A kvázistatikus regiszter alapcellája (1-bites egysége) a CMOS inverter bemeneti kapacitásának átmeneti töltéstároló képességét használja ki. Itt a *G* beírójel két felfutása, azaz két beírás között egy tartó (*H, HOLD*) impulzus rendszeres jelentkezése szükséges. A *H* impulzusok között a bemeneti kapacitás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

7.2.4 Élvezérelt regiszter

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

7.2.5 Soros elérésű tárolók

A soros elérésű memóriák alapeleme az élvezérlésű, ún. kétfázisú D-MS tároló. Ebből a tárolóból egy MPX2-1 multiplexer alkalmazásával olyan egységet kapunk, amelynek két adatbemenete közül (*DS, DP*) közül az *S/P* vezérlőjel szintjének egyike választ. A tároló *PRESET(Pr)* bemeneteit konstans logikai alacsony szintre kötjük, a *CLEAR(Cl)* bemeneteket ezzel szemben a kezdeti ’0’ állapot beállítására használni fogjuk. A 7.2.5.a ábrán látható egységet soros memóriák építőelemeként fogjuk felhasználni.

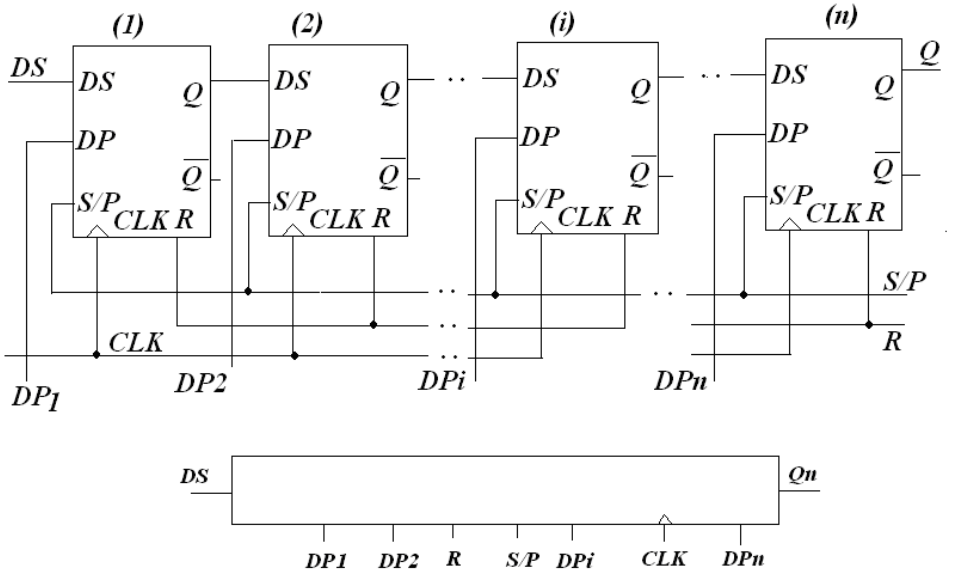


7.2.5.a ábra[1]

Az egybites soros memóriaelem MPX2-1 multiplexerrel(bal), és a sematikus szimbóluma(jobb)

7.2.5.1 Párhuzamosan is betölthető soros memóriák

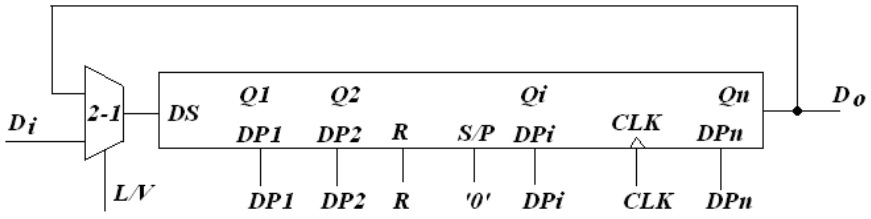
A 7.2.5.1.a ábrán egy nyitott, soros elérésű *n*-bites memóriablokk sémája látható. Az *S/P* vezérlőbemenet állapotától függően az órajelciklusra vagy balról-jobbra léptetés, vagy párhuzamos betöltés történik. Az egység az *R* jellel alapállapotba hozható.



7.2.5.1.a ábra[1]

Nyitott, soros elérésű 'n'-bites regiszter(felső kép), és általános sémája(alsó kép)

A 7.2.5.1.b 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* vezérlőbemenet segítségével. Ezt a megoldást alkalmazzák pl. az ún. gyűrűs számlálók kialakításánál. Egy *n* modulusú gyűrűs számlálónál az eredeti információ az *n*-dik lépés (ciklus) után kerül vissza a regiszter megfelelő helyiértékeire. A regisztereknek ezt a csoportját szokták felhasználni pl. soros működésű aritmetikai egység átmeneti tárolójaként, ha az egyik tényezőt - műveletvégzés után - változatlanul kell megtartani.

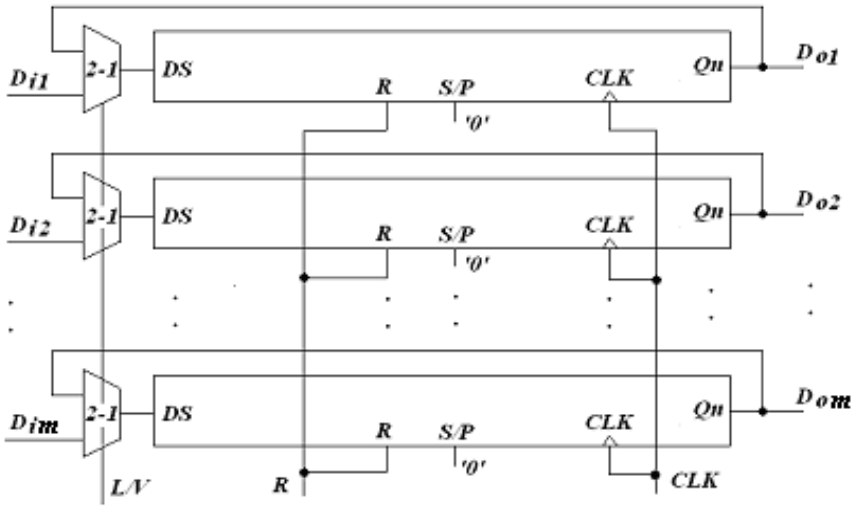


7.2.5.1.b ábra[1]

Párhuzamosan betölthető, sorosan rátölthető soros memória.

7.2.5.2 Szószervezésű soros memóriák

A megismert építőelemből 1-nél nagyobb, például *m* szószélességű soros memóriát építünk: elhagyjuk a párhuzamos beírás lehetőségét, azaz az *m* számú gyűrű *S/P* bemeneteit soros üzemmódra állítjuk be. A memória *L/V* vezérlővezetékével beállíthatjuk, hogy a memóriában lévő adatokat forgatjuk, vagy új adatot szúrunk be a régiek közé (7.2.5.2.a ábra):

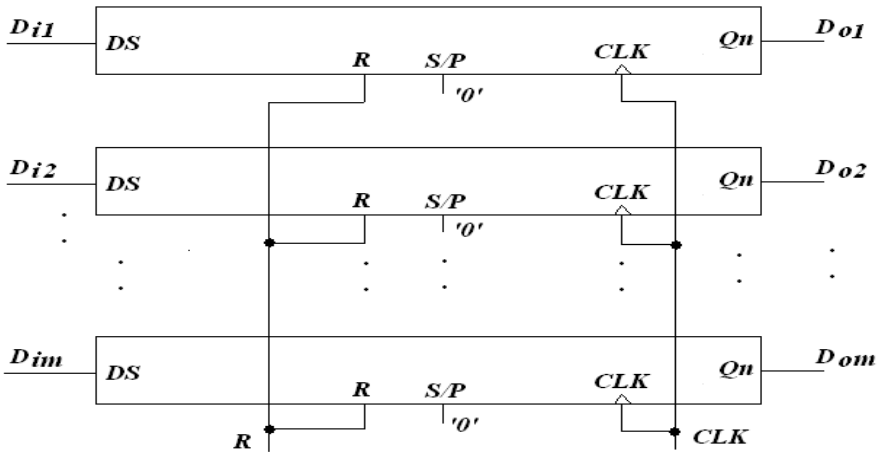


7.2.5.2.a ábra[1]

Egy m-bit szószélességű soros memória-egység

7.2.5.3 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 7.2.5.3.a ábra is mutatja:

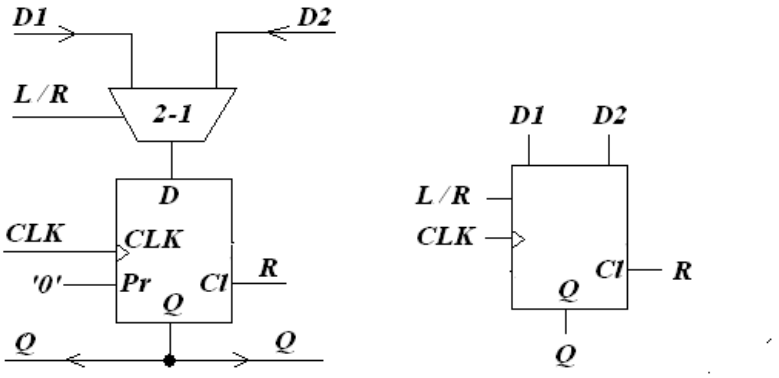


7.2.5.3.a ábra[1]

Egy m-bit szószélességű FIFO memória

7.2.5.4 LIFO memóriák

A LIFO memóriák alapelemeként szolgáló, MPX2-1 multiplexerrel kiegészített D-MS flip-flop sémája és szimbóluma a 7.2.5.4.a ábrán látható:

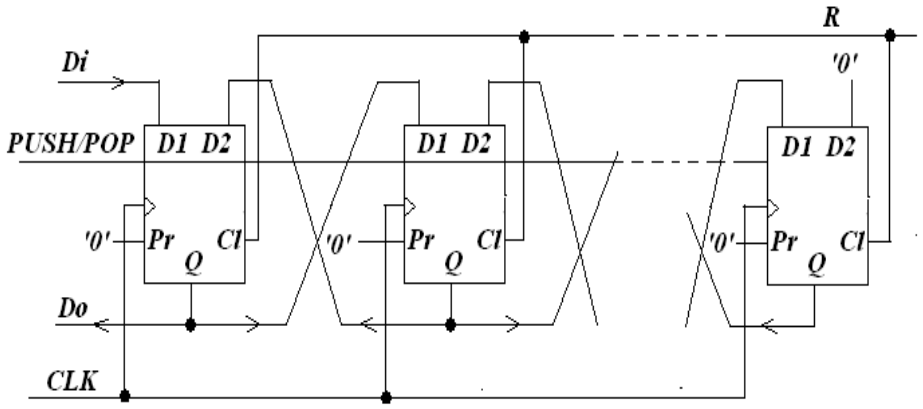


7.2.5.4.a ábra[1]

LIFO memória alapelem

A D-MS flip-flop kimenetén az *L/R* vezérlőjeltől függően vagy a baloldali D_1, vagy a jobboldali D_2 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 vezérlőbemenete van: betöltésre a 'betölt' (PUSH), kiolvasásra a 'kiugrat' (POP) vezérlőbemeneteket használjuk, természetesen egymás kizárásával. A LIFO egyetlen adatcsatlakozása tehát bemenet és kimenet szerepét is betölti.

A LIFO elemekből alkotott 1-bit szélességű LIFO memória a 7.2.5.4.b. ábrán látható. Ebből könnyen alkothatunk *m* szószélességű LIFO memóriát.

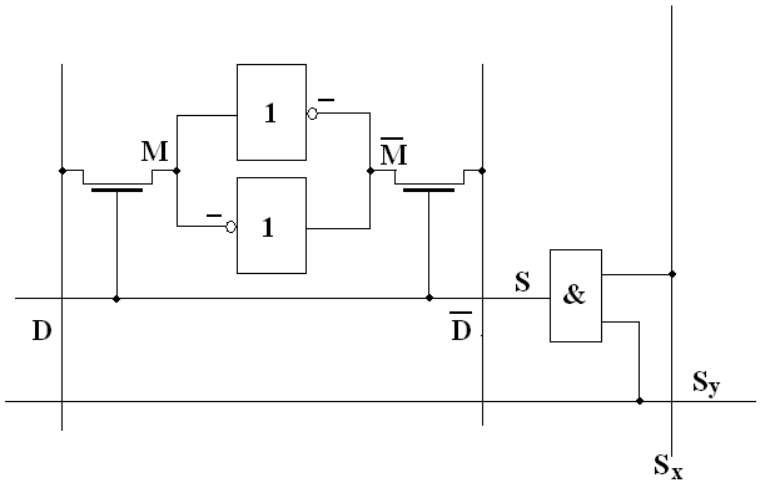


7.2.5.4.b ábra[1]

LIFO memória sor sémája

7.2.6 Párhuzamos hozzáférésű memóriák

Egy memóriablokk esetében a párhuzamos hozzáférés azt jelenti, hogy a memória minden egyes bitjéhez, vagy minden egyes szavához annak helyétől független elérési idővel férünk hozzá akár átírás(WRITE), akár kiolvasás(READ) 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. Egy ilyen, 1-bites alapcellát láthatunk a 7.2.6.a ábrán:



7.2.6.a ábra[1]

1-bites RAM alapcella felépítése

A cella működését a következőképpen értelmezhetjük:

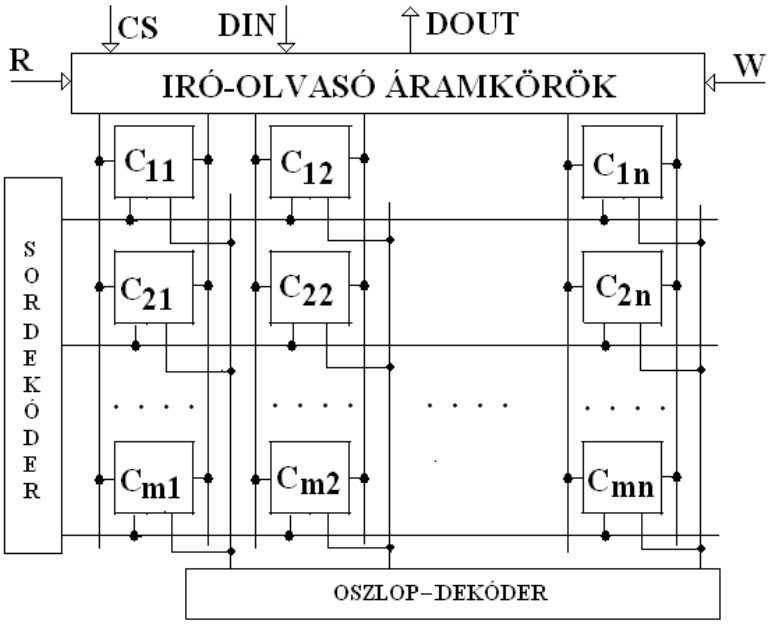
a realizációs rajzon az egymást ölelő inverterek bistabil cellát alkotnak. Ha a két elektronvezetésű MOSFET kapcsoló zárva van, azaz az *S* bemenet alacsony szintű, akkor az inveterek őrzik az utoljára beírt állapotot (az *S* vezérlőjelet egy S_x oszlopdekóder-bit és egy S_y sordekóder-bit kiválasztó jel 'ÉS' kapcsolatával állítjuk elő). Ez a <u>tárolás</u> állapota.

Ha *S* magas szintű lesz, akkor két eset van:

1. <u>olvasás(READ)</u>: ha a *D* és a $\overline{D}$ vonalakat kívülről lebegtetjük, akkor az *S* felemelkedésekor a *D* vonalon az *M*, a $\overline{D}$ vonalon az $\overline{M}$ jelenik meg. Ez tekinthető a tárolt adat kiolvasásának.

2. <u>írás(WRITE):</u> ha a *D* és a $\overline{D}$ 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 a $\overline{D}$ értékeket az *M* és az $\overline{M}$ pontra kényszerítő MOSFET kapcsolók. Ha *D*=1 és *M*=0, illetve a $\overline{D}$=0 és $\overline{M}$=1, akkor a memóriacella átbillen a másik, az előzővel ellentétes állapotba. Ez az írás folyamata. (Az erősebb illetve gyengébb logikai meghajtóképességet az invetereket és a kapcsolókat alkotó MOSFET eszközök megfelelő méretezésével lehet elérni.)

