

COLECȚIA DE AUTOMATICĂ

F. Moldoveanu D. Floroian

Circuite logice și comenzi secvențiale

Circuite logice combinaționale

Editura Universității Transilvania
Brașov, 2003

©2003 EDITURA UNIVERSITĂȚII TRANSILVANIA DIN BRAȘOV

Adresa: 500030 Brașov,
B-dul Eroilor, Nr. 9
Tel./Fax: 0268-475348
E-mail: repa@unitbv.ro

Tipărit la:



Tipografia Universității Transilvania din Brașov
B-dul Eroilor, Nr. 9
Tel./Fax: 0268-475348

Toate drepturile rezervate.

**Editură acreditată de CNCSIS.
Adresa nr. 1615 din 29 mai 2002.**

**Referenți științifici: Prof. dr. ing. Iulian ȚOPA
Prof. dr. ing. Ștefan DAN**

**Descrierea CIP a Bibliotecii Naționale a României
MOLDOVEANU, FLORIN**
**Circuite logice și comenzi sevențiale; Circuite logice
combinatoriale/ Florin Moldoveanu, Dan Floroian. – Brașov: Editura
Universității „Transilvania”, 2003**
Bibliogr.
Index
ISBN 973-635-237-4

I. Floroian, Dan

004.312

Editarea acestei lucrări s-a făcut cu sprijinul MECT, prin contractul nr. 40533,
grant nr. 7, cod CNCSIS 972.

CUPRINS

PREFĂTĂ	5
1. BAZELE TEORETICE ALE CIRCUITELOR LOGICE	7
1.1 ALGEBRA LOGICĂ.....	7
1.1.1 Axiomele și proprietățile algebrei booleene.....	8
1.2 FUNCȚII BOOLEENE	11
1.2.1 Operații cu funcții booleene	12
1.2.2 Funcții booleene elementare.....	13
1.2.3 Reprezentări ale funcțiilor booleene.....	16
1.2.3.1 Reprezentarea funcțiilor booleene prin tabele de adevăr	16
1.2.3.2 Reprezentarea funcțiilor booleene prin diagrame Karnaugh.....	16
1.2.3.3 Reprezentarea analitică a funcțiilor booleene.....	17
1.2.3.4 Reprezentarea funcțiilor booleene prin scheme logice.....	20
1.2.3.5 Reprezentarea funcțiilor booleene prin diagrame de timp	21
1.2.4 Sisteme complete de funcții	23
1.2.4.1 Implementarea FCD și FCC cu funcții Pierce și Sheffer	24
2. CIRCUITE LOGICE COMBINAȚIONALE	27
2.1 NOTIUNI INTRODUCTIVE	27
2.2 ANALIZA CIRCUITELOR LOGICE COMBINAȚIONALE	28
2.2.1 Analiza schemelor logice combinaționale realizate cu elemente logice ȘI, SAU, NU	29
2.2.2 Analiza schemelor logice combinaționale realizate cu elemente logice ȘI-NU și SAU-NU	32
2.3 SINTEZA CIRCUITELOR LOGICE COMBINAȚIONALE.....	35
2.3.1 Minimizarea funcțiilor booleene	35
2.3.1.1 Minimizarea funcțiilor booleene pe baza axiomelor și teoremelor algebrei booleene	38
2.3.1.2 Minimizarea funcțiilor booleene prin metoda Quine-McCluskey	38

CUPRINS

2.3.1.3 Minimizarea funcțiilor booleene prin metoda diagramelor Karnaugh.....	46
2.3.1.4 Minimizarea funcțiilor booleene prin metoda consensurilor	54
2.3.1.5 Minimizarea funcțiilor booleene incomplet definite	60
2.3.1.6 Minimizarea sistemelor de funcții booleene	65
2.3.2 Implementarea funcțiilor booleene	69
2.3.2.1 Implementarea funcțiilor booleene cu relee electromagnetice ...	69
2.3.2.2 Implementarea funcțiilor booleene cu circuite integrate	71
2.3.2.2.1 Implementarea cu circuite integrate SSI.....	73
2.3.2.2.2 Implementarea cu circuite integrate MSI	83
2.3.2.2.3 Implementarea cu circuite integrate LSI	104
2.4 HAZARDUL ÎN CIRCUITELE LOGICE COMBINATORIALE	133
2.5 ALTE TIPURI DE CIRCUITE LOGICE COMBINATORIALE	142
2.5.1 Convertoare de cod	142
2.5.2 Decodificatorul BCD-7 segmente.....	148
2.5.3 Circuite codificatoare.....	150
2.5.4 Comparatoare numerice.....	157
2.5.5 Detectoare și generatoare de paritate	161
2.5.6 Sumatoare binare	165
2.5.7 Circuite generatoare de funcții.....	180
2.5.8 Unitatea aritmetică și logică	181
2.5.9 Circuite multiplicatoare	188
ANEXA A	195
ANEXA B	197
ANEXA C	205
ANEXA D	219
BIBLIOGRAFIE	231
INDEX	233

PREFATĂ

Această carte se adresează tuturor acelora care doresc să se familiarizeze cu tehniciile specifice analizei și sintezei sistemelor logice de comandă: studenți ai specializărilor de automatică, calculatoare, electronică, electrotehnică, ingineri specialiști în domeniile menționate precum și alțor specialiști care sunt preocupați de aplicarea tehniciilor digitale în diferite domenii ale științei și tehnicii.

Organizarea materialului cărții a avut la bază ideea că atingerea scopului propus trebuie să aibă în vedere atât însușirea cunoștințelor fundamentale dar și înțelegerea modului în care se petrec lucrurile în „lumea reală”.

Deoarece volumul de cunoștințe specifice domeniului tratat este destul de mare autorii au considerat utilă structurarea materialului în două părți: prima, care face obiectul volumului de față, se focalizează pe studiul circuitelor logice combinaționale, urmând ca cea de a doua, ce va fi publicată separat, să se ocupe de circuitele logice secvențiale.

În prima parte a lucrării sunt tratate noțiunile fundamentale de algebră logică (booleană), necesare pentru analiza și sinteza circuitelor de comutare combinaționale și secvențiale, din punctul de vedere al inginerului proiectant de circuite digitale. Din acest motiv, pentru demonstrarea multora dintre teoremele utilizate se fac trimiteri la literatura de specialitate, reținându-se, în cadrul cărții, numai partea aplicativă (reguli practice de utilizare, algoritmii de rezolvare a problemelor, etc.).

După prezentarea metodelor specifice de analiză a circuitelor logice combinaționale, în continuare, în lucrare sunt expuse, într-o manieră pe cât posibil unitară, principiile proiectării acestui tip de circuite logice, folosindu-se operatori logici de tipuri diferite și făcând parte din familii diferite, integrate atât pe scară mică (SSI) cât și pe scară medie (MSI) sau mare (LSI).

În general, funcția de prelucrare realizată de către un circuit logic poate fi privită independent de modul în care acesta este construit. Funcția poate fi definită și se poate opera cu această noțiune, până la un anumit punct, fără a se preciza modul în care ea este realizată. Mai mult decât atât, este posibil ca un anumit circuit logic să poată realiza nu doar o singură funcție de prelucrare a semnalelor ci o familie de funcții asemănătoare.

Pe de altă parte, modul de realizare a unei funcții nu este unic. Ca urmare, circuitului logic i se asociază un set de parametrii care arată în ce condiții și cât de bine

acesta realizează funcția dorită. S-a denumit ca fiind implementarea unei funcții, modul în care aceasta este realizată (modelată fizic) de către un circuit logic. Cele două aspecte enunțate mai sus se pot deci sintetiza: în funcția realizată de circuit și în implementarea acestei funcții.

Din mulțimea funcțiilor posibile, ca rezultat al interdependenței procesului de utilizare și a celui de implementare, s-a impus un număr relativ redus de funcții. S-a creat astfel un set de funcții de bază, grupate în clase de funcții de prelucrare a semnalelor, cu variantele lor de implementare, grupate în familii de circuite.

Dezvoltarea explozivă a tehnologiilor a determinat diversificarea extraordinară a tipurilor de funcții, făcând posibilă realizarea unor prelucrări de semnale, asemănătoare sau distințe, cu un singur circuit, deci cu o singură variantă de implementare. S-au obținut astfel circuite multifuncționale care, prin modificări minime de schemă (hardware) pot realiza una sau alta dintre funcțiile de prelucrare din clasa pentru care au fost construite. Vârful atins în această direcție îl constituie circuitele a căror funcție de prelucrare poate fi modificată prin program (software).

Lucrarea, deși acoperă o gamă largă de circuite, nu prezintă toate tipurile de circuite digitale. Acest domeniu al circuitelor logice moderne are o evoluție extrem de rapidă, noi și noi tipuri de circuite făcându-și apariția la intervale scurte de timp, motiv pentru care prezentarea acestora la zi nici nu este posibilă.

Trebuie precizat că pentru proiectarea unui sistem digital mai mult sau mai puțin complex, este necesar ca informațiile prezentate în această lucrare să fie completate cu datele din cataloagele și din notele de aplicații prezentate de firmele producătoare.

Autorii aduc și pe această cale mulțumiri domnilor profesori: prof. dr. ing. Iulian ȚOPA și prof. dr. ing. Ștefan DAN, referenții științifici ai prezentei lucrări, care, prin citirea atentă a manuscrisului și prin observațiile și sugestiile făcute, referitoare la structurarea materialului, au contribuit la aducerea manualului la forma pe care autorii o prezintă cititorilor.

Domnului ing. Ștefan HURUBEANU îi suntem recunoscători pentru discuțiile tehnice deosebit de utile pe care le-am avut pe marginea materialului conținut în această lucrare.

Autorii apreciază contribuția personală la realizarea prezentei lucrări după cum urmează: Florin Moldoveanu 70% respectiv Dan Floroian 30%, din volumul acesteia.

De asemenea, mulțumim anticipat tuturor acelora, care, în urma lecturii cărții, vor transmite la adresele moldof@unitbv.ro și danf@unitbv.ro observațiile și sugestiile lor privind conținutul, modul de tratare a problemelor, forma de prezentare și vor sesiza eventualele greșeli apărute, astfel încât să poată fi utilizate într-o ediție ulterioară a acestui text.

Brașov, noiembrie 2003

Florin MOLDOVEANU
Dan FLOROIAN

1

BAZELE TEORETICE ALE CIRCUITELOR LOGICE

1.1 ALGEBRA LOGICĂ

Definirea riguroasă a problemelor tehnice privind circuitele logice se poate face folosind principiile logicii matematice, în particular principiile calculului propozițiilor.

Spre mijlocul secolului al XIX-lea, matematicianul și logicianul englez George Boole (1815–1864) a propus o interpretare a logicii propozițiilor bivalente, fundamentând algebra propozițiilor cu două valori care adesea este denumită *algebra logicii* sau *algebra booleană*. Apariția elementelor, circuitelor și sistemelor care în funcționare pot avea doar două stări stabile distințe a condus la aplicarea în tehnică a perceptelor logicii bivalente.

Algebrele booleene și în special algebra booleană cu două valori constituie fundația teoretică a circuitelor logice.

Algebra logică, numită, așa cum s-a mai precizat și calculul propozițional, operează cu propoziții, despre care are sens să afirmăm că sunt adevărate sau false. Din însăși definirea propozițiilor rezultă că o propoziție poate fi adevărată sau falsă. Propozițiile pot fi simple și compuse. Cele compuse se obțin ca rezultat al legăturii propozițiilor simple, prin intermediul unor conective logice. Adevărul sau falsitatea unei propoziții compuse este funcție de valoarea propozițiilor simple din care se compune și de tipul legăturii logice.

În logica simbolică se poate face abstracție de sensul propozițiilor, operând cu relațiile de valoare. Convenim ca unei propoziții adevărate să-i atribuim valoarea binară 1, iar falsitatea acesteia să o notăm cu valoarea binară zero (0).

Propoziția compusă a cărei valoare depinde de valorile propozițiilor simple, putând avea tot două valori, se numește *funcție logică* sau *funcție binară*.

Funcția compusă este complet definită cu ajutorul unui tabel finit, în care se trec valorile funcției în corespondență cu valorile propozițiilor simple considerate independente. O astfel de exprimare a funcției compuse este cunoscută sub numele de *tabel de adevăr*.

Calculul propozițiilor poate fi extrapolat în tehnică la sistemele automate care utilizează elemente de comutăție cu două stări. Acestea le putem atribui relații de valoare (1 sau 0) binară corespunzătoare adevărului (1) și falsității (0) propozițiilor.

1.1.1 Axiomele și proprietățile algebrei booleene

Considerând cunoscute noțiunile elementare de teoria mulțimilor, în continuare se vor defini axiomele și proprietățile algebrei booleene pornind de la structura algebraică, mai generală, de *latice* [3,5,10,20,28].

Fie A o mulțime nevidă, împreună cu două operații binare pe A , denumite *reuniune* și *intersecție* și notate cu \cup și \cap .

Prin definiție, tripletul:

$$L = (A, \cup, \cap), \quad (1.1)$$

este o *latice* și se bucură de următoarele proprietăți:

- *comutativitatea*:

$$a_1 \cup a_2 = a_2 \cup a_1, \quad a_1 \cap a_2 = a_2 \cap a_1, \quad \forall a_1, a_2 \in A, \quad (1.2)$$

- *asociativitatea*:

$$a_1 \cup (a_2 \cup a_3) = (a_1 \cup a_2) \cup a_3, \quad a_1 \cap (a_2 \cap a_3) = (a_1 \cap a_2) \cap a_3,$$

$$\forall a_1, a_2, a_3 \in A, \quad (1.3)$$

- *absorbția*:

$$a_1 \cup (a_1 \cap a_2) = a_1, \quad a_1 \cap (a_1 \cup a_2) = a_1, \quad \forall a_1, a_2 \in A. \quad (1.4)$$

Proprietățile (1.2) ÷ (1.4) constituie axiome pentru latici. Se poate observa că în acest sistem de axiome se pot schimba între ele simbolurile \cup și \cap . Evident, acest lucru se poate face în orice afirmație care decurge din sistemul de axiome, proprietate cunoscută sub denumirea de *principiul dualității pentru latici*.

Având în vedere proprietățile (1.2) și (1.3), operațiile de reuniune și de intersecție se pot extinde la orice număr arbitrar, dar finit, de termeni, indiferent de ordinea termenilor sau factorilor:

$$\bigcup_{i=1}^n a_i = a_1 \cup a_2 \cup \dots \cup a_n, \quad \bigcap_{i=1}^n a_i = a_1 \cap a_2 \cap \dots \cap a_n. \quad (1.5)$$

Plecând de la axiomele definite mai sus se poate demonstra și următoarea proprietate:

- *idempotența*:

$$a \cup a \cup \dots \cup a = a, \quad a \cap a \cap \dots \cap a = a, \quad \forall a \in A. \quad (1.6)$$

O latice se poate defini și ca o mulțime parțial ordonată $L = (A, \leq)$, care are o cea mai mică margine superioară (c.m.m.m.s.) – s și o cea mai mare margine inferioară (c.m.m.m.i.) – p , pentru fiecare pereche de elemente. Legătura între cele două definiții se

poate face notând $s = a_1 \cup a_2$, $p = a_1 \cap a_2$.

Prin definiție, o latice finită (mărginită) are un element care este c.m.m.m.i., numit *ultim element al laticei*, notat prin 0, astfel încât:

- *legile lui 0*:

$$a \cup 0 = a, a \cap 0 = 0, \forall a \in A \quad (1.7)$$

și un element care este c.m.m.m.s., numit *prim element al laticei*, notat prin 1, astfel încât:

- *legile lui 1*:

$$a \cup 1 = 1, a \cap 1 = a, \forall a \in A. \quad (1.8)$$

Fie $L = (A, \cup, \cap, 0, 1)$ o latice finită și $a \in A$. Un *element complementar* sau, pe scurt, un *complement* al elementului a , este elementul \bar{a} (non a), astfel încât:

- *principiul terțului exclus*:

$$a \cup \bar{a} = 1, \quad (1.9)$$

care atestă faptul că nu există o a treia posibilitate într-o reuniune cu variabile complementare și:

- *principiul contradicției*:

$$a \cap \bar{a} = 0. \quad (1.10)$$

Trebuie menționat că nu orice element dintr-o latice finită are un complement. Astfel, în laticea finită $L = (\{0, a, 1\}, \cup, \cap, 0, 1)$, elementul a nu are complement. De asemenea, complementul unui element al unei latice, dacă acesta există, nu este în mod necesar unic. În schimb, elementele 0 și 1 au fiecare un complement unic, respectiv 1 și 0: $\bar{0} = 1$, $\bar{1} = 0$.

Dacă într-o latice finită orice element a are un complement unic \bar{a} , această latice se numește *complementară*.

Prin definiție o latice L este *distributivă* dacă și numai dacă:

- *distributivitatea*:

$$(a_1 \cup a_2) \cap a_3 = (a_1 \cap a_3) \cup (a_2 \cap a_3), \forall a_1, a_2, a_3 \in A, \quad (1.11)$$

$$(a_1 \cap a_2) \cup a_3 = (a_1 \cup a_3) \cap (a_2 \cup a_3), \forall a_1, a_2, a_3 \in A. \quad (1.12)$$

Proprietatea (1.11) poartă denumirea de distributivitatea reuniunii în raport cu intersecția iar (1.12) distributivitatea intersecției în raport cu reuniunea.

Definiție. O algebră booleană este o latice distributivă și complementară. Din definiție rezultă că o algebră booleană este un 4-uplu:

$$B = (A, \cup, \cap, -), \quad (1.13)$$

în care „-“ este operația unară de complementare.

Într-o algebră Boole se mai pot demonstra și următoarele proprietăți care au o deosebită importanță pentru studiul circuitelor de comutare:

- principiul involuției (dublei negații):

$$\bar{\bar{a}} = a, \forall a \in A. \quad (1.14)$$

De asemenea, într-o algebră Boole sunt adevărate:

- relațiile lui De Morgan:

$$\overline{a_1 \cup a_2} = \overline{a_1} \cap \overline{a_2}, \quad \forall a_1, a_2 \in A, \quad (1.15)$$

$$\overline{a_1 \cap a_2} = \overline{a_1} \cup \overline{a_2}, \quad \forall a_1, a_2 \in A. \quad (1.16)$$

Dacă într-o algebră Boole mulțimea A are numai elementele 0 și 1, se obține algebra Boole cu două elemente:

$$\mathbf{B}_\gamma = (\{0,1\}, \cup, \cap, -), \quad (1.17)$$

în care, operațiile sunt date în următoarele tabele:

Tab.1.1 Tabelele operatiilor de reuniune (a), intersectie (b) si complementare (c).

\cup	0	1	\cap	0	1	-	0	1
0	0	1	0	0	0	-	1	0
1	1	1	1	0	1	-	1	0

Din cele prezentate mai sus rezultă că în algebra Boole calculul este definit prin relațiile (1.2) \div (1.16). Pentru cele trei operații, în afara denumirilor menționate se mai folosesc și următoarele:

- operația SAU, numită și disjuncție ori sumă logică pentru reuniune, fiind notată și cu simbolurile „ \vee “ sau „ $+$ “. Astfel, următoarele notații sunt echivalente: $a_1 \cup a_2 = a_1 \vee a_2 = a_1 + a_2$. Din motive de simplitate a exprimării, în această lucrare se va folosi simbolul „ $+$ “;
 - operația ŞI, numită și conjuncție ori produs logic pentru intersecție, fiind notată și cu simbolurile „ $\&$ “ sau „ $:$ “. Ultimul simbol este cel mai folosit, în scriere, el putând fi omis. Astfel, următoarele notații sunt echivalente: $a_1 \cap a_2 = a_1 \& a_2 = a_1 \cdot a_2 = a_1 a_2$;
 - operația NU sau negația logică pentru complementare, notațiile următoare fiind echivalente: $\bar{a} = a^c = a'$.

Algebra booleană cu două elemente are aplicație directă în teoria circuitelor logice. În acest caz, între valorile mulțimii {0,1} și cele două stări ale elementelor funcționând în regim de comutare se stabilește o corespondență biunivocă. Astfel, o variabilă asociată unui element de comutare poate lua numai două valori, 0 sau 1, definind o variabilă bivalentă booleană sau, pe scurt, o *variabilă booleană*. Rezultă că pentru circuitele

materializate cu elemente de comutație, modelul matematic îl constituie funcțiile de variabile binare. Deoarece circuitele realizate cu elemente binare nu pot avea decât două stări stabile distințe, funcțiile care descriu aceste circuite vor lua numai două valori. Aceste funcții bivalente de variabile binare se numesc *funcții booleene* sau *funcții logice* și au o deosebită importanță pentru studiul circuitelor logice.

1.2 FUNCȚII BOOLEENE

Pentru definirea funcțiilor booleene [3,5,10,11,20,28,33] se consideră vectorul $X = (x_1, x_2, \dots, x_n)$, ale cărui elemente (x_1, x_2, \dots, x_n) pot lua valorile 0 sau 1. În acest caz rezultă că pot exista 2^n vectori X . Se notează mulțimea acestor vectori cu B_2^n . De asemenea, fiecărui vector din B_2^n i se pot atribui valorile 0 sau 1.

Definiție. Se numește *funcție booleană* funcția $f(X) = f(x_1, x_2, \dots, x_n)$, $X = (x_1, x_2, \dots, x_n)$, care aplică mulțimea B_2^n în mulțimea $\{0,1\}$.

Fie K o submulțime a lui B_2^n și \bar{K} complementara lui K față de B_2^n : $K, \bar{K} \subset B_2^n$, $K \cup \bar{K} = B_2^n$, $K \cap \bar{K} = \emptyset$. Atunci o funcție booleană de n argumente, $f(X) = f(x_1, x_2, \dots, x_n)$, $X = (x_1, x_2, \dots, x_n)$, se poate defini și astfel:

$$X \in K \rightarrow f(X) = 1, X \in \bar{K} \rightarrow f(X) = 0. \quad (1.18)$$

Deci, unei funcții booleene i se asociază un vector $V_f = (f(X))$ cu 2^n componente egale cu 0 sau 1, fiecare componentă fiind asociată unui vector X dat.

Consecință. Deoarece există 2^{2^n} vectori bivalenți de 2^n elemente, rezultă că numărul funcțiilor booleene distințe de n argumente este finit și egal cu 2^{2^n} .

Se notează valorile fixe ale elementelor unui vector din B_2^n prin $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n)$. Aceste valori pot fi privite ca o combinație de valori ale argumentelor unei funcții booleene. Deoarece numărul acestor combinații este finit și egal cu 2^n , atunci orice funcție booleană poate fi complet definită printr-un tabel finit cu 2^n linii. În acest tabel, în partea stângă se trec combinațiile valorilor argumentelor iar în partea dreaptă valorile corespunzătoare, 0 sau 1, ale funcției (v. tab. 1.2), cu $\alpha_i \in \{0,1\}$.

Există situații când pentru unele combinații ale valorilor argumentelor o funcție booleană să nu aibă valoarea determinată. Astfel de funcții nedeterminate, pentru una sau mai multe combinații ale valorilor argumentelor, se numesc *funcții booleene incomplet definite*. În mod obișnuit în tabelele de definiție ale funcțiilor, valorile nedeterminate sunt indicate cu (*) sau d^1 . Tabelul de definiție al unei funcții incomplet definite este reprezentat în tabelul 1.3. Funcția este nedeterminată pentru combinațiile (0,1,0), (1,0,1) și (1,1,0) ale valorilor argumentelor, valorile putând fi aplicate arbitrar din mulțimea $\{0,1\}$. Atribuind funcției valorile 0 sau 1 pentru combinațiile respective ale valorilor argumentelor se pot obține 8 funcții complet definite.

¹ Simbolul „ d ” provine de la prescurtarea expresiei englezesti „don't care” (nu ține cont).

Tab.1.2 Tabel ce definește complet o funcție booleană.

x_1	x_2	...	x_{n-1}	x_n	$f(x_1, x_2, \dots, x_n)$
0	0	...	0	0	α_1
0	0	...	0	1	α_2
0	0	...	1	0	α_3
...
1	1	...	1	1	α_n

În general, dacă o funcție booleană nu este definită pentru r combinații ale valorilor argumentelor atunci, prin definire arbitrară, se pot obține 2^r funcții noi complet definite. Funcțiile incomplet definite se întâlnesc frecvent în practică, evidențieră situațiilor de nedefinire și atribuirea voită a valorilor 0 sau 1 fiind foarte importantă pentru simplificarea lor.

1.2.1 Operații cu funcții booleene

Operațiile cu funcții booleene se definesc pe domeniul valorilor funcțiilor [3,10,20,28]. Se consideră două funcții $f_1(x_1, x_2, \dots, x_n)$ și $f_2(x_1, x_2, \dots, x_n)$. Se spune că aceste funcții sunt *identice* dacă iau valori identice pentru toate combinațiile posibile ale valorilor argumentelor. În mod obișnuit identitatea a două funcții booleene se scrie astfel:

$$f_1(x_1, x_2, \dots, x_n) = f_2(x_1, x_2, \dots, x_n). \quad (1.19)$$

Dacă pentru cel puțin o singură combinație a valorilor argumentelor (un n -uplu) cele două funcții nu au aceeași valoare atunci:

$$f_1(x_1, x_2, \dots, x_n) \neq f_2(x_1, x_2, \dots, x_n). \quad (1.20)$$

Fie f_1 , f_2 , și f_3 funcții booleene de n argumente. Operațiile „+“, „:“, „-“, cu funcții se definesc în modul următor:

- *reuniunea (suma logică) funcțiilor:*

$$f_1(x_1, x_2, \dots, x_n) + f_2(x_1, x_2, \dots, x_n) = f_3(x_1, x_2, \dots, x_n), \quad (1.21)$$

dacă și numai dacă valorile funcțiilor se combină corespunzător tabelului operației „+“, pentru fiecare combinație a valorilor argumentelor;

- *intersecția (produsul logic) funcțiilor:*

$$f_1(x_1, x_2, \dots, x_n) \cdot f_2(x_1, x_2, \dots, x_n) = f_3(x_1, x_2, \dots, x_n), \quad (1.22)$$

Tab.1.3 Tabelul de definiție al unei funcții booleene incomplet definite.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	*
0	1	1	1
1	0	0	0
1	0	1	*
1	1	0	*
1	1	1	1

dacă și numai dacă valorile funcțiilor se combină conform tabelului operației „·“, pentru fiecare n -uple al argumentelor;

- negarea (complementarea) funcțiilor:

$$\bar{f}_1(x_1, x_2, \dots, x_n) = f_2(x_1, x_2, \dots, x_n), \quad (1.23)$$

dacă și numai dacă valorile funcțiilor respectă tabelul de definiție al operației de complementare.

Din cele de mai sus rezultă că pentru operarea cu funcții booleene se consideră succesiv valorile funcțiilor corespunzătoare celor 2^n combinații ale argumentelor. Funcțiile f_1 și f_2 aplică fiecare din cele 2^n n -uple în mulțimea $\{0,1\}$. Se obțin astfel 2^n perechi de valori ale funcțiilor. Operațiile binare SAU (suma logică) și SI (produsul logic) asupra celor două funcții aplică cele 2^n perechi în mulțimea $\{0,1\}$. Operația unară de complementare aplică cele 2^n n -uple ale unei funcții în $\{0,1\}$. Pentru operarea cu funcții booleene este avantajoasă folosirea tabelelor.

1.2.2 Funcții booleene elementare

În acest paragraf se vor defini funcțiile booleene fundamentale, cu ajutorul cărora se pot construi funcții mai complexe [3,5,10,20,28].

Aceste funcții booleene denumite și *funcții elementare* au o deosebită importanță practică pentru realizarea circuitelor logice modulare. În mod obișnuit, funcțiile elementare se definesc pe mulțimea funcțiilor de două argumente. Cele $2^{2^2} = 16$ funcții booleene de două argumente sunt prezentate în tabelul 1.4.

Tab.1.4 Definirea funcțiilor booleene elementare.

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	1	0	1	0	1	0	1	0	1	0	1	0	1	0	
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	
1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	

Din examinarea tabelului rezultă și se definesc următoarele funcții elementare:

• funcțiile f_0 și f_{15} sunt funcții constante, nedepinzând de argumentele x_1 și x_2 . Sunt denumite și funcțiile logice *constanta 0* și respectiv *constanta 1*:

$$f_0(x_1, x_2) = 0, \quad f_{15}(x_1, x_2) = 1, \quad (1.24)$$

- funcțiile f_{10} și f_{12} corespund valorilor argumentelor:

$$f_{10}(x_1, x_2) = x_2, \quad f_{12}(x_1, x_2) = x_1 \quad (1.25)$$

și care se mai numesc *funcții identitate*;

- funcțiile f_3 și f_5 corespund funcțiilor f_{10} și f_{12} negate:

$$f_3(x_1, x_2) = \bar{f}_{12}(x_1, x_2) = \bar{x}_1, \quad f_5(x_1, x_2) = \bar{f}_{10}(x_1, x_2) = \bar{x}_2 \quad (1.26)$$

și se numesc *funcțiile negație*.

Observație. Funcțiile definite mai sus depind numai de unul dintre argumente sau de nici unul. Se spune că aceste funcții elementare sunt funcții degenerate de două argumente.

- funcția f_8 corespunde produsului logic al funcțiilor f_{10} și f_{12} și se numește *funcția conjuncție* sau *funcția ȘI*:

$$f_8(x_1, x_2) = f_{10}(x_1, x_2) \cdot f_{12}(x_1, x_2) = x_1 x_2, \quad (1.27)$$

- funcția f_{14} corespunde sumei logice a funcțiilor f_{10} și f_{12} , numindu-se *funcția disjuncție* sau *funcția SAU*:

$$f_{14}(x_1, x_2) = f_{10}(x_1, x_2) + f_{12}(x_1, x_2) = x_1 + x_2, \quad (1.28)$$

- funcția f_1 poartă denumirea de *funcția lui Pierce* sau *funcția lui Webb* și se notează în modul următor:

$$f_1(x_1, x_2) = x_1 \downarrow x_2. \quad (1.29)$$

Examinând valorile acestei funcții se observă că ea reprezintă negata funcției disjunctive:

$$f_1(x_1, x_2) = \bar{f}_{14}(x_1, x_2) = \overline{x_1 + x_2} = \bar{x}_1 \cdot \bar{x}_2. \quad (1.30)$$

Având în vedere cele de mai sus se mai poate scrie:

$$f_1(x_1, x_2) = f_3(x_1, x_2) \cdot f_5(x_1, x_2) = f_{10}(x_1, x_2) \downarrow f_{12}(x_1, x_2). \quad (1.31)$$

Din relația precedentă se observă că simbolul „ \downarrow “ are rol de operator, definind situația de „nici x_1 și nici x_2 “. Din acest motiv funcția f_1 se mai numește și *funcția NICI*. În literatură este întâlnită și sub denumirea de *funcția NOR*, provenind din limba engleză (NOT OR = NOR).

- funcția f_7 se numește *funcția lui Sheffer* și se simbolizează astfel:

$$f_7(x_1, x_2) = x_1 \uparrow x_2 = x_1 / x_2. \quad (1.32)$$

Examinând în tabel valorile acestei funcții, se poate observa că ea reprezintă negarea funcției conjunctive de unde își rezultă și denumirea de *funcție ȘI-NU* sau *funcție NAND* (în limba engleză NOT AND \equiv NAND):

$$f_7(x_1, x_2) = \bar{f}_8(x_1, x_2) = \overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2 = f_{10}(x_1, x_2) \uparrow f_{12}(x_1, x_2). \quad (1.33)$$

Corespunzător relațiilor de mai sus se mai poate spune că funcția lui Sheffer definește situația de „numai x_1 sau numai x_2 ”, fapt ce a condus și la denumirea de *funcție NUMAI*.

Funcțiile NICI și NUMAI au o deosebită importanță atât pentru teoria funcțiilor booleene cât și pentru aplicarea practică a acesteia în realizarea circuitelor logice modulare.

- funcția f_9 are de asemenea importanță pentru teoria funcțiilor booleene, fiind denumită *funcția echivalență*:

$$f_9(x_1, x_2) = x_1 \sim x_2, \quad (1.34)$$

desemnând „echivalență între x_1 și x_2 “. Se poate demonstra că această funcție poate fi descrisă prin funcțiile conjuncție, disjuncție și negație:

$$f_9(x_1, x_2) = x_1 \sim x_2 = (\bar{x}_1 + x_2)(x_1 + \bar{x}_2). \quad (1.35)$$

Într-adevăr, folosind tabelele (v. tab. 1.5.a,b), pentru cele două părți ale identității, se obține rezultatul precizat mai sus. Întrucât pentru toate combinațiile posibile ale valorilor argumentelor cele două părți ale identității au aceleași valori, demonstrația este făcută.