A 7.2.6.b ábrán egy *m*x*n* RAM alapcellából álló bit-szervezésű RAM blokk látható:



7.2.6.b ábra[1]

Bit-szervezésű, m_xn bites RAM blokk

A kapacitásától függetlenül csak egy adatbemenete(*DIN*) és egy adatkimenete(*DOUT*) van, címzése pedig sor- és oszlopdekóderekkel történik. A blokk tartalmaz még egy kapcsolódó *'író-olvasó áramkörök'* egységet is, melyen keresztül zajlik a cellákkal történő kommunikáció. Ennek lényege, hogy a megcímzett bit cellájának az állapota olvasáskor (*R*) a „DOUT" bit-kimenetre kerül, míg íráskor (*W*) a „DIN" bitvektorra kapcsolt érték a memóriának a kijelölt cellájába kerül. Mindkét művelet végrehajtásának a feltétele az is, hogy a használni kívánt blokk kijelölésre kerüljön, azaz a *CS* (Chip Select) vezérlő bemenet magas szintű legyen.

7.2.7 Állapotregiszterek, számlálók

A számlálókat - egy korábbi fejezetben(4.6) már bemutatottak szerint - főként időzítő-vezérlő áramkörökben rögzített funkciójú időzítő egységként szokták alkalmazni, így kapcsolódnak az adatfolyamatokat vezérlő egységek regisztereihez.

7.2.7.1 Szinkron számlálók

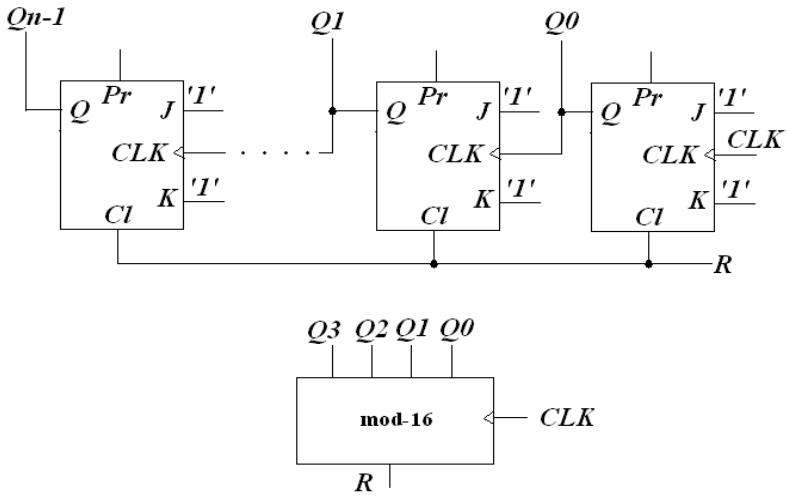
Korábbi tanulmányainkból (4.6 fejezet) már tudjuk, hogy a 'Pr' és 'Cl" segédbemenetekkel rendelkező, élvezérelt JK -MS tároló (egy 'E' =enable engedélyező

bemenettel kiegészítve) képezi a szinkron számlálók alapegységét. Ezeknek a számlálóknak a sajátosságaival, a különböző kialakítási és felhasználási lehetőségeivel – konkrét példákon és feladatmegoldásokon keresztül – már megismerkedtünk és foglalkoztunk, így ebben a fejezetben történő tárgyalásától – a továbbiakban – eltekintünk.

7.2.7.2 Aszinkron számlálók

Az aszinkron számlálók alapját –a szinkron számlálókhoz hasonlóan – a JK-MS tárolók alkotják, és ebben az esetben az ún. „kettes-osztó" funkciót használjuk ki az aszinkron számláncok létrehozásánál. Ismétlésként rögzítsük: a „kettes-osztó" funkció úgy valósul meg, hogy amennyiben a 'mester' tároló beírása az órajel felfutó, a 'szolga' beírása a lefutó élre történik, az órajel frekvenciája megfeleződik, azaz az így kapcsolt JK flip-flop az órajel frekvenciát 2-vel osztja (4.6.1.1 fejezet). Aszinkron számláncokban az órajel logikai szerepet kap, ezért olyan tárolót kell választani, amelyben a 'Cl' bemenet közvetlenül és azonnal hat a kimenetre (az órajel közreműködése nélkül), azaz aszinkron módon működik.

Aszinkron *mod-m* (m = 2n) számlálót kapunk, ha *n* számú kettes osztót a 7.2.7.2. ábrán látható módon kapcsolunk össze:



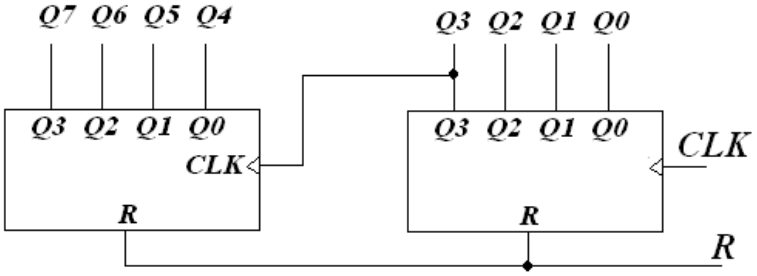
7.2.7.2.a. ábra[1]

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

A számlálási (inkrementálási) sorrend ($Q_{n-1}...Q_0$) *R* (RESET)vezérlés után:

$Q_{n-1}\ldots Q_1 Q_0$	$eredmény$
000…00	’0’
000…01	’1’
000…10	’2’
… … …	..
100…00	’2^{n-1}’
… … …	..
111…11	’2^n-1’ (=m-1)
000…00	’0’

Adott modulusú számlálókból (hasonlóan a szinkron számlálókhoz) kaszkádosítással nagyobb modulusú számlálót is ki lehet alakítani. Erre mutat példát a 7.2.7.2.b ábra. Itt mod-16-os aszinkron számlálókból állítottunk elő mod-256-os számláló egységet.



7.2.7.2.b ábra[1]

Mod-256 számláló 4-bites aszinkron számláncok kaszkádosításával

7.3 Funkciós egységek

A funkciós egységek egy vagy két, binárisan ábrázolt adattal valamilyen műveletet végeznek el. Működésük megértéséhez elsőként néhány fontos alapelvet rögzítenünk kell.

A funkciós egységek operandusai és eredményei bináris kódban ábrázolt számok. A kettő hatványaival súlyozott bináris kódolás elvét, az egész számok bináris alakjának felírását, a bináris számok decimálisakká való alakításának szabályait matematikai előtanulmányainkból már ismerjük. Szükséges ismeret még a negatív számok (egészek és törtek) ábrázolásának az a módja is, amely könnyűvé teszi az előjeles bináris számokkal való aritmetikai műveleteket. Ezért ebben a bevezető részben megismerjük a komplemens kódok fogalmát és a velük való számolás fő szabályait.

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

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

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 helyiértékű 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.

A fentiekben felvázolt matematikai alapok áttekintése után ismerjük meg részletesen a digitális hálózatokban leggyakrabban alkalmazott funkciós egységek felépítését és működését.

7.3.1 Komparátorok

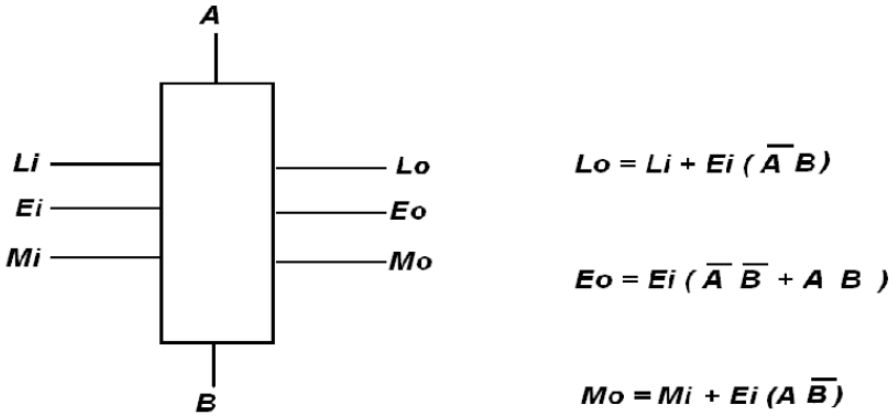
A komparátorok binárisan ábrázolt adatok összehasonlítását végzik el. (A következőkben csak pozitív bináris számok összehasonlításával foglalkozunk, de megjegyezzük, hogy léteznek a komplemens kódra kiterjesztett komparátor változatok is.)

Az összehasonlításnak általában háromféle eredménye lehet:

- az egyik adat nagyobb a másiknál (***Mo***),
- azonos értékű a másikkal (***Eo***),
- kisebb a másiknál (***Lo***).