Tab.1.5 Tabelele funcției echivalență.

x_1	x_2	$x_1 \sim x_2$	x_1	x_2	\bar{x}_1	\bar{x}_2	$\bar{x}_1 + x_2$	$x_1 + \bar{x}_2$	$(\bar{x}_1 + x_2)(x_1 + \bar{x}_2)$
0	0	1	0	0	1	1	1	1	1
0	1	0	0	1	1	0	1	0	0
1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	1

(a)

(b)

- funcția f_6 se numește *funcția suma modulo 2* sau *funcția SAU-EXCLUSIV* și se notează astfel:

$$f_6(x_1, x_2) = x_1 \oplus x_2. \quad (1.36)$$

Comparând în tabelul de definiție valorile acestei funcții cu ale funcției f_9 , rezultă:

$$f_6(x_1, x_2) = \bar{f}_9(x_1, x_2) = \overline{x_1 \sim x_2} = \overline{(\bar{x}_1 + x_2)(x_1 + \bar{x}_2)} = x_1 \bar{x}_2 + \bar{x}_1 x_2, \quad (1.37)$$

- funcțiile f_{11} și f_{13} sunt denumite *funcțiile implicație*:

$$f_{11}(x_1, x_2) = x_1 \rightarrow x_2, \text{ implicația lui } x_1 \text{ în } x_2, \quad (1.38)$$

$$f_{13}(x_1, x_2) = x_1 \leftarrow x_2, \text{ implicația lui } x_2 \text{ în } x_1, \text{ sau implicația inversă.} \quad (1.39)$$

Folosind reprezentarea tabelară se poate arăta ușor că:

$$f_{11}(x_1, x_2) = x_1 \rightarrow x_2 = \bar{x}_1 + x_2, \quad (1.40)$$

$$f_{13}(x_1, x_2) = x_1 \leftarrow x_2 = x_1 + \bar{x}_2, \quad (1.41)$$

- funcțiile f_2 și f_4 sunt denumite *funcțiile interdicție* sau *funcțiile inhibare*:

$$f_2(x_1, x_2) = x_1 + x_2 = \bar{x}_1 x_2, \quad x_1 \text{ inhibă } x_2, \quad (1.42)$$

$$f_4(x_1, x_2) = x_2 + x_1 = x_1 \bar{x}_2, \quad x_2 \text{ inhibă } x_1, \text{ sau inhibare inversă.} \quad (1.43)$$

Din definirea funcțiilor booleene elementare de două argumente se poate desprinde concluzia că se pot genera funcții booleene noi prin:

- permutarea argumentelor;
- introducerea de funcții ca argumente ale noilor funcții.

1.2.3 Reprezentări ale funcțiilor booleene

Studiul funcțiilor booleene se face în multe cazuri pe reprezentările acestora. Există o mare diversitate de reprezentări ale funcțiilor booleene care pot fi grupate în reprezentări grafice (geometrice) și analitice [3,10,20,28]. Reprezentările din prima categorie sunt intuitive și se folosesc pentru studiul funcțiilor booleene cu un număr redus de argumente. Din această categorie fac parte reprezentările prin tabel de adevăr, diagrame Euler, Venn, Veitch sau Karnaugh, prin grafuri sau pe hipercub. A doua categorie asigură o reprezentare prin expresii algebrice sau sub formă de coduri. Reprezentările din această categorie permit studiul funcțiilor booleene cu un număr arbitrar de argumente, cu posibilitatea utilizării mijloacelor numerice de calcul. În continuare se va insista asupra celor mai folosite reprezentări ale funcțiilor booleene în scopuri tehnice [10,20,28].

1.2.3.1 Reprezentarea funcțiilor booleene prin tabel de adevăr

Acest mod de reprezentare corespunde reprezentării tabelare a funcțiilor booleene și de care s-a uzitat până aici. Tabelele denumite *de adevăr* sau *combinatoriale* conțin în partea stângă un număr de linii egal cu numărul combinațiilor posibile ale valorilor argumentelor, iar în partea dreaptă valorile funcției pentru fiecare combinație de valori ale argumentelor. Tabelul de adevăr este cea mai completă reprezentare a unei funcții booleene deoarece pentru fiecare combinație posibilă a valorilor argumentelor se indică valoarea funcției.

1.2.3.2 Reprezentarea funcțiilor booleene prin diagrame Karnaugh

Diagrama Karnaugh este tot o reprezentare tabelară dar, în raport cu tabelul de adevăr, este mai compactă datorită disponerii bidirecționale a valorilor argumentelor.

În cazul general al unei funcții booleene de n argumente, diagrama Karnaugh conține 2^p linii și 2^q coloane, astfel ca $p + q = n$. Dacă n este par, în mod obișnuit $p = q$, iar dacă n este impar, $q = p + 1$ (sau $p = q + 1$). Rezultă o diagramă cu $2^p \cdot 2^q = 2^n$ câmpuri.

puri (compartimente sau locații) în care se trec valorile funcției pentru combinațiile corespunzătoare ale valorilor argumentelor. Valorile argumentelor se indică la capetele liniilor și coloanelor diagramei.

Veitch a fost acela care a introdus acest mod de reprezentare a funcțiilor booleene. În reprezentarea propusă de acesta combinațiile valorilor argumentelor pe linii și coloane se dispun conform codului binar natural. Această dispunere conduce la dificultăți în folosirea diagramei pentru simplificarea funcțiilor booleene (pentru care a și fost concepută). Ceva mai târziu Karnaugh propune construirea acestor diagrame folosind codul Gray (binar reflectat) care fiind un cod continuu și ciclic asigură *adiacența* între câmpurile diagramei. Din motivele arătate, această reprezentare mai este cunoscută și ca *diagrama Veitch-Karnaugh*.

Notă. Într-un sistem de numerație în baza B , două cifre sunt adiacente dacă diferă cu o unitate *modulo* B . În sistemul de numerație binar două cifre vor fi adiacente dacă diferă cu cifra 1.

Mai multe amănunte despre modul concret de construcție și de utilizare al diagramei Karnaugh se vor prezenta la paragraful dedicat minimizării funcțiilor booleene prin metoda diagramelor Karnaugh (v. §2.3.1).

1.2.3.3 Reprezentarea analitică a funcțiilor booleene

Se consideră combinația valorilor argumentelor unei funcții booleene de n argumente $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n)$. Numărul „ i “ atașat n -uplului și definit prin relația:

$$i = \tilde{x}_1 2^{n-1} + \tilde{x}_2 2^{n-2} + \dots + \tilde{x}_n 2^0, \quad (1.44)$$

se numește *numărul combinației*.

De asemenea, se consideră funcția $P_i(x_1, x_2, \dots, x_n)$ care se definește în modul următor:

$$P_i = \begin{cases} 1, & \text{dacă numărul combinației este } „i“, \\ 0, & \text{în caz contrar.} \end{cases} \quad (1.45)$$

Această funcție se numește *constituuent al unității* sau *funcția caracteristică a unității*. Presupunând că se cunoaște expresia analitică a funcției P_i , a cărei determinare explicită se va face mai târziu, atunci se poate enunța următoarea teoremă:

Teoremă. Orice funcție booleană dată prin tabel de adevăr poate fi scrisă sub următoarea formă analitică:

$$f(x_1, x_2, \dots, x_n) = P_{i_1} + P_{i_2} + \dots + P_{i_k} = \sum_{i_j \in M_1} P_{i_j}, \quad (1.46)$$

unde M_1 este mulțimea numerelor combinațiilor valorilor argumentelor pentru care funcția ia valoarea 1.

Reprezentarea unei funcții booleene se poate face și sub altă formă analitică dacă se consideră funcția $S_i(x_1, x_2, \dots, x_n)$ definită în modul următor:

$$S_i = \begin{cases} 0, & \text{dacă numărul combinației este „}i\text{“}, \\ 1, & \text{în celelalte cazuri.} \end{cases} \quad (1.47)$$

Funcția S_i va fi denumită *funcția caracteristică a lui zero* sau *constituuentul lui zero*. Din relațiile de definiție (1.45) și (1.47) rezultă că $P_i(x_1, x_2, \dots, x_n) = \bar{S}_i(x_1, x_2, \dots, x_n)$.

Într-un mod similar se poate arăta că orice funcție booleană de n argumente poate fi reprezentată analitic și sub forma:

$$f(x_1, x_2, \dots, x_n) = S_{i_1} \cdot S_{i_2} \cdot \dots \cdot S_{i_k} = \prod_{i_j \in M_0} S_{i_j}, \quad (1.48)$$

unde M_0 este mulțimea numerelor combinațiilor valorilor argumentelor pentru care funcția ia valoarea 0.

Tab.1.6 Valorile variabilei booleene x^a pentru toate combinațiile posibile ale lui x_i și a_i .

x_i	a_i	$x_i^{a_i}$
0	0	$x_i^0 = \bar{x}_i = \bar{0} = 1$
0	1	$x_i^1 = x_i = 0$
1	0	$x_i^0 = \bar{x}_i = \bar{1} = 0$
1	1	$x_i^1 = x_i = 1$

Reprezentarea funcției booleene sub forma (1.46) se numește *reprezentare disjunctivă* iar sub forma (1.48), *reprezentare conjunctivă*.

Pentru a se stabili expresiile funcțiilor P_i și S_i se introduce următoarea notație pentru o variabilă booleană:

$$x^a = \begin{cases} x, & \text{dacă } a = 1, \\ \bar{x}, & \text{dacă } a = 0. \end{cases} \quad (1.49)$$

Se consideră expresia booleană:

$$x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}. \quad (1.50)$$

În această conjuncție de n argumente orice termen $x_i^{a_i}$ este 1 dacă și numai dacă $x_i = a_i$. Într-adevăr, considerând toate combinațiile posibile pentru x_i și a_i și ținând cont de (1.49) se obțin rezultatele concentrate din tabelul 1.6. Conform celor arătate rezultă că expresia (1.50) este 1 dacă și numai dacă $x_1 = a_1, x_2 = a_2, \dots, x_n = a_n$, fiind 0 pentru toate celelalte cazuri. Având în vedere definiția (1.45) a funcției P_i , rezultă:

$$P_i(x_1, x_2, \dots, x_n) = x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}, \quad (1.51)$$

cu condiția ca $i = a_1 2^{n-1} + a_2 2^{n-2} + \dots + a_n 2^0$. În aceste condiții, orice funcție booleană poate fi descrisă printr-o expresie analitică de forma:

$$f^{\text{FCD}}(x_1, x_2, \dots, x_n) = \sum_1 (x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}), \quad (1.52)$$

unde, prin \sum_1 s-a notat faptul că se consideră disjuncția termenilor conjunctivi (1.50)

pentru care funcția f ia valoarea 1.

Reprezentarea funcției booleene sub forma (1.52) se numește *forma canonica disjunctivă* (FCD) a funcției, iar termenii (1.50), *termeni canonici conjunctivi* (TCC) sau *termeni minimali (mintermi)*.

Teorema prezentată anterior și formula (1.52) permit stabilirea algoritmului trecerii de la tabelul de adevăr al unei funcții booleene la FCD.

Algoritmul 1.1. Forma canonica disjunctivă a unei funcții booleene dată prin tabel de adevăr (sau diagramă Karnaugh) se obține astfel:

1. Din tabelul de adevăr (diagrama Karnaugh) se consideră toate n -uplele pe care funcția le aplică în 1.

2. Se scriu termenii canonici conjunctivi care corespund acestor n -uple. În acești termeni argumentul x_i intră ca atare sau negat după cum în n -uplul considerat are valoarea 1 sau respectiv 0.

3. Termenii canonici conjunctivi obținuți se reunesc cu operația de disjuncție.

Exemplu. Se consideră funcția $f(x_1, x_2, x_3)$ dată prin tabelul de adevăr reprezentat în tabelul 1.7. Se consideră combinațiile valorilor argumentelor pentru care funcția are valoarea 1. Se scriu TCC corespunzători:

$$\bar{x}_1 \bar{x}_2 x_3, \bar{x}_1 x_2 \bar{x}_3, x_1 \bar{x}_2 \bar{x}_3, x_1 \bar{x}_2 x_3, x_1 x_2 x_3.$$

Tab.1.7 Tabelul de adevăr al funcției booleene f , din exemplu.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Reunind TCC prin operația de disjuncție se obține FCD:

$$f^{\text{FCD}}(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 x_3. \quad (1.53)$$

Corespunzător relației (1.48) și definiției funcțiilor caracteristice, se poate scrie:

$$f(x_1, x_2, \dots, x_n) = \prod_0 S_{i_j} = \prod_0 \overline{P_{i_j}} = \prod_0 \overline{x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}}. \quad (1.54)$$

Aplicând relațiile De Morgan, rezultă:

$$f(x_1, x_2, \dots, x_n) = \prod_0 (\overline{x_1^{a_1}} + \overline{x_2^{a_2}} + \dots + \overline{x_n^{a_n}}), \quad (1.55)$$

unde, prin \prod_0 s-a notat faptul că se consideră numai n -uplele pentru care funcția ia valoarea 0.

Din relația (1.55) rezultă:

$$S_i(x_1, x_2, \dots, x_n) = \overline{x_1^{a_1}} + \overline{x_2^{a_2}} + \dots + \overline{x_n^{a_n}}, \quad (1.56)$$

cu condiția ca $i = a_1 2^{n-1} + a_2 2^{n-2} + \dots + a_n 2^0$.

Rezultatele obținute în tabelul 1.6 confirmă condiția impusă și totodată permit verificarea identității $\overline{x_i^{a_i}} = x_i^{\overline{a_i}}$. În acest caz relația (1.55) se mai poate scrie:

$$f^{\text{FCC}}(x_1, x_2, \dots, x_n) = \prod_0 (x_1^{\overline{a_1}} + x_2^{\overline{a_2}} + \dots + x_n^{\overline{a_n}}). \quad (1.57)$$

Relațiile (1.55) și (1.57) sunt cunoscute sub denumirea de *formă canonica conjunctivă* (FCC), fiind *duala* formei canonice disjunctive. Termenii disjunctivi $(x_1^{\overline{a_1}} + x_2^{\overline{a_2}} + \dots + x_n^{\overline{a_n}})$ sunt numiți *termeni canonici disjunctivi* (TCD) sau *termeni maximali (maxtermi)*. și în acest caz relațiile (1.55) și (1.57) permit stabilirea algoritmului realizării FCC dacă se cunoaște tabelul funcției.

Algoritmul 1.2. Forma canonica conjunctivă a unei funcții booleene dată prin tabel de adevăr se obține în modul următor:

1. Din tabelul de adevăr al funcției se consideră toate n -uplele pe care funcția le aplică în 0.
2. Se scriu termenii canonici disjunctivi care corespund acestor n -uple. În expresia TCD argumentul x_i intră ca atare sau negat după cum în combinația considerată are valoarea 0 sau 1.
3. Termenii canonici disjunctivi obținuți la pasul 2 se reunesc prin semnul conjuncției.

Exemplu. Să se stabilească FCC pentru funcția dată prin tabelul din exemplul precedent. Conform algoritmului prezentat mai sus rezultă următorii TCD: $x_1 + x_2 + x_3$, $x_1 + \bar{x}_2 + \bar{x}_3$, $\bar{x}_1 + \bar{x}_2 + x_3$. Forma canonica conjunctivă rezultă imediat:

$$f^{\text{FCC}}(x_1, x_2, x_3) = (x_1 + x_2 + x_3)(x_1 + \bar{x}_2 + \bar{x}_3)(\bar{x}_1 + \bar{x}_2 + x_3). \quad (1.58)$$

Cele două forme canonice, disjunctivă și conjunctivă, sunt unice pentru o funcție booleană complet definită. Alegerea unei forme sau a celeilalte depinde de criteriul care stă la baza dezvoltării funcției în formă analitică. Dacă acest criteriu este cel al economicității atunci alegerea FCD sau FCC depinde de forma tabelului funcției respective. Astfel, dacă majoritatea valorilor funcției sunt zero este de preferat FCD; în caz contrar o economie mai mare o asigură FCC.

1.2.3.4 Reprezentarea funcțiilor booleene prin scheme logice (logigrame)

Schema logică (logograma, rețeaua de comutare sau, mai rar, schema de operatori) este o reprezentare grafică a funcției booleene obținută prin adoptarea unor semne (simboluri) convenționale pentru operatorii logici. În general, numim *operator n*-ar o funcție booleană de n variabile $y = f(x_1, x_2, \dots, x_n)$ [3,20]. Multimea operatorilor utilizați în practică este determinată de posibilitatea realizării acestora prin elemente fizice. Schema logică indică de fapt topologia unui circuit logic care materializează o funcție booleană. Ca urmare, simbolurile grafice adoptate pentru operatorii logici constituie o reprezentare a circuitelor logice care materializează funcțiile logice elementare. Una dintre posi-

bilitățile de realizare fizică a acestora o reprezintă *porțile logice* (v. §2.3.2). În tabelul 1.8 sunt indicate cele mai utilizate simboluri grafice pentru principalele funcții elementare de două argumente, în conformitate cu trei sisteme de norme:

- normele Comisiei Electrotehnice Internaționale (CEI)², care sunt recunoscute internațional;
- normele americane MIL-STD-806 B (MIL)^{3,4}, care sunt foarte frecvent utilizate în practică (adoptate și în literatura de specialitate românească);
- normele germane DIN 40700, ediția 1963.

Folosind aceste simboluri grafice expresiile algebrice ale funcțiilor booleene pot fi reprezentate sub formă de scheme logice.

Exemplu. Să se reprezinte prin schemă logică funcția $f^{FCD}(x_1, x_2, x_3) = x_1x_2x_3 + x_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2\bar{x}_3$. Având în vedere tabelul 1.8, în figura 1.1 este prezentată logograma funcției date. Schema logică indică și nivelele logice compuse din elemente fizice care operează simultan. De câte ori este posibil, elementele aceluiași nivel logic se reprezintă pe aceeași coloană.

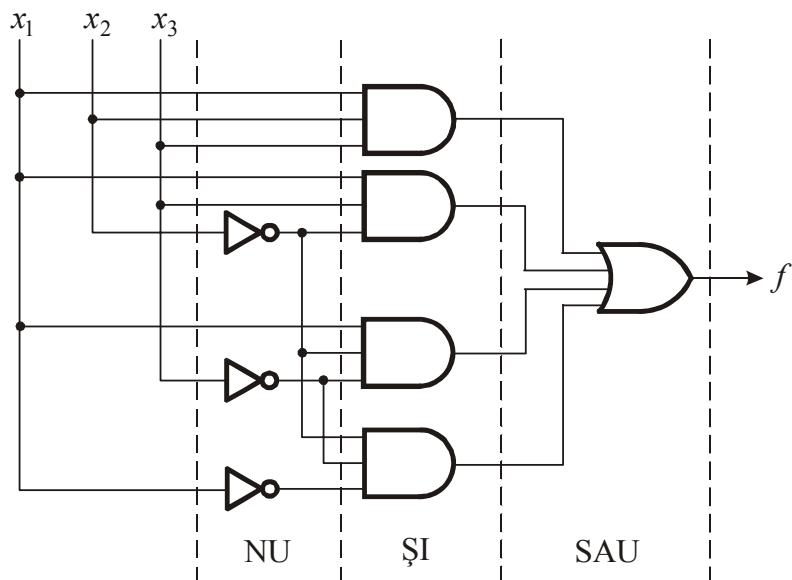


Fig.1.1 Schema logică și nivelele logice pentru funcția f din exemplu.

1.2.3.5 Reprezentarea funcțiilor booleene prin diagrame de timp

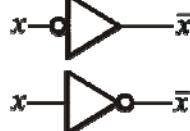
Diagrama de timp reprezintă grafic o funcție booleană prin forma semnalelor corespunzătoare argumentelor și funcției. Cifrele binare 0 și 1 se atașează semnalelor de nivel coborât și respectiv ridicat, astfel încât să existe o diferențiere netă a acestora. Reprezentarea prin diagrame de timp este deosebit de utilă în studiul sistemelor secvențiale.

² COMMISSION ELECTROTECHNIQUE INTERNATIONALE, *Publication 617-12, Symboles graphiques recommandés, cinquième partie: Opérateurs logiques binaires*, Bureau Central de la Commission Electrotechnique Internationale, Genève, 1972.

³ Military Standard, *Graphic Symbols for Logic Diagrams*, MIL-STD-806 B, Department of Defense, Washington, 1962.

⁴ ANSI, IEEE, *American National Standard, graphic symbols for logic diagrams (two-state devices)*, IEEE Std 91-1984, ANSI Y 32.14-1984, Institute of Electric and Electronics Engineers, New York, 1984.

Tab.1.8 Principalele funcții elementare de două argumente și simbolurile lor grafice.

Denumirea funcției	Funcția	Denumirea circuitului logic	Reprezentarea simbolică		
			Norme CEI	Norme MIL	Norme DIN
Negația	$f = \bar{x}$	INVERSOR (INVERTER)	 $x \rightarrow \bar{x}$	 $x \rightarrow \bar{x}$	 $x \rightarrow \bar{x}$
Conjuncția	$f = x_1 x_2$	ȘI (AND)	 $x_1 \quad \& \quad x_2 \rightarrow x_1 x_2$	 $x_1 \quad x_2 \rightarrow x_1 x_2$	 $x_1 \quad x_2 \rightarrow x_1 x_2$
Disjuncția	$f = x_1 + x_2$	SAU (OR)	 $x_1 \quad \geq 1 \quad x_2 \rightarrow x_1 + x_2$	 $x_1 \quad x_2 \rightarrow x_1 + x_2$	 $x_1 \quad x_2 \rightarrow x_1 + x_2$
Sheffer	$f = x_1 \uparrow x_2 =$ $= \bar{x}_1 \bar{x}_2 =$ $= \bar{x}_1 + \bar{x}_2$	ȘI-NU (NAND)	 $x_1 \quad \& \quad x_2 \rightarrow \bar{x}_1 \bar{x}_2$	 $x_1 \quad x_2 \rightarrow \bar{x}_1 \bar{x}_2$	 $x_1 \quad x_2 \rightarrow \bar{x}_1 \bar{x}_2$
Pierce (Webb)	$f = x_1 \downarrow x_2 =$ $= \bar{x}_1 + \bar{x}_2 =$ $= \bar{x}_1 \cdot \bar{x}_2$	SAU-NU (NOR)	 $x_1 \quad \geq 1 \quad x_2 \rightarrow \bar{x}_1 + \bar{x}_2$	 $x_1 \quad x_2 \rightarrow \bar{x}_1 + \bar{x}_2$	 $x_1 \quad x_2 \rightarrow \bar{x}_1 + \bar{x}_2$
Suma modulo 2	$f = x_1 \oplus x_2 =$ $= \bar{x}_1 x_2 + x_1 \bar{x}_2$	SAU EXCLUSIV (XOR)	 $x_1 \quad \oplus \quad x_2 \rightarrow x_1 \oplus x_2$	 $x_1 \quad x_2 \rightarrow x_1 \oplus x_2$	 $x_1 \quad x_2 \rightarrow x_1 \oplus x_2$

ale în a căror evoluție intervine și timpul. De asemenea, folosind această reprezentare se pot studia fenomenele tranzitorii de comutare și fenomenele de hazard datorate funcționării neideale a elementelor care materializează variabile sau funcții booleene.

Exemplu. Să se reprezinte prin diagramă de timp funcția logică $f(x_1, x_2) = x_1 \uparrow x_2$, cunoscând evoluția în timp a semnalelor atașate argumentelor (v. fig. 1.2,a). Având în vedere tabelul de definiție al funcției NUMAI, în figura 1.2,b se dă reprezentarea prin diagramă de timp a funcției considerate, pentru evoluția dată a argumentelor.

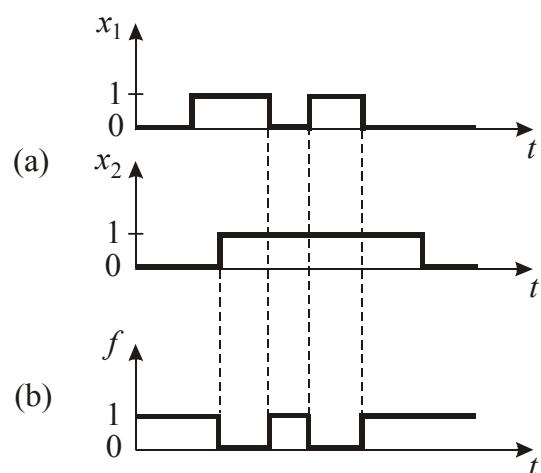


Fig.1.2 Diagrama de timp pentru funcția din exemplu.

1.2.4 Sisteme complete de funcții

Așa cum s-a mai precizat (v. §1.2.2) funcțiile booleene se pot forma prin superpoziția funcțiilor elementare [3,10,20]. Problema formării funcțiilor booleene poate fi privită și astfel: să se stabilească un sistem de funcții care să poată reprezenta orice funcție booleană și în plus, dacă acest sistem există, să conțină un număr cât mai mic de funcții. Evident, problema formării funcțiilor booleene pusă în acest mod are o mare importanță pentru sinteza schemelor logice.

Definiție. Sistemul de funcții booleene (f_1, f_2, \dots, f_k) se numește *complet* (bază respectiv *sistem funcțional complet de bază*) în clasă \mathcal{R} , dacă orice funcție φ aparținând lui \mathcal{R} poate fi reprezentată prin superpoziția funcțiilor (f_1, f_2, \dots, f_k) .

În calitate de clasă \mathcal{R} se poate considera clasa tuturor funcțiilor booleene care depind de n argumente B_2^n . Conform celor arătate în prima parte a subcapitolului 1.2, numărul total al funcțiilor care depind de n argumente este egal cu 2^{2^n} . Deci, în clasa B_2^n există un sistem complet compus din cele 2^{2^n} funcții ale acestei clase, dar acest sistem este trivial. În §1.2.3 s-a stabilit că orice funcție booleană poate fi reprezentată sub FCD sau FCC. Rezultă că sistemul de funcții compus din conjuncție, disjuncție și negare constituie un sistem complet de funcții. Se pune problema dacă sistemul complet de funcții ($,+^{\wedge}$, $,\cdot^{\wedge}$, $,-^{\wedge}$) conține numărul strict necesar pentru reprezentarea oricărei funcții booleene. Pentru ca un sistem complet să fie minim (bază minimală) este necesar să satisfacă următoarea definiție:

Definiție. Un sistem complet de funcții (f_1, f_2, \dots, f_k) este *minim* (bază minimală) dacă înălțurând oricare dintre funcțiile aparținând sistemului acesta devine incomplet.

Compleitudinea sistemului de funcții ($,+^{\wedge}$, $,\cdot^{\wedge}$, $,-^{\wedge}$) permite demonstrarea completitudinii oricărui alt sistem de funcții arbitrar format. Pentru aceasta este suficient să se arate că funcțiile sistemului ales pot reprezenta funcțiile sistemului ($,+^{\wedge}$, $,\cdot^{\wedge}$, $,-^{\wedge}$).

Corespunzător celor arătate mai sus, interes practic prezintă evidentierea următoare-

relor teoreme:

Teoremă. Sistemul de funcții format din conjuncție și negație („·“, „–“), este un sistem complet în clasa B_2^n .

Teoremă. Sistemul de funcții format din disjuncție și negație („+“, „–“), este un sistem complet în clasa B_2^n .

Dacă conform celor două teoreme prezentate, sistemele („·“, „–“) și („+“, „–“) sunt complete, în același timp formează baze minime în raport cu sistemul („+“, „·“, „–“).

Teoremă. Funcția lui Sheffer formează în clasa B_2^n un sistem complet.

Teoremă. Funcția lui Pierce formează în clasa B_2^n un sistem complet.

Ultimele două teoreme prezentate sunt deosebit de importante pentru aplicarea practică la sinteza circuitelor logice, permitând folosirea unui singur tip de circuit pentru materializarea oricărei funcții booleene. În acest context devine importantă trecerea de la FCD și FCC la forme cu funcții Pierce sau Sheffer. Această trecere este cunoscută și sub denumirea de *implementare*.

1.2.4.1 Implementarea FCD și FCC cu funcții Pierce și Sheffer

Prin analogie cu definirea funcțiilor elementare Pierce și Sheffer pentru două argumente, se vor defini aceste funcții pentru n argumente folosind tabelul de adevară:

Tab.1.9 Generalizarea funcțiilor elementare Pierce și Sheffer.

x_1	x_2	x_3	...	x_{n-1}	x_n	Pierce	Sheffer
0	0	0	...	0	0	1	1
0	0	0	...	0	1	0	1
0	0	0	...	1	0	0	1
...
1	1	1	...	0	1	0	1
1	1	1	...	1	0	0	1
1	1	1	...	1	1	0	0

Conform acestui tabel, rezultă:

- funcția Pierce de n argumente: $P_n = x_1 \downarrow x_2 \downarrow \dots \downarrow x_n = \overline{x_1 + x_2 + \dots + x_n}$;
- funcția Sheffer de n argumente: $S_n = x_1 \uparrow x_2 \uparrow \dots \uparrow x_n = \overline{x_1 \cdot x_2 \cdot \dots \cdot x_n}$.

Pentru $n = 2$ din tabelul de adevară se obțin funcțiile Pierce și Sheffer de două argumente, iar pentru $n = 1$ ambele funcții se transformă în funcția negație: $P_1 = S_1 = \bar{x}$.

Exprimarea prin funcții Pierce a unei funcții booleene se obține considerând FCC a acesteia și negând de două ori termenii disjunctivi:

$$f^{\text{FCC}}(x_1, x_2, \dots, x_n) = \prod_0 (x_1^{\bar{a}_1} + x_2^{\bar{a}_2} + \dots + x_n^{\bar{a}_n}) = \prod_0 \overline{(x_1^{\bar{a}_1} + x_2^{\bar{a}_2} + \dots + x_n^{\bar{a}_n})}. \quad (1.59)$$

Aplicând relațiile De Morgan și ținând cont de definirea funcției Pierce de n argumente, se obține:

$$f(x_1, x_2, \dots, x_n) = \prod_0^{\overline{\overline{x_1^{\bar{a}_1} x_2^{\bar{a}_2} \dots x_n^{\bar{a}_n}}}} = \downarrow_0(x_1^{\bar{a}_1} \downarrow x_2^{\bar{a}_2} \downarrow \dots \downarrow x_n^{\bar{a}_n}), \quad (1.60)$$

unde prin \downarrow_0 s-a notat faptul că se consideră numai n -uplele pe care funcția le aplică în zero.

Concluzie. Pentru implementarea unei funcții booleene cu funcții Pierce se pleacă de la FCC și se înlocuiesc operațiile „+“ și „.“ cu operația \downarrow .

Având în vedere cele arătate mai sus se poate stabili algoritmul implementării oricărei funcții booleene cu funcții NICI plecând de la tabelul de adevăr sau de la diagramea Karnaugh.

Algoritm 1.3. Implementarea unei funcții booleene de n argumente cu funcții NICI plecând de la tabelul de adevăr se obține astfel:

1. Se consideră n -uplele pe care funcția le aplică în 0.

2. Fiecarui n -uplu considerat îi corespunde un termen implementat cu funcții NICI. În acești termeni fiecare argument intră ca atare sau negat după cum în combinația respectivă are valoarea 0 sau respectiv 1.

3. Toți termenii obținuți la pasul 2 se reunesc prin simbolul funcției NICI.

4. Excepție de la punctul 3 face situația când funcția are un singur n -uplu aplicat în 0. În acest caz termenul respectiv se neagă. Explicația rezultă din faptul că $x \downarrow x = \bar{x}$.

Tab.1.10 Tabelul de adevăr al funcției booleene f din exemplu.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Exemplu. Să se implementeze cu funcții NICI funcția booleană dată prin tabelul de adevăr alăturat.

Conform algoritmului 1.3, se obține imediat:

$$f(x_1, x_2, x_3) = (x_1 \downarrow x_2 \downarrow x_3) \downarrow (x_1 \downarrow \bar{x}_2 \downarrow \bar{x}_3) \downarrow (\bar{x}_1 \downarrow x_2 \downarrow \bar{x}_3). \quad (1.61)$$

Dacă funcția ar aplica în 0 numai combinația (1,0,1) ar rezulta:

$$f(x_1, x_2, x_3) = \overline{\bar{x}_1 \downarrow x_2 \downarrow \bar{x}_3}. \quad (1.62)$$

Exprimarea prin funcții Sheffer a unei funcții booleene se obține considerând FCD și negând de două ori termenii conjunctivi:

$$f^{\text{FCD}}(x_1, x_2, \dots, x_n) = \sum_1^{\overline{\overline{(x_1^{a_1} x_2^{a_2} \dots x_n^{a_n})}}} = \sum_1^{\overline{\overline{(x_1^{a_1} x_2^{a_2} \dots x_n^{a_n})}}}. \quad (1.63)$$

Aplicând relațiile De Morgan și ținând cont de relația de definiție a funcției Sheffer de n argumente rezultă:

$$f(x_1, x_2, \dots, x_n) = \sum_1 \overline{(x_1^{a_1} \uparrow x_2^{a_2} \uparrow \dots \uparrow x_n^{a_n})} = \uparrow_1 (x_1^{a_1} \uparrow x_2^{a_2} \uparrow \dots \uparrow x_n^{a_n}). \quad (1.64)$$

unde, prin \uparrow_1 s-a notat faptul că se consideră numai n -uplele aplicate de funcție în 1.