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

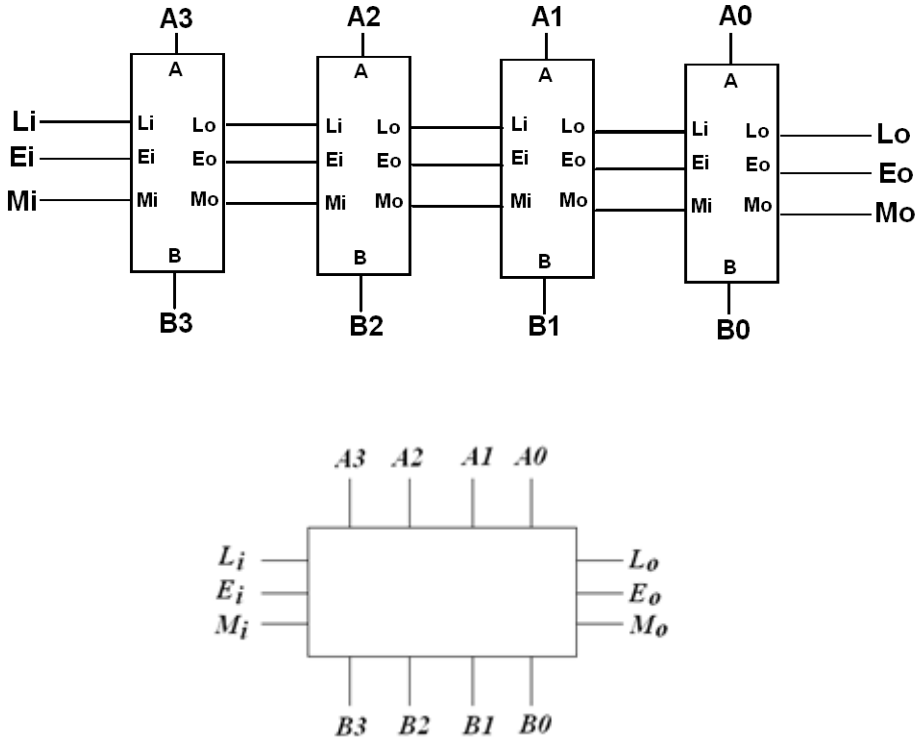
A fenti „univerzális egység" szükségességének elvét követve egészítsük ki az 1-bites alapegységet három bemenettel (***Mi, Ei, Li***), melyekkel biztosítjuk, hogy a már feljebb (MSB→) meghozott ***Mo*** = 1 vagy ***Lo*** = 1 döntés egyenesen kijusson a megfelelő kimenetre. Ezzel egy tetszőlegesen bővíthető (kaszkádosítható) komparátor egységet kapunk, melynek szimbóluma a 7.3.1.a ábrán látható:



7.3.1.a ábra[1]

Egybites univerzális komparátor sematikus ábrája

Egy 1-bites egységekből felépített 4-bites komparátor összeállítását és sematikus vázlatát láthatjuk a 7.3.1.b ábrán:

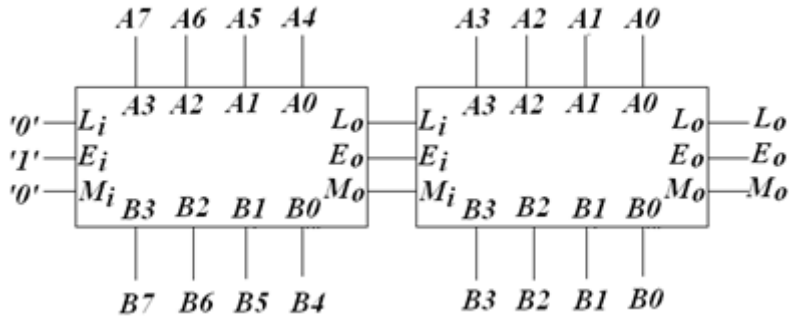


7.3.1.b ábra[1]

4-bites komparátor összeállítása 1-bites egységekből(fent), és sematikus vázlata(lent)

A 7.3.1.c. ábra mutatja, hogy pl. hogyan kapcsoljunk össze két 4-bites egységet, hogy egy 8-bitest kapjunk.

Megjegyzés: az MSB „felől", azaz a baloldalról „érkező" adatértékek helyére – a helyes működés érdekében - az egység ***Mi, Ei, Li*** *bemeneteire a megfelelő logikai konstans értékeket kell kapcsolni.*



7.3.1.c ábra[1]

8-bites komparátor kialakítása két 4-bites egység kaszkádosításából

7.3.2 Összeadók

A digitális hálózatok aritmetikai logikai egységeinek leggyakrabban használt alapeleme az 1-bites összeadó. Az egység működésének leírása az igazság táblázata alapján egyszerűen definiálható: adjuk meg a táblában a bináris összeadás műveletének eredményét egy adott helyiértéken, ahová áthozhatunk értéket a megelőző, kisebb helyiértékről, ($\boldsymbol{C_i}$), hozzáadjuk az adott helyiértéken megjelenő operandus bitek ($\boldsymbol{A}$, $\boldsymbol{B}$) összegét ($\boldsymbol{S}$), és képezzük a nagyobb helyiértékre való átvitel értékét is ($\boldsymbol{C_o}$)(7.3.2.a ábra). Ezzel egy olyan univerzális egységet alkottunk, melyet teljes összeadónak hívunk, és ennek segítségével tetszőleges bitszámú operandusokon elvégezhető a művelet (7.3.2.b ábra).

A	B	C_i	S	C_o
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

7.3.2.a ábra

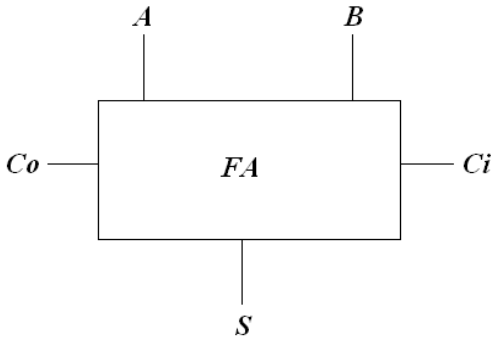
A teljes összeadó igazságtáblája

Az eredmény (**S**) és az átvitel (**C_o**) függvénye az összevonások alapján a következő algebrai alakban adható meg:

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

$$C_o = A\,B + B\,C_i + A\,C_i \quad = \overline{\overline{(AB)}\ \overline{(B\,C_i)}\ \overline{(A\,C_i)}}$$

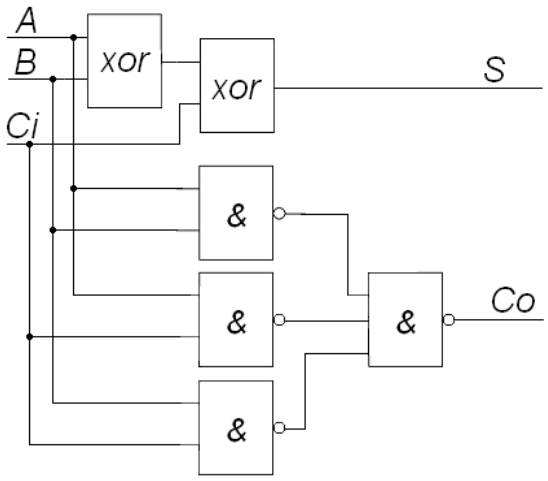
A teljes összeadó szimbóluma a 7.3.2.b ábrán látható:



7.3.2.b ábra[1]

A teljes 1-bites összeadó szimbóluma

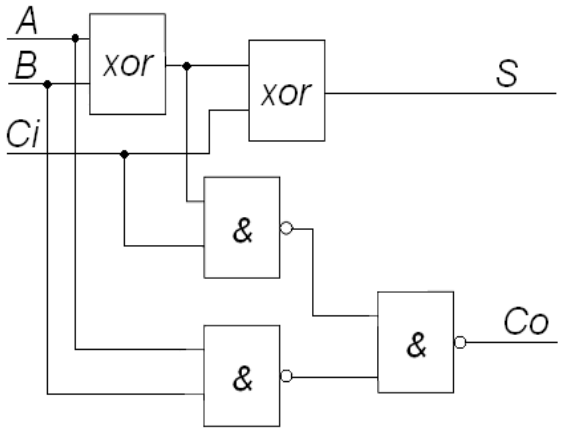
A táblából megadott függvényalak szerinti realizációt a 7.3.2.c ábrán áthatjuk:



7.3.2.c ábra[1]

A teljes összeadó egy lehetséges NAND-NAND realizációja (1)