Concluzie. Pentru implementarea cu funcții Sheffer a unei funcții booleene se pleacă de la FCD în care se înlocuiesc simbolurile „+“ și „·“ cu simbolul \uparrow .

Având în vedere (1.64) și algoritmul 1.2 se poate stabili și în acest caz un algoritm pentru implementarea cu funcții NUMAI a unei funcții booleene plecând de la tabelul de adevăr sau diagramă Karnaugh.

Algoritm 1.4. Implementarea unei funcții booleene de n argumente cu funcții NUMAI plecând de la tabelul de adevăr se obține astfel:

1. Se consideră toate n -uplele pe care funcția le aplică în 1.
2. Fiecare n -uplu îi corespunde un termen implementat cu funcții NUMAI, în care fiecare variabilă se ia ca atare sau negată după cum în combinația considerată are valoarea 1 sau respectiv 0.
3. Termenii obținuți la pasul 2 se reunesc prin simbolul funcției NUMAI.
4. Excepție de la punctul 3 face cazul când funcția aplică în 1 numai un n -uplu; termenul implementat cu funcții NUMAI se neagă deoarece $x \uparrow x = \bar{x}$.

Exemplu. Să se implementeze cu funcții NUMAI funcția de la exemplul precedent.

Considerând combinațiile valorilor argumentelor pe care funcția le aplică în 1 și ținând cont de algoritm 1.4, rezultă:

$$\begin{aligned} f(x_1, x_2, x_3) &= (\bar{x}_1 \uparrow \bar{x}_2 \uparrow x_3) \uparrow (\bar{x}_1 \uparrow x_2 \uparrow \bar{x}_3) \uparrow (x_1 \uparrow \bar{x}_2 \uparrow \bar{x}_3) \uparrow \\ &\quad \uparrow (x_1 \uparrow x_2 \uparrow \bar{x}_3) \uparrow (x_1 \uparrow x_2 \uparrow x_3). \end{aligned} \quad (1.65)$$

Dacă însă funcția ar aplica în 1 numai combinația (0,1,0) atunci:

$$f(x_1, x_2, x_3) = \overline{\bar{x}_1 \uparrow x_2 \uparrow \bar{x}_3}. \quad (1.66)$$

2

CIRCUITE LOGICE COMBINATORIALE

2.1 NOTIUNI INTRODUCTIVE

Un circuit logic combinațional (CLC) este un circuit de comutare care se caracterizează prin aceea că starea ieșirilor sale la un moment dat depinde numai de starea intrărilor sale la momentul considerat. Legătura între starea intrărilor și starea ieșirilor circuitului este dată de funcțiile de transfer ale acestuia, denumite în acest caz funcții de comutare.

Studiul CLC-urilor se face, aşa cum s-a mai precizat (v. §1.2.3), folosind un model al acestora, *schema logică* sau *rețeaua de comutare*, în care se face abstracție de caracteristicile constructive ale elementelor care compun circuitul, considerându-se doar proprietățile lor funcționale. Din acest motiv, rețelele de comutare au un caracter general, permitând studiului unor clase mari de circuite de comutare.

Un CLC cu n intrări și m ieșiri, reprezentat schematic prin schema-bloc din figura 2.1, se definește în felul următor:

Definiție. Un circuit de comutare având intrările x_1, x_2, \dots, x_n și ieșirile z_1, z_2, \dots, z_m se numește *circuit logic combinațional*, iar modelul său grafic *rețea de comutare combinațională* dacă ieșirile sale pot fi exprimate ca m funcții de comutare de variabilele de intrare x_1, x_2, \dots, x_n :

$$\begin{aligned}
 z_1 &= f_1(x_1, x_2, \dots, x_n), \\
 z_2 &= f_2(x_1, x_2, \dots, x_n), \\
 &\dots \\
 z_m &= f_m(x_1, x_2, \dots, x_n),
 \end{aligned} \tag{2.1}$$



Fig.2.1 Modelul general al unui CLC.

unde, funcțiile f_1, f_2, \dots, f_m sunt denumite și *funcții proprii* ale circuitului.

Dependența de timp nu se marchează explicit în relațiile (2.1), întrucât se subînțelege că este vorba de variabilele considerate la același moment de timp.

CLC-ul constituie un *caz particular de automat finit* [20,28], care poate fi reprezentat formal prin tripletul:

$$A = A(X, Z, f), \quad (2.2)$$

compus din două mulțimi nevide X – mulțimea semnalelor de intrare și Z – mulțimea semnalelor de ieșire, ale automatului A și din funcția f definită pe aceste mulțimi:

$$f : X \rightarrow Z. \quad (2.3)$$

Independența față de timp a relațiilor (2.1) ÷ (2.3) presupune, așa cum s-a mai precizat, că la modificarea simultană a variabilelor de intrare ale CLC-ului are loc modificarea instantaneă și simultană a variabilelor de ieșire. Această ipoteză nu este valabilă în cazul CLC-urilor reale datorită întârzierilor care au loc la propagarea variației vectorului de intrare înspre ieșiri. Pe durata procesului tranzitoriu de stabilire a variabilelor de ieșire, vectorul ieșirilor poate lua valori intermediare diferite de cele finale, determinate de relațiile (2.1). Ordinea în care se stabilesc valorile finale ale variabilelor de ieșire depinde de diferențele dintre timpii de propagare corespunzători circuitelor logice elementare utilizate pentru implementarea CLC-urilor. Acest fenomen este denumit *hazard combinatorial*, iar influența lui asupra funcționării sistemului numeric în care se încadrează CLC-ul trebuie avută în vedere la proiectarea acestuia.

Modalitatea adoptată pentru *realizarea fizică a CLC-ului depinde de complexitatea circuitului, de numărul variabilelor de intrare și ieșire*, precum și de *considerante de ordin economic*. Astfel, implementarea CLC-ului se poate face utilizând circuite integrate pe scară mică, medie sau largă. În paragrafele următoare se vor prezenta aceste posibilități de realizare fizică, precum și câteva circuite combinatoriale tipice, integrate pe scară medie sau largă.

2.2 ANALIZA CIRCUITELOR LOGICE COMBINATORIALE

În cadrul problemelor de analiză se cunoaște structura circuitului și se cere să se stabilească valorile posibile la ieșiri pentru toate combinațiile posibile ale valorilor semnalelor de la intrări. Problemele de analiză se soluționează căutând expresiile funcțiilor booleene corespunzătoare semnalelor de ieșire, ca apoi să se poată stabili valorile lor pentru diferențele combinații ale valorilor argumentelor. În general, funcțiile proprii care se stabilesc plecând de la configurația circuitului se obțin sub formă normală¹. Pentru a putea stabili toate valorile acestor funcții pentru toate combinațiile valorilor argumentelor este necesară evidențierea termenilor canonici. Rezultă că din punct de vedere matematic problemele de analiză conduc la necesitatea dezvoltării funcțiilor booleene sub forme canonice, pentru a le pune în evidență constituenții. Din cele expuse mai sus se pot formula etapele care trebuie parcursă în rezolvarea problemelor de analiză:

¹ Forma normală va fi definită în §2.3.1.

1. Cunoscând schema logică a circuitului se determină din “aproape în aproape” funcțiile corespunzătoare ieșirilor.
2. Se dezvoltă funcțiile proprii ale circuitului sub formă canonica.
3. Folosind tabelul de adevăr, sau diagrama Karnaugh, se stabilesc valorile funcțiilor pentru toate combinațiile posibile ale argumentelor.
4. O etapă opțională, dar de multe ori interesantă, din punct de vedere practic, este de a se stabili expresiile minime ale formelor canonice ale funcțiilor în vederea unor posibile simplificări. Această etapă vizează posibilitatea realizării unui circuit echivalent cu primul, însă mai economic. Dar, în acest caz este necesar să se cunoască dacă nu există și alte criterii de realizare mai puternice decât cel economic (fiabilitate, funcționare fără hazard).

2.2.1 Analiza schemelor logice combinatoriale realizate cu elemente logice řI, SAU, NU

Rețeaua de comutare combinatorială sau schema logică combinatorială reprezintă, așa cum s-a mai precizat, modelul grafic al CLC-ului realizat cu operatori logici. În cazul general, o rețea de comutare cu elemente logice are n intrări, x_1, x_2, \dots, x_n și m ieșiri, z_1, z_2, \dots, z_m . Intrările x_1, x_2, \dots, x_n se aplică unor elemente logice ale căror ieșiri pot fi ieșiri ale rețelei sau intrări pentru alte elemente logice ale acesteia. În figura 2.2 se prezintă o rețea de comutare cu elemente logice, cu 10 intrări și 3 ieșiri.

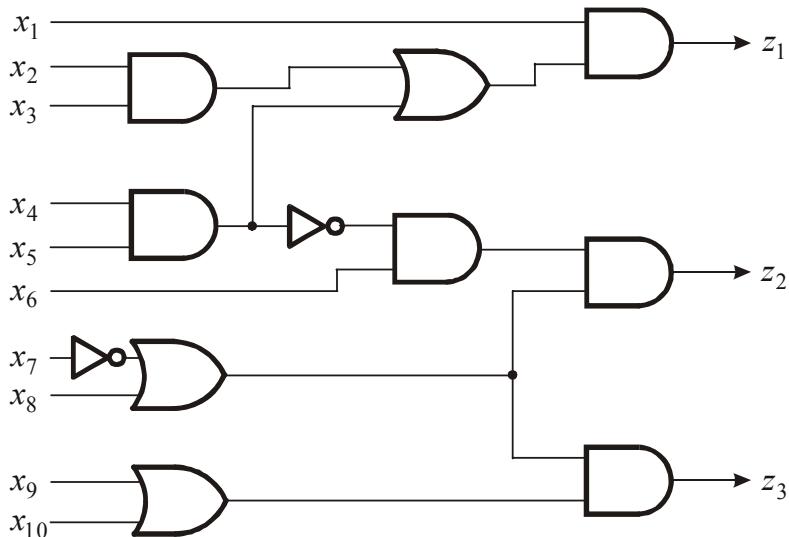


Fig.2.2 Rețea de comutare cu elemente logice řI, SAU, NU.

Fiecare element logic din rețea corespunde unei porți logice din circuitul de comutare modelat. În schemele logice nu se admite legarea ieșirilor elementelor logice decât prin intermediul altor elemente logice. Atunci când circuitul de comutare modelat conține porți logice care au proprietatea de realizare a funcțiilor logice cablate, se reprezintă simbolic în rețea elementul logic prin care sunt legate din punct de vedere funcțional ieșirile porțiilor respective.

Semnalele aplicate la intrările unui CLC cu porți parcurg, în general, mai multe porți până se obțin semnalele de ieșire. Acest lucru se reflectă în rețea prin numărul

elementelor logice interpuse între intrările și ieșirile rețelei. Maximumul numărului de elemente logice aflate între intrările și ieșirile unei rețele logice dă numărul de nivele logice al rețelei. Numerotarea acestora se face, în mod convențional, de la ieșire spre intrare (v. fig. 2.3,a). În rețelele de comutare combinaționale sunt admise legături inverse, adică legarea ieșirii unui element logic la intrările elementelor logice precedente acestuia (v. fig. 2.3,b), cu condiția respectării definiției CLC-ului. Trebuie menționat însă că orice rețea de elemente logice fără legături inverse este combinațională, în schimb numai anumite rețele cu legături inverse satisfac această condiție. Rețelele cu elemente logice fără legătură inversă se mai numesc *grafuri booleene*.

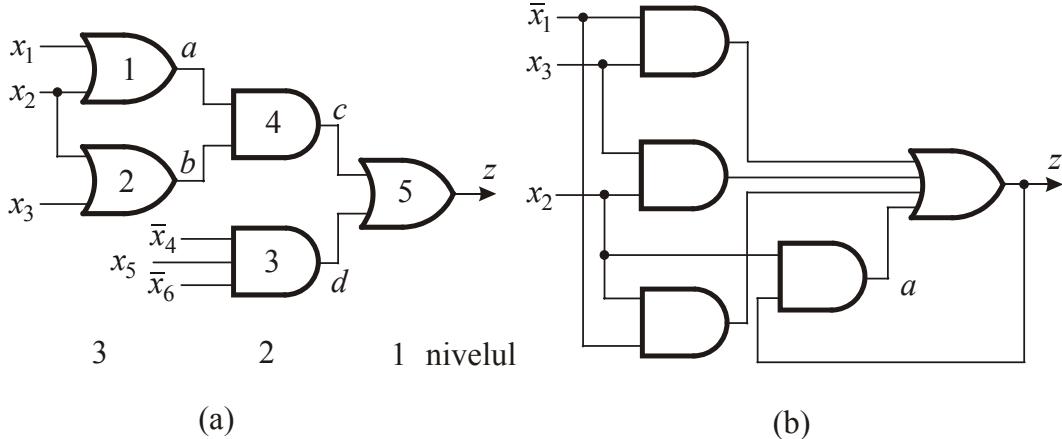


Fig.2.3 Rețele de comutare fără legătură inversă (a) și cu legătură inversă (b).

Existența unei legături inverse într-o rețea de comutare cu elemente logice se poate determina folosind următoarea regulă de numerotare a elementelor [22]:

Regulă.

1. Elementele rețelei ale căror intrări fac parte din mulțimea intrărilor rețelei $X = \{x_1, x_2, \dots, x_n\}$, se numerotează, într-o ordine arbitrară, cu numerele 1 până la k , unde k este numărul elementelor ce îndeplinesc această condiție.

2. Elementele rețelei ale căror intrări sunt fie intrări din mulțimea X , fie ieșiri ale elementelor numerotate, la punctul 1, respectiv 2, se numerotează în continuare cu numerele $k+1$ până la m , unde $m-k$ este numărul elementelor ce îndeplinesc această condiție.

3. Dacă procedând în acest fel s-au putut numera toate elementele rețelei, adică m este numărul de elemente logice din rețea, rezultă că rețeaua nu are legături inverse. În caz contrar în rețea există cel puțin o legătură inversă.

Prin procedura de mai sus se face de fapt o ordonare parțială a elementelor logice din rețea. Această ordonare nu este posibilă decât atunci când un anumit element nu mai poate primi număr de ordine, deoarece unele dintre intrările sale provin de la elemente nenumerotate care succed elementul dat, ceea ce înseamnă că există o legătură inversă.

Aplicând regula de mai sus pentru rețeaua din figura 2.3,b se constată că nu este posibilă o ordonare parțială a tuturor elementelor rețelei, prin urmare rețeaua are o legătură inversă.

Pentru rețelele la care se poate face o ordonare parțială a elementelor, deci care nu au legături inverse, se poate scrie expresia ieșirii fiecărui element logic, în ordinea numerotării acestora, ca o funcție de comutare de variabilele de intrare ale rețelei, de

unde rezultă că și în expresia ieșirii intră numai variabilele de intrare ale rețelei. Funcția de comutare a unei asemenea rețele depinde deci numai de variabilele de intrare, prin urmare rețeaua este combinațională. Pentru rețeaua logică din figura 2.3,a, se obține:

$$a = x_1 + x_2, \quad (2.4 \text{ a})$$

$$b = x_2 + x_3, \quad (2.4 \text{ b})$$

$$d = \bar{x}_4 x_5 \bar{x}_6, \quad (2.4 \text{ c})$$

$$c = a \cdot b = (x_1 + x_2)(x_2 + x_3), \quad (2.4 \text{ d})$$

$$z = c + d = (x_1 + x_2)(x_2 + x_3) + \bar{x}_4 x_5 \bar{x}_6. \quad (2.4 \text{ e})$$

Observație. Pentru ca expunerea să nu devină dificil de urmărit nu se vor exemplifica toate punctele algoritmului.

Din relația (2.4 e) rezultă că z este o funcție numai de variabilele de intrare ale rețelei, deci rețeaua din figura 2.3,a este o rețea combinațională.

Pentru rețelele care au legături inverse expresia ieșirii se poate scrie numai dacă se introduc variabile de intrare secundare datorate legăturilor inverse. Pentru ca o asemenea rețea să fie combinațională trebuie ca funcția de comutare realizată de rețea să nu depindă decât de variabilele de intrare ale rețelei adică variabilele de intrare secundare să fie neesențiale. Faptul că variabilele secundare sunt neesențiale se poate vedea numai în tabelul de adevăr al funcției considerate. Din acest motiv, pentru a se dovedi că o rețea de comutare cu elemente logice și legături inverse este combinațională se întocmește un tabel de adevăr în care se dau toate valorile posibile, atât pentru variabilele de intrare principale, cât și pentru cele secundare (de exemplu, variabila a din figura 2.3,b). Dacă din acest tabel rezultă că valoarea funcției de comutare realizată de rețea nu depinde de valorile variabilelor secundare rețeaua este combinațională. Din cele prezentate rezultă că oricărei rețele de comutare combinaționale cu legături inverse îi corespunde cel puțin o rețea de comutare combinațională fără legătură inversă.

Având în vedere faptul că rețelele de comutare cu elemente logice, fără legătură inversă sunt reprezentative pentru rețele combinaționale cu elemente logice, în cele ce urmează se vor trata numai acestea.

Rețelele de comutare combinaționale cu elemente logice cu o singură ieșire, la care fiecare dintre intrările rețelei se aplică la un singur element logic, iar ieșirea unui element logic poate fi aplicată ca intrare la un singur element logic se numește *arbore boolean*. Arboarele boolean este un caz particular al grafului boolean. Un exemplu de arbore boolean este dat în figura 2.4.

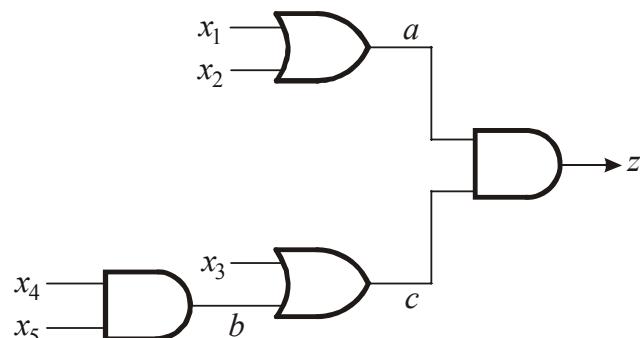


Fig.2.4 Rețea combinațională sub formă de arbore boolean.

2.2.2 Analiza schemelor logice combinaționale realizate cu elemente logice ȘI-NU și SAU-NU

Pentru a se putea scrie mai ușor expresia ieșirii unei rețele de comutare realizate cu elemente logice ȘI-NU și SAU-NU, trebuie observat că o variabilă de intrarea apare complementată în expresia ieșirii unei rețele dacă a parcurs un număr impar de elemente de inversare (negare), respectiv necomplementată dacă a parcurs un număr par de asemenea elemente. Numerotând nivelele de inversare de la ieșire spre intrare și ținând cont de teoremele lui De Morgan și de observația de mai sus rezultă următoarea regulă pentru exprimarea operației realizate de un element logic ȘI-NU, respectiv SAU-NU aflat într-un anumit nivel [22]:

Regulă. Un element logic ȘI-NU (SAU-NU) realizează operația SAU (ȘI) asupra variabilelor de intrare complementate dacă se află într-un nivel de inversare impar, respectiv operația ȘI (SAU), asupra variabilelor de intrare necomplementate dacă se află într-un nivel de inversare par.

Pe baza acestei reguli se poate formula următorul algoritm de analiză a unei rețele de comutare realizate cu elemente logice ȘI-NU și SAU-NU:

Algoritmul 2.1.

1. Se consideră elementul logic de la care se obține ieșirea relației ca fiind în primul nivel de inversare, elementele precedente în al doilea ș.a.m.d.
2. Se consideră că toate elementele logice ȘI-NU din nivelele impare realizează operația logică SAU asupra intrărilor.
3. Se consideră că toate elementele logice ȘI-NU din nivelele pare realizează operația logică ȘI asupra intrărilor.
4. Se consideră că toate elementele logice SAU-NU din nivelele impare realizează operația logică ȘI asupra intrărilor.
5. Se consideră că toate elementele logice SAU-NU din nivelele pare realizează operația logică SAU asupra intrărilor.
6. Toate variabilele de intrare care se aplică elementelor logice din nivelele impare se iau complementate.
7. Toate variabilele de intrare care se aplică elementelor logice din nivelele pare se iau necomplementate.
8. Aplicând regulile 1÷7 pentru toate elementele logice din rețea, respectiv pentru toate variabilele de intrare se obține expresia booleană a ieșirii rețelei.

Pentru rețelele cu mai multe ieșiri se aplică algoritmul 2.1 pentru fiecare ieșire. În acest caz, un anumit element logic poate fi într-un nivel par, raportat la una dintre ieșiri și într-un nivel impar, raportat la o altă ieșire.

Exemplu. Se consideră rețeaua de comutare din figura 2.5. Folosindu-se algoritmul 2.1 să se facă analiza acestei rețele.

Aplicând algoritmul 2.1 pentru obținerea expresiei ieșirii z_1 a rețelei din figura 2.5, rezultă:

1. Poarta 5 (ȘI-NU) este în nivel impar (1); se consideră că realizează operația SAU.
2. Poarta 4 (ȘI-NU) este în nivel par (2); se consideră că realizează operația ȘI, iar variabila de intrare x_6 apare în expresia ieșirii z_1 sub formă necomplementată.
3. Poarta 3 (SAU-NU) este în nivel par (2); se consideră că realizează operația SAU, iar variabila de intrare x_3 apare în expresia ieșirii z_1 sub formă necomplementată.

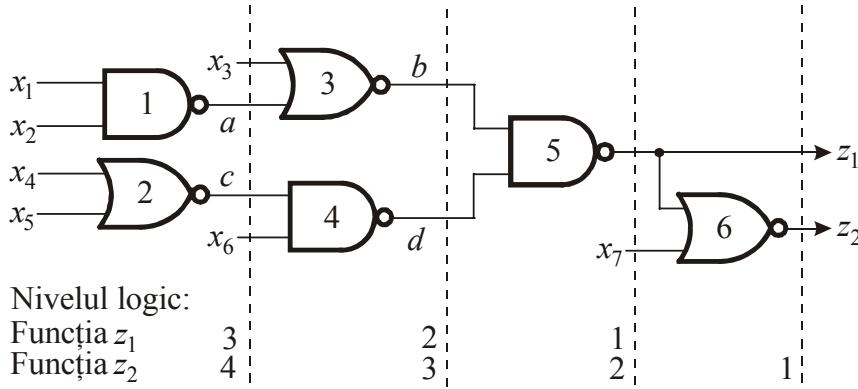


Fig.2.5 Analiza unei rețele realizate cu elemente řI-NU și SAU-NU.

4. Poarta 2 (SAU-NU) este în nivel impar (3); se consideră că realizează operația řI, iar variabilele de intrare x_4 și x_5 apar în expresia ieșirii z_1 sub formă complementată.

5. Poarta 1 (řI-NU) este în nivel impar (3); se consideră că realizează operația SAU, iar variabilele de intrare x_1 și x_2 apar în expresia ieșirii z_1 sub formă complementată.

Prin urmare, expresia logică a ieșirii z_1 este:

$$z_1 = [(\bar{x}_1 + \bar{x}_2) + x_3] + [(\bar{x}_4 \bar{x}_5)x_6] = \bar{x}_1 + \bar{x}_2 + x_3 + \bar{x}_4 \bar{x}_5 x_6. \quad (2.5)$$

Aplicând algoritmul 2.1 pentru obținerea expresiei ieșirii z_2 a rețelei din figura 2.5, rezultă:

1. Poarta 6 (SAU-NU) este în nivel impar (1); se consideră că realizează operația řI, iar variabila de intrare x_7 apare în expresia ieșirii z_2 sub formă complementată.

2. Poarta 5 (řI-NU) este în nivel par (2); se consideră că realizează operația řI.

3. Poarta 4 (řI-NU) este în nivel impar (3); se consideră că realizează operația SAU, iar variabila de intrare x_6 apare în expresia ieșirii z_2 sub formă complementată.

4. Poarta 3 (SAU-NU) este în nivel impar (3); se consideră că realizează operația řI, iar variabila de intrare x_3 apare în expresia ieșirii z_2 sub formă complementată.

5. Poarta 2 (SAU-NU) este în nivel par (4); se consideră că realizează operația SAU, iar variabilele de intrare x_4 și x_5 apar în expresia ieșirii z_2 sub formă necomplementată.

6. Poarta 1 (řI-NU) este în nivel par (4); se consideră că realizează operația řI, iar variabilele de intrare x_1 și x_2 apar în expresia ieșirii z_2 sub formă necomplementată.

Prin urmare:

$$z_2 = [(x_1 x_2) \cdot \bar{x}_3] \cdot [(x_4 + x_5) + \bar{x}_6] \cdot \bar{x}_7 = x_1 x_2 \bar{x}_3 \bar{x}_7 (x_4 + x_5 + \bar{x}_6). \quad (2.6)$$

Pentru a se pune în evidență avantajul utilizării algoritmului 2.1, în analiza rețelelor de comutare realizate cu elemente logice řI-NU și SAU-NU, în raport cu metoda "clasică" (prin care se scriu, "pas cu pas", operațiile logice efectuate de fiecare poartă logică) se vor obține, în continuare, expresiile logice ale ieșirilor z_1 și z_2 prin cea de a doua metodă menționată:

$$z_1 = \overline{b \cdot d} = \bar{b} + \bar{d}, \quad (2.7)$$

$$b = \overline{a + x_3}, \quad \bar{b} = a + x_3, \quad (2.8 \text{ a,b})$$

$$a = \overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2, \quad (2.9)$$

$$d = \overline{c \cdot x_6}, \quad \bar{d} = c \cdot x_6, \quad (2.10 \text{ a,b})$$

$$c = \overline{x_4 + x_5} = \bar{x}_4 \cdot \bar{x}_5, \quad (2.11)$$

$$z_1 = \bar{x}_1 + \bar{x}_2 + x_3 + \bar{x}_4 \bar{x}_5 x_6, \quad (2.12)$$

echivalentă cu expresia (2.5), respectiv:

$$z_2 = \overline{\bar{x}_1 + x_7} = \bar{x}_1 \cdot \bar{x}_7 = x_1 x_2 \bar{x}_3 \bar{x}_7 (x_4 + x_5 + \bar{x}_6), \quad (2.13)$$

echivalentă cu expresia (2.6).

O rețea care conține pe lângă elementele logice řI-NU și SAU-NU și elemente logice řI și SAU, poate fi analizată folosind algoritmul 2.1, dacă se transformă în prealabil într-o rețea având numai elemente řI-NU, respectiv SAU-NU. Această transformare se poate face ușor aplicând următoarea regulă:

Regulă. O rețea cu elemente logice řI, SAU, řI-NU și SAU-NU se transformă într-o rețea echivalentă având numai elemente řI-NU și SAU-NU, înlocuind fiecare element řI printr-un element řI-NU urmat de un inversor și fiecare element SAU cu un element SAU-NU urmat de un inversor. Elementele inversoare, având o singură intrare, pot fi considerate fie elemente SAU-NU fie elemente řI-NU, deci nu vor modifica expresia.

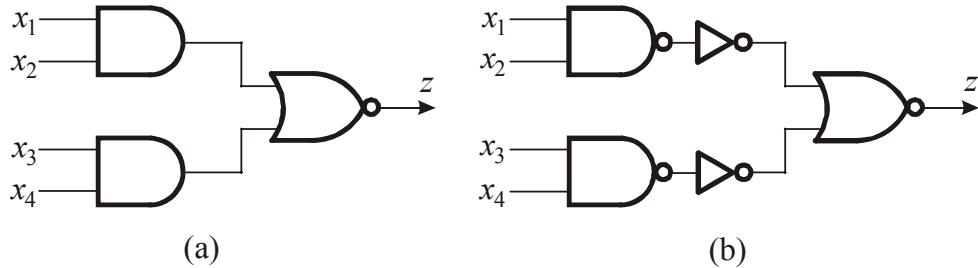


Fig.2.6 Transformarea unei rețele řI/SAU-NU într-o rețea řI-NU/SAU-NU.

Exemplu. Se consideră rețeaua řI/SAU-NU din figura 2.6,a. Folosindu-se regula de mai sus această rețea se transformă în rețeaua echivalentă din figura 2.6,b.

Pentru rețeaua din figura 2.6,b se obține, aplicând algoritmul 2.1, expresia:

$$z = (\bar{x}_1 + \bar{x}_2)(\bar{x}_3 + \bar{x}_4). \quad (2.14)$$

Expresia (2.14) este echivalentă cu expresia (2.15) obținută prin analiza rețelei 2.6,a:

$$z = \overline{x_1 x_2 + x_3 x_4}. \quad (2.15)$$

2.3 SINTEZA CIRCUITELOR LOGICE COMBINATORIALE

Problemele de sinteză ale CLC-urilor se definesc în modul următor: cunoscând semnalele de ieșire corespunzătoare diferitelor combinații ale semnalelor de intrare se cere să se stabilească structura circuitului sau, cu alte cuvinte, problema sintezei unui CLC constă în găsirea acelei rețele de comutare care să realizeze funcția (funcțiile) de comutare dată (date). Deci, sinteza CLC-urilor are ca finalitate obținerea efectivă a CLC-urilor cu elemente logice care satisfac condițiile unui sistem funcțional complet de bază (v. §1.2.4).

În general, există mai multe rețele de comutare care să realizeze aceeași funcție și anume toate rețelele ale căror expresii asociate sunt echivalente. Prin urmare, problema sintezei are mai multe soluții. Pentru practică interesează acea relație care corespunde circuitului realizabil cu cost cât mai mic, pentru tipul de elemente logice dat.

Din cele precizate mai sus se poate defini algoritmul sintezei CLC-ului și care presupune parcurgerea următoarelor etape:

1. Enunțul problemei care constă în descrierea foarte precisă a funcționării „instalației“ ce urmează a fi proiectată și care trebuie să reflecte, în mod complet, corespondența „intrare-iesire“ (text, desene, diagrame de timp).
2. Reprezentarea formală a acestei descrierii, de obicei, prin tabel de adevăr sau diagrame Karnaugh.
3. Reprezentarea printr-o funcție sau un sistem de funcții a tuturor condițiilor de comandă pentru toate elementele de execuție utilizate. Minimizarea, dacă este necesar, a acestor funcții.
4. Realizarea efectivă a schemei logice cu elemente logice ale unui sistem funcțional complet de bază.
5. Analiza schemei logice obținute, pentru a se vedea dacă corespunde condițiilor impuse inițial.
6. Completarea schemei logice cu elemente auxiliare de tipul: de amplificare, de formare a impulsurilor, de eliminare a hazardului.

2.3.1 Minimizarea funcțiilor booleene

În acest paragraf se vor prezenta principalele metode de obținere a celei mai simple forme (expresii booleene) de exprimare a funcțiilor booleene, denumită *formă minimă*.

Scopul minimizării, în practica proiectării circuitelor logice, constă în obținerea unei expresii care va costa mai puțin și/sau va opera mai rapid decât prin implementarea expresiei originale. Minimizarea constă, în principal, în transformarea formelor canonice și a formelor normale (partial simplificate) ale funcțiilor booleene în forme minime ale acestora. Procesul de minimizare poate fi extrapolat de la simplificarea fiecărei funcții booleene, la minimizarea corelată a unui ansamblu de funcții booleene.

În capitolul precedent s-a considerat problema reprezentării funcțiilor booleene prin sisteme complete având un număr minim de funcții elementare. Acest lucru vizează posibilitatea folosirii unui număr cât mai redus de tipuri de circuite logice pentru materializarea oricărei funcții booleene. În continuare se va prezenta și un alt aspect al problemei și anume cel care privește utilizarea unui număr cât mai redus de circuite standard. Din punct de vedere teoretic această problemă se reflectă în simplitatea funcțiilor booleene. În acest sens, trebuie menționat că formele canonice ale funcțiilor booleene sunt în general neeconomice.