Ez az áramköri megoldás viszonylag gyors műveleti eredményt szolgáltat, de több kaput tartalmaz, mint a 7.3.2.d ábrán látható realizáció:



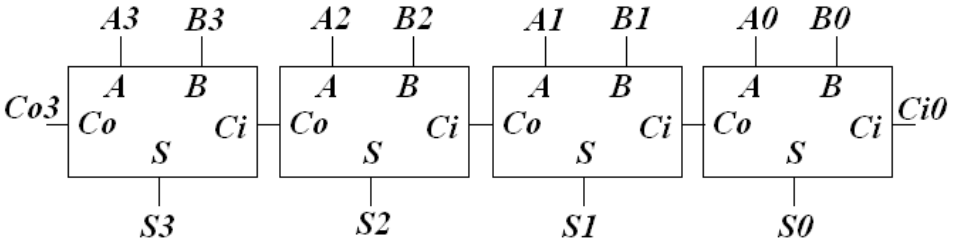
7.3.2.d ábra[1]

A teljes összeadó egy lehetséges NAND-NAND realizációja (2)

Ez a második (*2*) megoldás kevesebb kaput tartalmaz, mint az első (*1*), de lassabb a teljes eredmény (***S***, $\boldsymbol{C_o}$) megjelenítése szempontjából, hiszen az átvitel ($\boldsymbol{C_o}$) három kapun keresztül alakul ki, mert az átalakított függvényalakból ez következik:

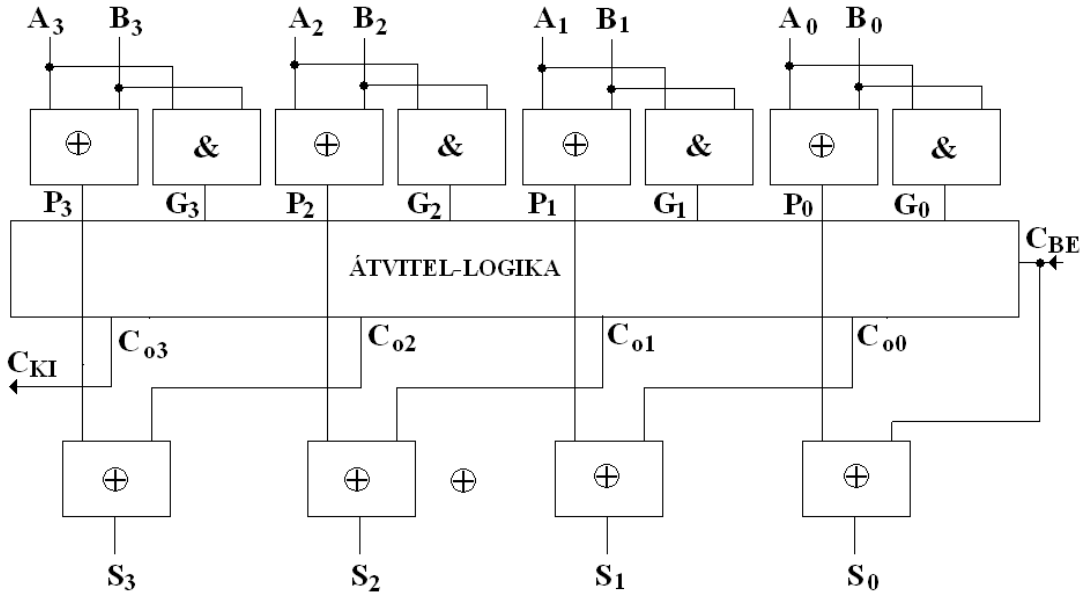
$$C_o = (A \oplus B)\ C_i + A\,B \ = \ \overline{\overline{(A \oplus B)\ C_i}\ \overline{(AB)}}$$

A teljes összeadók kaszkádosításával soros átvitelképzésű bit-vektor összeadót kapunk. (7.3.2.e ábra). A soros átvitelképzés óriási hátránya, hogy a legnagyobb helyiértéken 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 átvitelképzésű, illetve a vegyes, párhuzamos-soros átvitel-képzésű összeadók.



7.3.2.e ábra[1]

Soros átvitelképzésű 4-bites összeadó egység, 1-bites alapegységek kaszkádosításával



7.3.2.f ábra[1]

Egy 4-bites, párhuzamos átvitelképzésű összeadó felépítése

A párhuzamos átvitelképzésű összeadó esetében minden helyiértéken - így a *k.*-ik helyiértéken 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 kapjuk azokat a logikai

kifejezéseket, amelyek alapján összeállítható a 7.3.2.f ábrán bemutatott 4 bites párhuzamos átvitelképzésű összeadó egység.

7.3.3 Kivonók

Az alfejezet elején, a matematikai alapok áttekintése során már említésre került, hogy 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*.

Emlékeztetőül néhány fontos tulajdonsága a kettes-komplemens kódú számábrázolásnak a következő:

legyen

A : a szám, súlyozott bináris kóddal

KK(A) : a szám kettes komplemense, adott szabály szerint előállítva,

egy kettes komplemens kódú szám (-1)-szerese pedig a szám kettes komplemense, vagyis

A + **KK**(A) = 0!

A kettes komplemens kód összetétele:

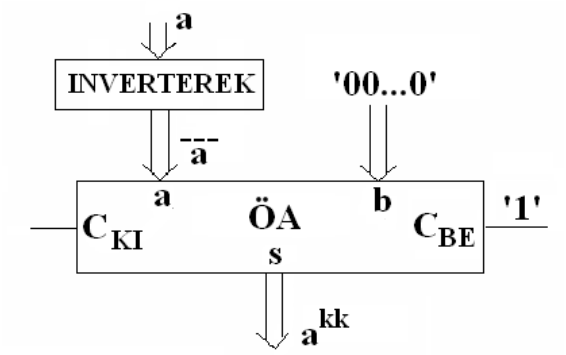
- *MSB* : előjel
- *(MSB-1) – LSB* : a számérték.

Ennek megfelelően

- ha a szám pozitív, akkor előjele ’0’, a számérték pedig a szám binárisan súlyozott abszolút értéke

- ha a szám negatív, akkor előjele ’1’, és az abszolút érték a kettes komplemens előállításával határozható meg

A 7.3.3.a ábra szerinti egység az *a* kettes-komplemens kódban ábrázolt szám kettes-komplemensét állítja elő: azaz, minden kivonandót rajta átengedve, összeadással hajthatjuk végre a kivonást:

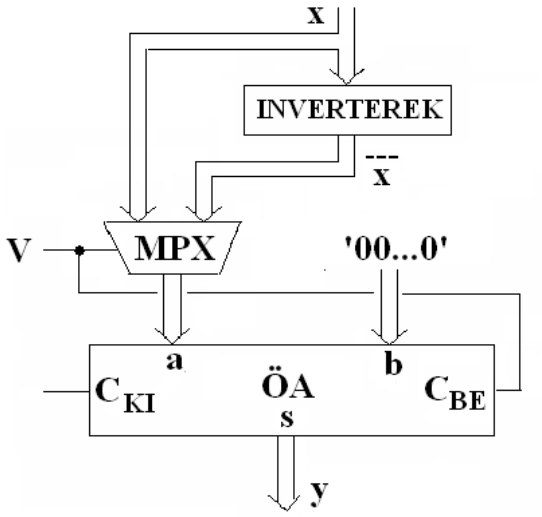


7.3.3.a ábra[1]

A kettes-komplemens képző egység sematikus ábrája, teljes összeadóból kialakítva

Ha ezt az egységet az összeadó *a* adatbemenetén a 7.3.3.b ábra szerinti kialakításban egy MPX2-1 multiplexerrel kiegészítjük, akkor egy ún. *vezérelhető kettes-komplemens képző egységet* kapunk. A működés lényege, hogy az egységünk

- ***V*** = 1 esetén az ***x*** kettes komplemensét állítja elő
- ***V*** = 0 esetén ***y*** = ***x***.

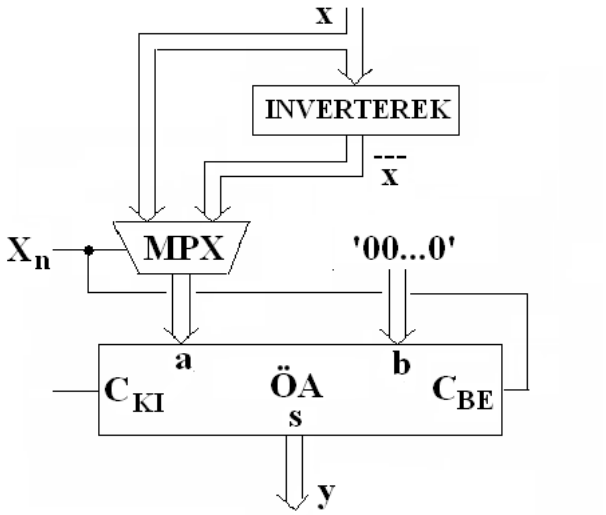


7.3.3.b ábra[1]

Vezérelhető kettes-komplemens képző egység.