Exemplu. Se dă următoarea funcție sub FCD:

$$f^{\text{FCD}}(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3 + x_1 x_2 x_3.$$

Asociind convenabil termenii și aplicând apoi proprietatea de distributivitate, se obține:

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 (\bar{x}_3 + x_3) + x_1 x_2 (\bar{x}_3 + x_3) = \\ &= \bar{x}_1 \bar{x}_2 x_3 + x_1 (\bar{x}_2 + x_2) = \bar{x}_1 \bar{x}_2 x_3 + x_1. \end{aligned} \quad (2.16)$$

Aplicând din nou proprietatea de distributivitate, rezultă:

$$f(x_1, x_2, x_3) = (\bar{x}_1 + x_1)(x_1 + \bar{x}_2 x_3) = x_1 + \bar{x}_2 x_3. \quad (2.17)$$

Cele două forme mai simple obținute din forma canonica sunt evident mai economice. Dintre ultimele două forme, evident ultima este cea mai simplă, având numărul minim de argumente (variabile). Din cele prezentate mai sus rezultă faptul că obținerea celei mai simple forme a unei funcții booleene prin aplicarea proprietăților algebrei Boole depinde de experiența operatorului. Acest fapt a condus la căutarea unor metode sistematice pentru obținerea expresiilor minime pentru reprezentarea funcțiilor booleene.

Problema simplificării funcțiilor booleene conduce la problema alegerii sistemului complet și la problema reprezentării cât mai economice în acest sistem. Pentru precizarea problemei minimizării funcțiilor booleene se consideră FCD și se definesc o serie de noțiuni [10,20,23].

Definiție. Conjunctiona $x_1^{a_1} x_2^{a_2} \dots x_k^{a_k}$ ($k < n$), în care fiecare variabilă se întâlnește numai o singură dată, se numește *termen normal conjunctiv*.

Definiție. Numărul literelor (variabilelor) distincte (negate sau nenegate) ale unui termen normal conjunctiv se numește *rangul* termenului normal.

Definiție. Disjunctiona termenilor normal conjunctivi se numește *formă normală disjunctivă* (FND).

Din definițiile date rezultă că FCD a unei funcții de n argumente este forma la care toți termenii sunt de rang n . Deci, FCD a unei funcții booleene este forma normală cea mai complexă.

Definiție. Forma normală disjunctivă care conține cel mai mic număr de litere $x_i^{a_i}$ în comparație cu toate celelalte FND ale unei funcții date se numește *formă minimă disjunctivă* (FMD).

În mod analog se pot face definiri similare dacă se pleacă de la FCC a unei funcții booleene. În continuare se va trata problema minimizării funcțiilor booleene plecând de la FCD, rezultatele putându-se extinde și pentru FCC.

Se va prezenta în continuare principiul unor metode sistematice de obținere a expresiilor minime pentru funcțiile booleene și care utilizează noțiunea de *implicant prim*. Se vor defini, mai întâi, noțiunile de *implicație* și *implicant*.

Definiție. Se spune că o funcție f_1 implică o altă funcție f_2 dacă pentru orice combinație de valori ale variabilelor pentru care funcția f_1 ia valoarea 1, funcția f_2 ia de asemenea valoarea 1, sau cu alte cuvinte, dacă nu există nici o combinație de valori ale variabilelor care generează cele două funcții pentru care, în urma evaluării, f_1 să

primească valoarea logică 1 iar f_2 valoarea logică 0. De exemplu, funcția $f_1 = \bar{x}_1\bar{x}_2\bar{x}_3$ implică funcția $f_2 = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3$.

Definiție. Se numesc *implicanți* ai unei funcții booleene de n argumente, termenii conjunctivi de forma $\varphi_k = x_1^{a_1}x_2^{a_2}\dots x_k^{a_k}$ ($k \leq n$) care implică funcția respectivă.

Definiție. Se numesc *implicanți primi* (IP) ai unei funcții booleene de n argumente, termenii conjunctivi de forma $\varphi_k = x_1^{a_1}x_2^{a_2}\dots x_k^{a_k}$ ($k \leq n$) care implică funcția respectivă, fără a se mai putea elimina vreo variabilă. Din definiție rezultă că implicanții primi sunt termeni de rang minim. De exemplu, dacă pentru o funcție booleană de patru argumente $f(x_1, x_2, x_3, x_4)$ au loc relațiile de implicație:

$$x_1x_2\bar{x}_3x_4 \rightarrow f(x_1, x_2, x_3, x_4), \quad (2.18)$$

$$x_2\bar{x}_3x_4 \rightarrow f(x_1, x_2, x_3, x_4), \quad (2.19)$$

$$x_2\bar{x}_3 \rightarrow f(x_1, x_2, x_3, x_4), \quad (2.20)$$

atunci $x_2\bar{x}_3$ este un implicant prim al funcției.

Implicanții primi ai unei funcții booleene se obțin plecând de la FCD și aplicând sistematic la câte doi termeni adiacenți identitatea evidentă:

$$ab + a\bar{b} = a. \quad (2.21)$$

Aplicarea identității (2.21) necesită combinarea a câte doi termeni canonici adiacenți, operație denumită *alipirea parțială* sau *compunere a vecinilor*.

Exemplu. Să se stabilească implicanții primi pentru funcția:

$$f^{\text{FCD}}(x_1, x_2, x_3) = x_1x_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + x_1x_2\bar{x}_3 + x_1x_2x_3 + \bar{x}_1\bar{x}_2x_3.$$

Folosind procedeul mai sus menționat și aplicând (2.21), rezultă:

$$x_1x_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 = x_2\bar{x}_3, \quad x_1x_2\bar{x}_3 + x_1x_2x_3 = x_1x_2.$$

Termenul $\bar{x}_1\bar{x}_2x_3$ nu se poate alipi. Astfel, implicanții primi ai funcției considerate sunt: $x_2\bar{x}_3$, x_1x_2 , $\bar{x}_1\bar{x}_2x_3$.

Implicanții primi ai unei funcții booleene se bucură de următoarele proprietăți:

- când funcția ia valoarea 0 toți implicanții primi au valoarea 0;
- când funcția ia valoarea 1 cel puțin unul dintre implicanții primi are valoarea 1; această proprietate poate fi enunțată și în felul următor: dacă unul dintre implicanții primi ia valoarea 1 și funcția ia valoarea 1.

Folosind funcția din exemplul precedent se vor verifica proprietățile enumerate. În tabelul 2.1 sunt trecute valorile funcției și ale implicanților săi primi pentru toate combinațiile posibile ale valorilor argumentelor. Se observă că atunci când $f = 0$ toți implicanții primi sunt 0 iar când $f = 1$ cel puțin unul dintre implicanții primi are valoarea 1. De asemenea, când implicanții primi sunt 1 și funcția are valoarea 1.

Este posibil ca în urma alipirilor parțiale și aplicării relației (2.21) să se mai poată aplica implicantilor rezultați și proprietatea de absorție:

$$a + ab = a, \text{ respectiv duala sa } a(a + b) = a. \quad (2.22)$$

Tab.2.1 Verificarea proprietăților implicantilor primi.

x_1	x_2	x_3	$x_2\bar{x}_3$	x_1x_2	$\bar{x}_1\bar{x}_2x_3$	$f(x_1, x_2, x_3)$
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	1	0	0	1
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	1	0	1
1	1	1	0	1	0	1

În literatura de specialitate sunt prezentate mai multe metode de minimizare a funcțiilor booleene, fiecare dintre acestea prezintând anumite avantaje. Metodele de minimizare pot fi grupate în:

- metode grafice de minimizare prin matrice de combinații sau diagrame;
- metode algebrice de minimizare.

În cele ce urmează sunt alese și prezentate numai câteva dintre acestea, cele mai reprezentative.

2.3.1.1 Minimizarea funcțiilor booleene pe baza axiomelor și teoremelor algebrei booleene

Folosind axiomele și teoremele algebrei booleene, o funcție booleană dată sub FCD sau sub FCC poate fi scrisă, în cazul general, sub o altă formă cu un număr mai mic de termeni, respectiv factori elementari căreia îi corespunde o rețea cu cost mai redus. Această metodă de minimizare a funcției booleene necesită, din partea proiectantului, multă îndemânare, ingeniozitate și experiență, motiv pentru care nu poate fi aplicată cu succes decât după o practică îndelungată în proiectarea circuitelor de comutare. Unul dintre principalele dezavantaje ale metodei îl constituie faptul că obținându-se prin calcule o anumită formă a funcției nu se poate stabili cu ușurință dacă aceasta este forma minimă sau se poate simplifica.

2.3.1.2 Minimizarea funcțiilor booleene prin metoda Quine-McCluskey

Metoda Quine-McCluskey este o metodă algebrică de minimizare a funcțiilor booleene folosită în cazul funcțiilor cu număr mare de variabile, pentru care metodele grafice de minimizare (de exemplu, metoda diagramelor Karnaugh) devin greu de utilizat. Metoda se bazează pe o teoremă datorată lui W. V. Quine, care are o mare importanță pentru

abordarea sistematică a minimizării funcțiilor booleene și care se enunță în felul următor:

Teorema lui Quine. Dacă în forma canonica disjunctivă a unei funcții booleene se fac toate operațiile de alipire parțială și apoi toate operațiile de absorție, se obține disjuncția implicantilor primi.

Fie un sistem de implicantă primă φ_k ai unei funcții de n argumente. Conform teoremei are loc relația:

$$f(x_1, x_2, \dots, x_n) = \sum_k \varphi_k . \quad (2.23)$$

Relația (2.23) trebuie să fie adevărată atât pentru $f = 0$ cât și pentru $f = 1$. Atunci când $f = 0$, aşa cum s-a arătat mai sus, toți implicantii primi ai funcției sunt 0, deci și $\sum \varphi_k = 0$. Când $f = 1$, va exista cel puțin un implicant prim $\varphi_j = 1$, astfel că întreaga disjuncție din partea dreaptă a relației (2.23) va avea valoarea 1.

Relația (2.23) este o formă normală disjunctivă a funcției și se numește *formă disjunctivă prescurtată* (FDP). Această formă nu este minimă deoarece, în general, există implicantă primă care implică suplimentar funcția. După eliminarea implicantilor primi redundanți, rămân numai implicantii primi strict necesari care sunt numiți *implicantii primi esențiali* (IPE) ai funcției. Disjuncția implicantilor primi esențiali conduce la forma minimă disjunctivă. Din cele prezentate rezultă că minimizarea unei funcții booleene date sub FCD comportă două etape:

1. determinarea FDP prin căutarea implicantilor primi;
2. căutarea reuniunii minime care conține cel mai mic număr de implicantă primă.

Pentru stabilirea reuniunii minime se construiește *tabelul implicantilor primi*, în care fiecare linie corespunde unui implicant prim, iar fiecare coloană unui termen canonice conjunctiv. Corespondențele între termenii canonici și implicantii primi se marchează la intersecția liniilor cu coloanele respective. Se rețin numai acei implicantă primă necesari să acopere toate coloanele; aceștia sunt implicantii primi esențiali.

În metoda concepută de Quine există un neajuns determinat de necesitatea comparării (alipirii) complete a perechilor de termeni în prima etapă. Cu creșterea numărului termenilor canonici care definesc FCD a funcției considerate crește numărul acestor comparații. Această creștere este caracterizată de o funcție factorială. Din acest motiv, la un număr suficient de mare de mintermi folosirea metodei Quine devine greoaie. McCluskey a îmbunătățit prima etapă a metodei lui Quine prin transcrierea binară a termenilor canonici. În acest mod se poate face o sistematizare a comparării mintermilor, grupându-i după numărul de biți 1. Astfel, în grupa i intră toate numerele corespunzătoare mintermilor care au în transcriere binară i biți 1. Compararea perechilor se poate face numai între grupe vecine, deoarece numai aceste grupe diferă între ele cu un singur bit 1. La reprezentarea termenilor normali rezultați prin eliminarea variabilelor, în locul acestor variabile se trece o linie orizontală.

Completață în modul menționat de către McCluskey, metoda de minimizare descrisă este cunoscută ca *metoda Quine-McCluskey* [10,20,22,23,...].

Revenind la prima etapă, pentru determinarea implicantilor primi ai funcției se aplică următoarea procedură [22]:

Algoritm 2.2.

1. Se pornește de la FCD a funcției în care termenii canonici sunt dați sub forma

unui produs de variabile fie prin notația simbolică. Fiecare termen canonic este reprezentat apoi sub formă de număr binar, prin n -uplul de zerouri și unități corespondente termenului respectiv.

2. Termenii canonici astfel scriși se împart în grupe, în funcție de ponderea acestora, adică de numărul unităților cuprinse în n -uplul respectiv.

3. Grupele de termeni canonici sunt aranjate pe o coloană, în ordinea crescătoare a ponderilor. (Acest lucru este util deoarece doi termeni canonici se pot asocia formând un subcub 1-dimensional² numai dacă fac parte din grupe ale căror ponderi diferă cu o unitate).

4. Se compară fiecare termen canonic al unei grupe cu toți termenii canonici ai grupei de pondere mai mare cu o unitate. Dacă numerele binare respective sunt adiacente, cei doi termeni canonici se pot asocia formând un subcub 1-dimensional, notat printr-un număr binar care are pe poziția prin care cei doi termeni compenenți diferă, o linie orizontală, ceea ce semnifică faptul că variabila corespondentă acelei poziții lipsește (este o variabilă redundantă); cei doi termeni canonici care au format subcubul 1-dimensional se bifează, iar termenul normal care reprezintă subcubul rezultat se înscrie pe o nouă coloană. Toți termenii normali (subcuburile 1-dimensionale) rezultați în urma comparării a două grupe din coloana termenilor canonici formează o grupă în coloana subcuburilor 1-dimensionale. Prin urmare, coloana subcuburilor 1-dimensionale va conține, în cazul general, cu o grupă mai puțin decât coloana termenilor canonici (a subcuburilor 0-dimensionale).

5. Se compară, în continuare, fiecare termen al unei grupe din coloana subcuburilor r -dimensionale (se consideră, pentru început, $r = 1$) cu toți termenii grupei cu pondere mai mare cu o unitate. Pentru ca doi asemenea termeni să se poată asocia formând un subcub $(r + 1)$ -dimensional trebuie ca în ambii termeni simbolurile „–“ să fie pe aceleași poziții. Doi termeni care îndeplinesc această condiție și sunt adiacenți se asociază formând un subcub $(r + 1)$ -dimensional care se notează cu un număr binar în care apare încă o linie orizontală pe poziția prin care cei doi termeni diferă. Termenii care participă la formarea acestui subcub se bifează, iar subcubul $(r + 1)$ -dimensional se înscrie pe o nouă coloană, coloana subcuburilor $(r + 1)$ -dimensionale, care, în cazul general, are cu o grupă mai puțin decât coloana subcuburilor r -dimensionale. Dacă se obține de mai multe ori un anumit termen, acesta se consideră doar o singură dată.

6. Se mărește r cu o unitate și se repetă punctul 5, până când subcuburile ultimei coloane nu se mai pot asocia în scopul formării unui subcub de dimensiune superioară. În acest moment prima etapă a algoritmului Quine-McCluskey este încheiată. Termenii rămăși nebifați în coloanele rezultate formează grupul implicantilor primi ai funcției considerate.

Exemplu. În tabelul 2.2 se dă un exemplu de aplicare a algoritmului prezentat mai sus pentru minimizarea funcției de cinci variabile dată sub FCD:

$$f^{\text{FCD}}(x_1, x_2, x_3, x_4, x_5) = \sum_{i \in k} P_i, \quad (2.24)$$

unde, $k = \{0, 3, 4, 7, 8, 10, 11, 12, 14, 15, 18, 19, 23, 26, 28, 29, 30\}$.

În tabelul 2.2 s-au scris alături de fiecare subcub și indicii termenilor canonici sau ai

² Pentru definirea noțiunii de *subcub* vezi metoda diagramelor Karnaugh.

SINTEZA CIRCUITELOR LOGICE COMBINATORIALE

Tab.2.2 Determinarea implicantilor primi pentru functia f din exemplu.