Ezzel a kiegészítéssel lényegében egy *abszolútérték képző egységet hoztunk létre* (7.3.3.c ábra), melynek használatával azt a funkciót biztosítjuk, hogy bármely kettes komplemens kódban érkező szám abszolút értéke kerül az egység kimenetére. Az ábrán X_n az MSB, egyben a szám előjele:

- ha értéke '1', akkor a szám kettes komplemense lesz az abszolút érték,
- ha értéke '0', akkor ***y*** = ***x.***



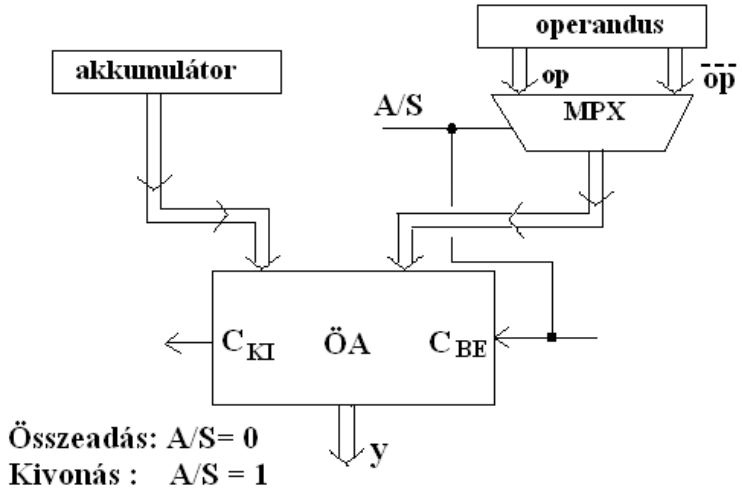
7.3.3.c ábra[1]

Abszolútérték képző egység

A 7.3.3.c ábrán látható egység képezi alapját a mikroprocesszorokban alkalmazott univerzális összeadó-kivonó egységnek(7.3.3.d ábra). A működés értelmezése a következő:

- 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.



7.3.3.d ábra[1]

Mikroprocesszorokban alkalmazott univerzális „összeadó-kivonó" aritmetikai egység

7.3.4 Szorzók és osztók

A szorzás és osztás műveletének elvégzésére a digitális rendszerekben – a felhasznált alapelvek és kialakítások alapján - sokféle és különböző megoldási lehetőségek adódnak az alkalmazható funkcionális egységeket tekintve. Ezek lehetnek pl. fix- vagy lebegőpontos műveletvégzők, ezen belül soros vagy párhozamos átvitel logikával működő egységek stb.

A teljesség igénye nélkül néhány példa ezek közül:

- szorzók:
  - előjeles vagy előjel nélküli szekvenciális szorzók
  - tömbszorzók (array-szorzók)
  - fa-szorzók (Booth-Wallace szorzók)

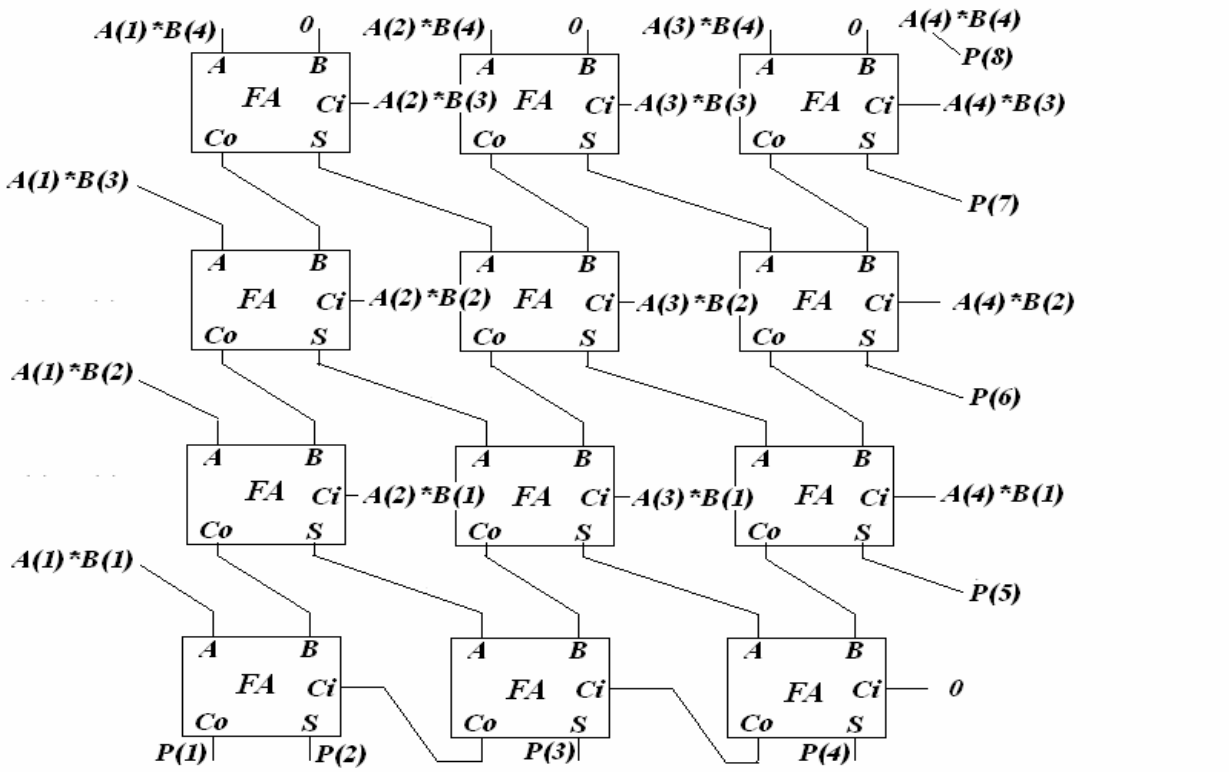
    …

- osztók:
    - előjel nélküli visszaállító osztó
    - előjeles nem visszaállító osztó
    - Radix 4 SRT osztó
    - funkcionális iterációs osztók (Newton-Raphson algoritmus)

      …

Ezeknek a funkcionális egységeknek a tételes és részletes bemutatása nem képezi ezen fejezet részét, de példaként egy tömbszorzó felépítésén keresztül betekintést nyerhetünk egy, a gyakorlatban gyakran alkalmazott logikai műveletvégző egység felépítésébe és működésébe.

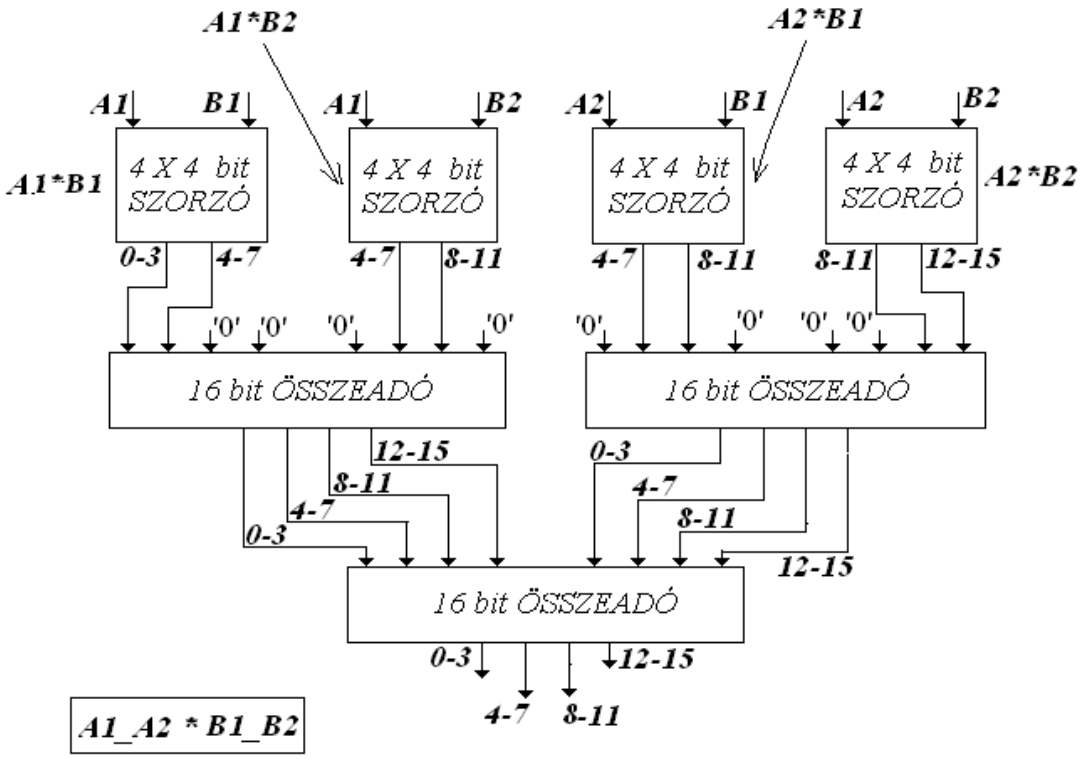
A 7.3.4.a ábrán egy 4x4 bites tömbszorzó (array-szorzó) kialakítását láthatjuk. A szorzókat, hasonlóan az összeadókhoz az jellemzi, hogy a bináris vektorok közötti bitenkénti 'szorzás-léptetés-összeadás' műveletsorát végzik el. A tömb speciálisan összekapcsolt teljes összeadókból áll. Az ***A(i)*B(j)*** 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 helyiértékek felé növekednek. Így a szorzat legnagyobb helyiértékén ***P(1)***, a legkisebb helyiértékén ***P(8)*** szorzat-bit szerepel. Hangsúlyozzuk, hogy az ábra szerinti tömb pozitív számok szorzására alkalmas, de hasonló tömbszorzó megoldásokat fejlesztettek ki kettes-komplemens kódban ábrázolt előjeles számok kettes-komplemens kódú eredményt szolgáltató szorzására is. Működés szempontjából lényeges tulajdonsága az egységnek, hogy az eredmény megjelenítési ideje, vagyis a blokk késleltetése a bitvektorok dimenzió számával közel lineárisan nő.



7.3.4.a ábra[1]

Egy 4 x 4–es tömbszorzó kialakítása teljes összeadókból valamint 'ÉS' kapukból

Nagyobb bitszámú egység kialakítására példa a 7.3.4.b ábrán látható, 4-bites tömbökből létrehozott 8x8 bites szorzó.

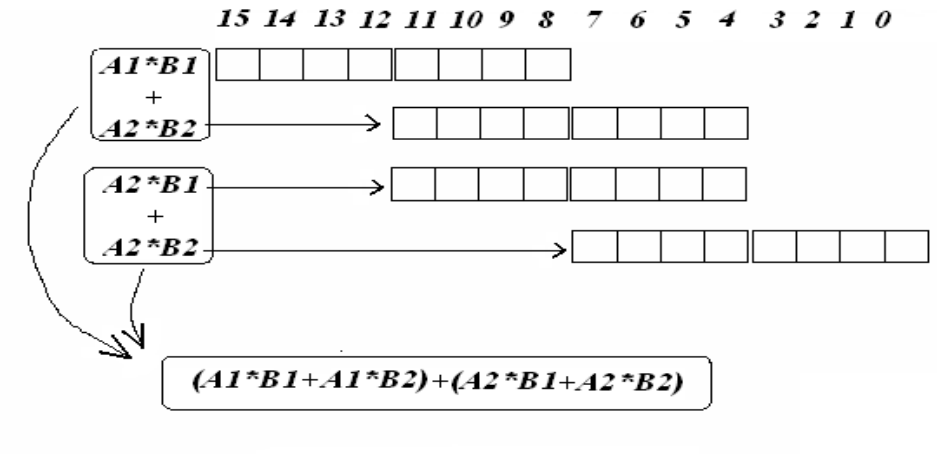


7.3.4.b ábra[1]

8 x 8 bites array-szorzó, 4 x 4-s komponensekből

Megjegyzés: A 7.3.4.b. ábra szerinti séma minden egyes összeköttetése egy 4-bites vektort jelöl. A helyes értelmezés: „A" szorzat MSB : 0, LSB: 15)

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 4x4 bites szorzókkal, részletekben is elő lehet állítani. Ezután minden részletszorzatot a maga helyiértékének megfelelő helyen kell 16-bites összeadókra vezetni (7.3.4.c ábra).



7.3.4.c ábra[1]

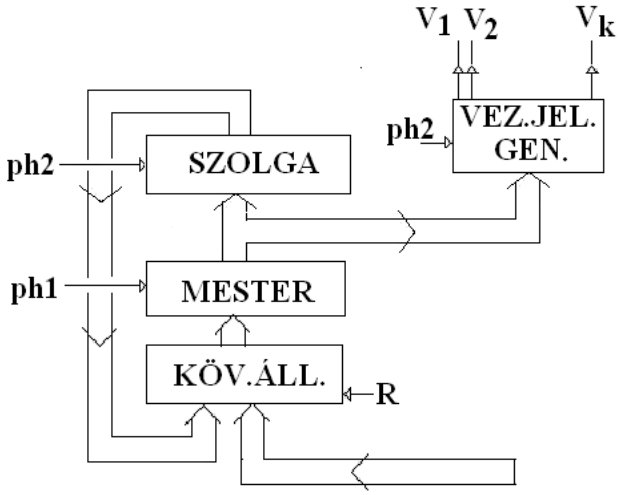
*Műveletvégzés a 8-bites array-szorzóban **A1_A2** és **B1_B2** operandusokkal*

7.3.5 Vezérlők

A 4.6.4 fejezetben már említést tettünk a digitális berendezések tervezésének egy fontos lépéséről, a dekompozícióról, melynek lényege, hogy elkülönítjük az adatutakat az azok kijelölését és időbeli mozgatását meghatározó időzítő vezérlő egységtől.

Mintapéldán keresztül (4.6.4.2) betekintést nyerhettünk egy szinkron számláló bázisú időzítő vezérlő egység tervezésébe és működtetésébe. Fontos gondolatként emeltük ki a hazárdmentes vezérlés biztosítását is.

A vezérlőknek egy másik csoportját alkotják az ún. FSM (Finite State Machine) típusú vezérlők. Strukturális kialakításukat tekintve egyik fő jellegzetességük, hogy az időzítő lényegében véges állapotszámú, MS tárolókból az adott célra felépített szinkron sorrendi hálózat. A kétfázisú (ph_1, ph_2) órajellel működtetett FSM vezérlő sematikus felépítése a 7.3.5.a ábrán látható:

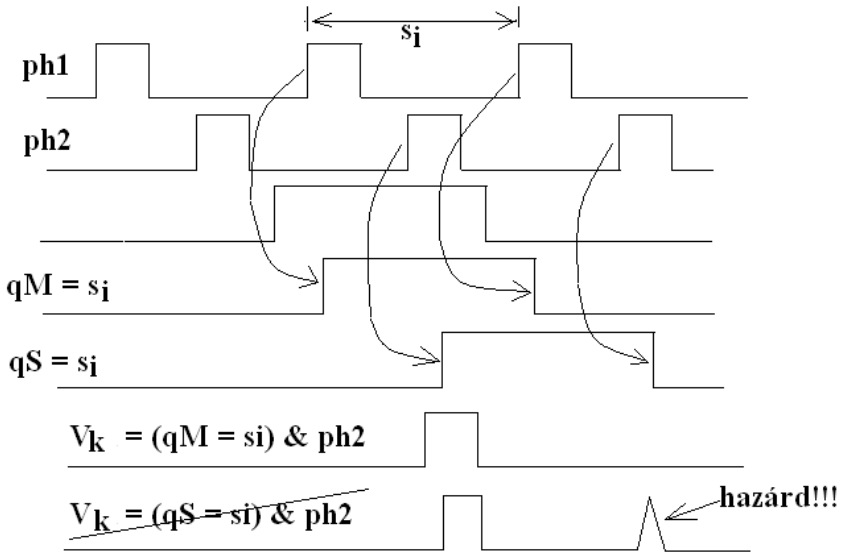


7.3.5.a ábra[1]

Kétfázisú órajellel(ph1,ph2) működő FSM típusú vezérlő

A rajzon jól felismerhető a klasszikus, kétfázisú szinkron sorrendi hálózat struktúra, ahogy a flip-flopok MESTER fokozatait is és a SZOLGA fokozatokat is összefogjuk egy-egy élvezérelt regiszterben. A működést reprezentáló idődiagramon (7.3.5.b ábra) az látható, hogyan kell kivitelezni egy ph_2 fázissal szinkronizált vezérlőjel generátorát: lényeges, hogy csak a MESTER fokozatról levett időzítő információ eredményez hazárdmentes megoldást. Fontos kitétel az is, hogy az állapotokhoz tartozó idő intervallumokat most a ph_1 felfutásai között kell definiálni.

Megjegyzés: 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.



7.3.5.b ábra[1]

Kétfázisú órajellel(ph1,ph2) működtetett FSM típusú időzítő idő-diagramja

Az FSM vezérlőket gyakran alkalmazzák önálló makro-cellákban. Az önálló makro-cellák LSI-VLSI áramkörök építőelemei lehetnek. Sajátosságuk, hogy egy speciális feladatra tartalmazzák mind az adatstruktúrát, mind az időzítő vezérlőt, és beilleszthetők egy nagyobb vezérlési folyamatba. Fontos, hogy több ilyen önálló egység osztozkodhat az adatstruktúra elemein, ha nem akadályozza ezt erőforrás igények közötti ütközés.

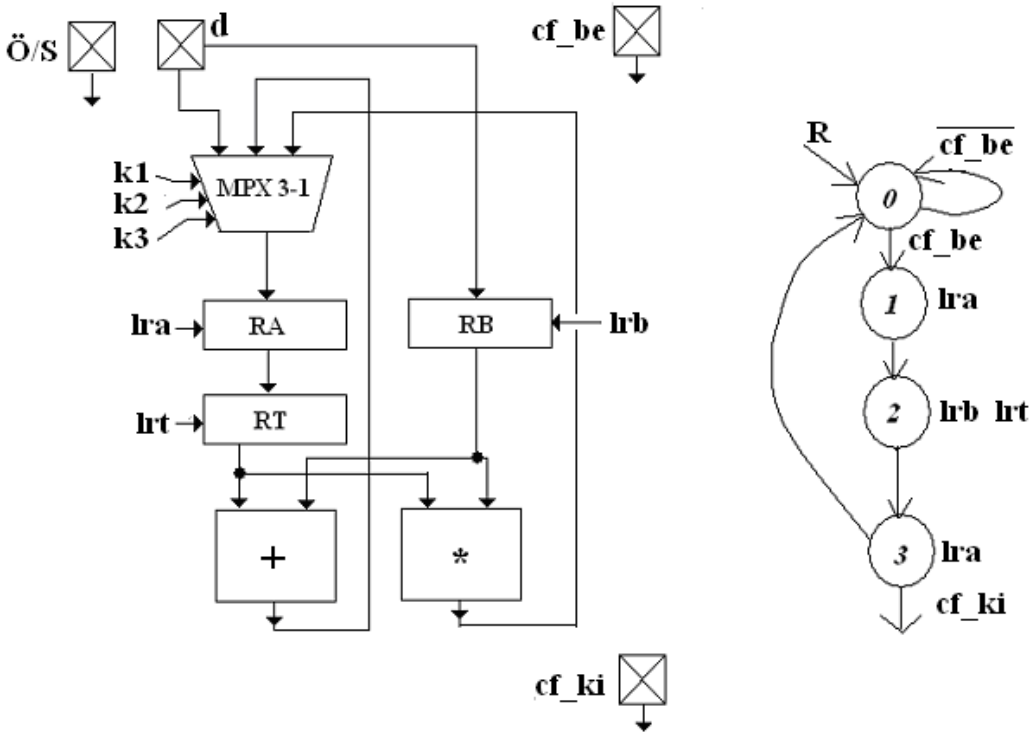
Példa:

*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'** adatsín 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 alapállapotába.*

Megoldás:

Ami az adatstruktúrát illeti, világos, hogy legalább két regiszterre és két funkciós egységre, egy összeadóra és egy szorzóra van szükségünk. Beláthatjuk azonban, hogy az

egyik operandusnak az eredménnyel való felülírása megköveteli egy átmeneti regiszter (***RT***) bevezetését is. Nyilvánvaló ugyanis, hogy a funkciós egységek kombinációs egységek, és bemenetükön nem írhatjuk felül azt az adatot, amely az eredmény stabilan tartásához és tárolásához szükséges. Ezért az ***RA*** tartalmát átmentjük egy átmeneti tárolóba, ezt kapcsoljuk a funkciós egységekre, és így az ***RA*** befogadhatja az eredményt. Azt is könnyen kitalálhatjuk, hogy az ***RA*** regiszterbe végül is három különböző forrásból kell tudni adatot betölteni. Ezek a '***d***' adatsín, az összeadó kimenete és a szorzó kimenete.



7.3.5.c ábra[1]

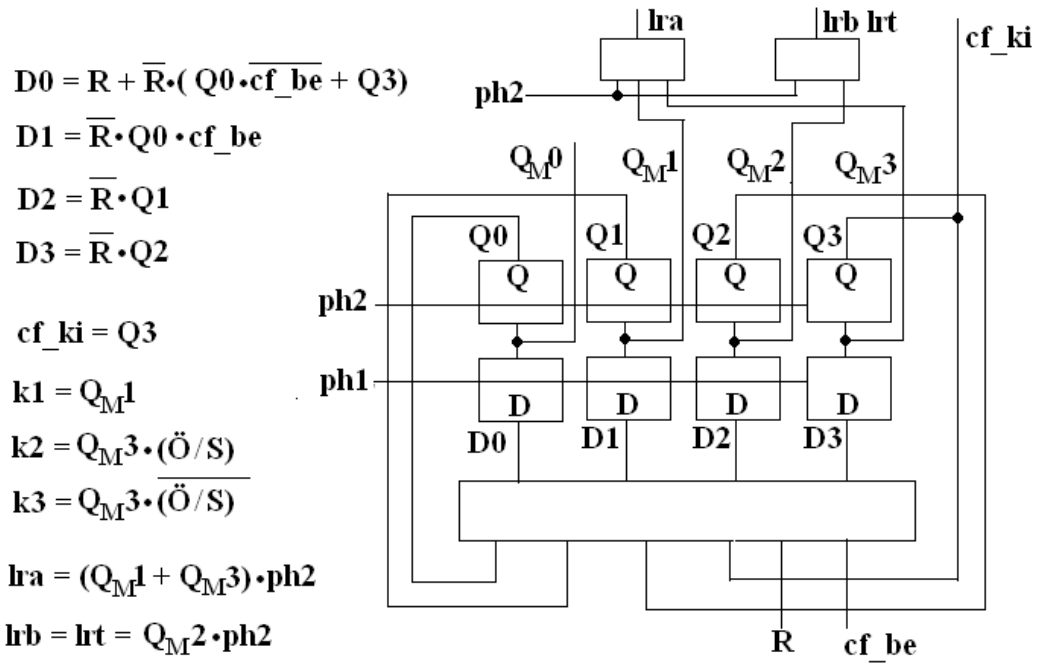
A mintapélda szerinti FSM típusú időzítő vezérlővel működő autonóm egység adatstruktúrája és időzítőjének állapotgráfja

Ezek után felvázolható a 7.3.5.c ábra bal oldalán látható adatstruktúra. Mindhárom regiszterből „kilógnak" a betöltőjelek, (***lra, lrb, lrt***) az MPX3-1 multiplexerből pedig a három kiválasztó bemenet, (***k1, k2, k3***). Természetesen a művelet-kiválasztó jel is szerepet kap a vezérlőben, csak úgy, mint az FSM-et elindító '***cf_be'*** külső vezérlő bemenet. Természetesen szolgáltatni kell a '***cf_ki'*** vezérlés továbbadó jelet is.

Az ábra jobb oldalán az FSM állapotgráffal megadott ütemezését 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. Ez alatt 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.

A megvalósított időzítő-vezérlő sémája a 7.3.5.d ábrán látható. A *'D'* bemenetekre csatlakozó fekete doboz tartalmát a baloldali logikai kifejezések mutatják. Az egyes *'D'* függvényalakokat elemezve látható, hogy '1'-es súlyú állapotkódolást alkalmaztunk.



7.3.5.d ábra[1]

A mintapéldában megfogalmazott időzítő vezérlő megvalósítása.

Felhasznált irodalom:

[1] Dr. Keresztes Péter: Digitális hálózatok (HEFOP elektronikus jegyzet)