Subcuburi 0-dimensionale			Subcuburi 1-dimensionale					Subcuburi 2-dimensionale				
Grupa	Indici	$x_1 \ x_2 \ x_3 \ x_4 \ x_5$	Grupa	Indici	$x_1 \ x_2 \ x_3 \ x_4 \ x_5$	Grupa	Indici	$x_1 \ x_2 \ x_3 \ x_4 \ x_5$				
0	0	0 0 0 0 0 ✓	0	0,4	0 0 - 0 0 ✓	0	0,4,8,12	0 - - 0 0				
1	4	0 0 1 0 0 ✓		0,8	0 - 0 0 0 ✓		0,8,4,12	0 - - 0 0				
	8	0 1 0 0 0 ✓	1	4,12	0 - 1 0 0 ✓	1	8,10,12,14	0 1 - - 0				
2	3	0 0 0 1 1 ✓		8,10	0 1 0 - 0 ✓		8,12,10,14	0 1 - - 0				
	10	0 1 0 1 0 ✓		8,12	0 1 - 0 0 ✓	2	3,7,11,15	0 - - 1 1				
	12	0 1 1 0 0 ✓	2	3,7	0 0 - 1 1 ✓		3,7,19,23	- 0 - 1 1				
	18	1 0 0 1 0 ✓		3,11	0 - 0 1 1 ✓		3,11,7,15	0 - - 1 1				
3	7	0 0 1 1 1 ✓		3,19	- 0 0 1 1 ✓		3,19,7,23	- 0 - 1 1				
	11	0 1 0 1 1 ✓		10,11	0 1 0 1 - ✓		10,11,14,15	0 1 - 1 -				
	14	0 1 1 1 0 ✓		10,14	0 1 - 1 0 ✓		10,14,26,30	- 1 - 1 0				
	19	1 0 0 1 1 ✓		10,26	- 1 0 1 0 ✓		10,26,14,30	- 1 - 1 0				
	26	1 1 0 1 0 ✓		12,14	0 1 1 - 0 ✓		12,14,28,30	- 1 1 - 0				
	28	1 1 1 0 0 ✓		12,28	- 1 1 0 0 ✓		12,28,14,30	- 1 1 - 0				
4	15	0 1 1 1 1 ✓	3	7,15	0 - 1 1 1 ✓							
	23	1 0 1 1 1 ✓		7,23	- 0 1 1 1 ✓							
	29	1 1 1 0 1 ✓		11,15	0 1 - 1 1 ✓							
	30	1 1 1 1 0 ✓		14,15	0 1 1 1 - ✓							
				14,30	- 1 1 1 0 ✓							
				19,23	1 0 - 1 1 ✓							
				26,30	1 1 - 1 0 ✓							
				28,29	1 1 1 0 -							
				28,30	1 1 1 - 0 ✓							

subcuburilor 0-dimensionale care sunt incluse în subcubul respectiv. Urmărind acești indici se observă că în coloana subcuburilor 0-dimensionale doi termeni canonici se pot asocia dacă diferența între indicele termenului din grupa cu ponderea i și indicele termenului din grupa cu ponderea $i-1$ este un număr întreg, pozitiv și egal cu o putere a lui 2. Puterea lui 2 indică poziția din numărul binar pe care apare simbolul „-“. De

exemplu, termenul cu indicele 4 din grupa 1 și termenul cu indicele 12 din grupa 2 respectă această condiție și anume $12 - 4 = 8$. Subcubul 1-dimensional rezultat prin asocierea acestor termeni conține partea comună a termenilor incluși, iar pe poziția de rang $2^3 = 8$ apare simbolul „–“. În cazul coloanelor subcuburilor de dimensiune mai mare decât zero, pentru ca două subcuburi să facă parte din grupe a căror pondere diferă cu o unitate să se poată asocia, trebuie ca diferența între indicii termenilor canonici incluși în subcubul din grupa superioară și indicii termenilor canonici corespondenți ai subcubului din grupa inferioară să fie o aceeași putere a lui 2. De exemplu, al treilea subcub 1-dimensional din grupa a 3-a și primul din grupa a 2-a respectă această condiție ($11 - 3 = 8$ și $15 - 7 = 8$). Înținând cont de aceste observații se poate ușura munca depusă pentru găsirea implicantilor primi.

Ca rezultat al aplicării regulilor de găsire a implicantilor primi ai funcției (2.24), din tabelul 2.2 rezultă următoarea expresie:

$$f(x_1, \dots, x_5) = x_1\bar{x}_2\bar{x}_3x_4 + x_1\bar{x}_3x_4\bar{x}_5 + x_1x_2x_3\bar{x}_4 + \bar{x}_1\bar{x}_4\bar{x}_5 + \bar{x}_1x_2\bar{x}_5 \\ + \bar{x}_1x_4x_5 + \bar{x}_2x_4x_5 + \bar{x}_1x_2x_4 + x_2x_4\bar{x}_5 + x_2x_3\bar{x}_5. \quad (2.25)$$

Această expresie se obține înlocuind în termenii rămași nebifați în tabel zerourile cu variabila corespunzătoare poziției respective negată, unitățile cu variabila corespunzătoare poziției respective nenegată și omitând variabilele corespunzătoare pozițiilor pe care apare simbolul „–“.

Pentru a găsi forma minimă disjunctivă a unei funcții trebuie aleși numai acei implicantă primă care includ toți termenii canonici ai funcției și conduc la o formă a funcției realizată cu cost minim. Pentru găsirea acoperirii cu cost minim trebuie căutate toate acoperirile posibile pentru funcția dată din care se alege acoperirea care îndeplinește condiția de cost minim față de un anumit criteriu de cost [22]. Costul C_R se definește ca fiind suma costurilor implicantilor primi din acoperirea considerată. Costul unui implicant prim al unei funcții de n variabile, din care lipsesc r variabile, adică a unui subcub r -dimensional este $n - r$, deoarece fiecare variabilă prezentă necesită un contact. Atunci costul C_R al unei acoperiri este dat de relația:

$$C_R = \sum_{r=0}^{n-1} g_r(n-r), \quad (2.26)$$

unde, g_r este numărul subcuburilor r -dimensionale din acoperirea considerată, iar însumarea se face pentru toate subcuburile de dimensiune $0 \leq r \leq n$. Costul C_R al unei acoperiri este minim când suma costurilor implicantilor primi este minimă.

În cazul circuitelor de comutare cu porți, costul circuitelor este în general cu atât mai mic cu cât numărul de porți este mai mic, iar porțile respective au număr cât mai mic de intrări, ceea ce înseamnă costul C_R minim. Din acest motiv, atunci când funcția se realizează cu porți, se alege acoperirea a cărei cost C_P , definit de relația:

$$C_P = C_R + p = \sum_{r=0}^{n-1} g_r(n-r) + \sum_{r=0}^{n-1} g_r = \sum_{r=0}^{n-1} g_r(n-r+1), \quad (2.27)$$

este minimă. În relația (2.27), p este numărul implicantilor primi ai acoperirii, iar restul notațiilor au aceeași semnificație ca în relația (2.26).

De obicei, acoperirea minimală, adică acoperirea cu număr minim de elemente, satisfac atât condiția de C_R minim cât și condiția de C_P minim. Algoritmul de obținere a acoperirii cu cost minim, plecând de la mulțimea implicantilor primi, obținuți în etapa anterioară, este următorul [22]:

Algoritm 2.3.

1. Se construiește un tabel al implicantilor primi, având drept cap de linie implicantii primi ai funcției și cap de coloană termenii canonici ai funcției. La intersecția unei linii cu o coloană se plasează un semn, de exemplu un asterisc, dacă implicantul prim de pe linia respectivă include termenul canonic de pe coloana respectivă (v. tab. 2.3).

Tab.2.3 Tabelul implicantilor primi pentru funcția f din exemplu.

Implicantii primi					Indici	Termeni canonici															Obs		
x_1	x_2	x_3	x_4	x_5		0	4	8	3	10	12	18	7	11	14	19	26	28	15	23	29	30	
1	0	0	1	-	(18,19)						*				*								
1	-	0	1	0	(18,26)						*					*							IPE
1	1	1	0	-	(28,29)						*												IPE
0	-	-	0	0	(0,4,8,12)	*	*	*		*	*												
0	1	-	-	0	(8,10,12,14)			*	*	*	*				*								
0	-	-	1	1	(3,7,11,15)			*				*	*		*								
-	0	-	1	1	(3,7,19,23)			*				*	*		*								
0	1	-	1	-	(10,11,14,15)			*		*			*	*									
-	1	-	1	0	(10,14,26,30)			*		*			*			*							
-	1	1	-	0	(12,14,28,30)										*								*

2. Se inspectează tabelul construit la punctul 1. Dacă pe o anumită coloană există un singur semn, ceea ce înseamnă că termenul canonic de pe acea coloană este inclus într-un singur implicant prim, atunci implicantul prim de pe linia însemnată devine implicant prim esențial și intră obligatoriu în forma minimă a funcției. Se construiește un nou tabel, al implicantilor primi neesențiali, care rezultă eliminând din tabelul implicantilor primi liniile cu implicantii primi esențiali și coloanele cu termenii canonici incluși în aceștia. Pentru a găsi mai ușor aceste coloane se încercuiesc toate semnele de pe linia unui implicant prim esențial și se elimină apoi toate coloanele care au semne încercuite. De asemenea, se elimină liniile pe care nu au mai rămas semne și coloanele care au semne pe aceleași rânduri ca și o altă coloană din tabelul implicantilor primi neesențiali, adică dacă mai mulți termeni canonici sunt incluși în exact aceiași implicant primi neesențiali se reține un singur reprezentat al acestora (de exemplu termenii canonici 11 și 15), deoarece orice implicant care îl include pe acesta va include automat și pe cei omisi (v. tab. 2.4).

3. Se inspectează tabelul implicantilor primi neesențiali în scopul găsirii unei acoperiri cu cost C_P minim, pentru toți termenii canonici rămași în acest tabel. În unele cazuri se pot găsi mai multe acoperiri care satisfac această condiție, fiecare având același cost. În astfel de situații funcția are mai multe forme minime disjunctive.

4. Făcând suma booleană a implicantilor primi esențiali găsiți la punctul 2 și a celor

neesențiali făcând parte din acoperirea obținută la punctul 3, se obține FMD a funcției date.

Tab.2.4 Tabelul implicantilor primi neesențiali pentru funcția f din exemplu.

Nr.	Implicantii primi					Indici	Termeni canonici						Obs.
	x_1	x_2	x_3	x_4	x_5		10	18	11	14	26	30	
1	1	0	0	1	-	(18,19)	*	*			*		
2	1	-	0	1	0	(18,26)		*				*	
3	0	1	-	-	0	(8,10,12,14)	*			*			A
4	0	-	-	1	1	(3,7,11,15)			*				
5	0	1	-	1	-	(10,11,14,15)	*	*	*	*			A
6	-	1	-	1	0	(10,14,26,30)	*			*	*	*	A
7	-	1	1	-	0	(12,14,28,30)			*			*	

Pentru exemplul considerat anterior, funcția dată prin expresia (2.24), se obține tabelul implicantilor primi prezentat în tabelul 2.3.

Inspectând acest tabel se obțin implicantii primi esențiali $x_1x_2x_3\bar{x}_4$, $\bar{x}_1\bar{x}_4\bar{x}_5$ și $\bar{x}_2x_4x_5$ (de pe rândurile marcate cu IPE). După eliminarea termenilor canonici acoperiți de acești implicantii primi se obține tabelul implicantilor neesențiali (v. tab. 2.4).

Din tabelul 2.4 rezultă că, luând implicantii $x_1\bar{x}_3x_4\bar{x}_5$, $\bar{x}_1x_2x_4$, și $x_2x_4\bar{x}_5$, de pe rândurile marcate cu A, se obține acoperirea minimală căreia îi corespunde FMD a funcției, dată de relația:

$$f^{\text{FMD}}(x_1, x_2, x_3, x_4, x_5) = x_1x_2x_3\bar{x}_4 + \bar{x}_1\bar{x}_4\bar{x}_5 + \bar{x}_2x_4x_5 + x_1\bar{x}_3x_4\bar{x}_5 + \bar{x}_1x_2x_4 + x_2x_4\bar{x}_5. \quad (2.28)$$

Analizând tabelul 2.4 rezultă că mai există și alte acoperiri minimale pentru această funcție. Pentru găsirea acestora se procedează în felul următor [22]:

Algoritm 2.4.

1. Se împarte mulțimea implicantilor primi neesențiali în submulțimi M_i , astfel încât elementele unei submulțimi să conțină toți implicantii primi neesențiali care includ termenul canonic cu indicele i . Notând implicantii primi neesențiali din tabelul 2.4 cu numărul lor de ordine, rezultă submulțimile:

$$M_{10} = \{3,5,6\}; M_{18} = \{1,2\}; M_{11} = \{4,5\}; M_{14} = \{3,5,6,7\};$$

$$M_{26} = \{2,6\}; M_{30} = \{6,7\}. \quad (2.29)$$

2. Se alcătuiește, cu elementele submulțimii M_i , expresia formală (2.30), în care suma se interpretează ca operația logică SAU iar produsul ca operația logică SI:

$$E = (3+5+6)(1+2)(4+5)(3+5+6+7)(2+6)(6+7). \quad (2.30)$$

3. Efectuând calculele în expresia (2.30), ținând cont de faptul că $i+i=i$ și

$i(i+j) = i$, unde i și j sunt elemente ale submulțimilor M_i , se obține o nouă formă a expresiei E , o sumă de produse. Fiecare dintre aceste produse de implicantă primă reprezintă una dintre acoperirile mulțimii termenilor canonici din tabelul implicantelor primă neessențiali.

Tinând cont de cele de mai sus, expresia (2.30) devine:

$$\begin{aligned} E &= (3+5+6)(1+2)(4+5)(2+6)(6+7) = (6+7)(3+5)(2+1\cdot 6)(4+5) = \\ &= (6+7\cdot 3+7\cdot 5)(2\cdot 4+2\cdot 5+1\cdot 6\cdot 4+1\cdot 6\cdot 5) = 6\cdot 2\cdot 4+6\cdot 2\cdot 5+1\cdot 6\cdot 4+ \\ &\quad +1\cdot 6\cdot 5+7\cdot 3\cdot 2\cdot 4+7\cdot 3\cdot 2\cdot 5+7\cdot 3\cdot 1\cdot 6\cdot 4+7\cdot 3\cdot 1\cdot 6\cdot 5+7\cdot 5\cdot 2\cdot 4+ \\ &\quad +7\cdot 5\cdot 2+7\cdot 5\cdot 1\cdot 6\cdot 4+7\cdot 5\cdot 1\cdot 6. \end{aligned} \quad (2.31)$$

4. Se ia forma minimală a expresiei (2.31) în care intră numai produsele cu cost minim. Termenii acestei expresii reprezintă acoperirile cu cost minim. Pentru expresia (2.30) forma minimală, considerând costul C_R , este:

$$E_{min} = 6\cdot 2\cdot 4+6\cdot 2\cdot 5+1\cdot 6\cdot 4+1\cdot 6\cdot 5+7\cdot 5\cdot 2. \quad (2.32)$$

Expresia (2.32) conține cele cinci acoperiri cu cost C_R minim ale tabelului 2.4. Costul fiecărei acoperiri este 10. Deoarece fiecare dintre cele cinci acoperiri are același număr de elemente, ele au și același cost C_P .

Așa cum s-a precizat mai sus minimizarea funcțiilor booleene prin metoda Quine-McCluskey necesită considerarea funcțiilor sub FCD. Problema poate fi tratată și pentru funcțiile booleene date sub FCC, în care caz operațiile de alipire parțială a termenilor disjunctivi sunt urmate de aplicarea sistematică a formelor duale pentru relațiile (2.21) și (2.22):

$$(a+b)(a+\bar{b}) = a \text{ și respectiv } a(a+b) = a. \quad (2.33)$$

Etapa a doua, de căutare a intersecției minime, se realizează similar ca și pentru FCD, obținându-se în final *forma minimă conjunctivă* (FMC).

O altă modalitate de obținere a FMC pentru o funcție booleană dată prin FCC constă în considerarea funcției negate:

$$\bar{f}(x_1, x_2, \dots, x_n) = \overline{\prod_0 (x_1^{\bar{a}_1} + x_2^{\bar{a}_2} + \dots + x_n^{\bar{a}_n})} = \prod_0 (\overline{x_1^{\bar{a}_1}} \overline{x_2^{\bar{a}_2}} \dots \overline{x_n^{\bar{a}_n}}), \quad (2.34)$$

care se poate trata ca și FCD. Se obține astfel forma minimă disjunctivă a funcției negate. Negând expresia obținută se revine la forma conjunctivă, corespunzând formei minime conjunctive.

Metoda Quine-McCluskey începe să devină greu de manipulat pentru un număr mare de variabile ($n \geq 6$) însă prezintă avantajul realizării unor algoritmi pentru calcul numeric. De asemenea, principiul metodei expuse poate fi transpus pe diagrame numite Karnaugh, căpătând un caracter intuitiv și ușurință în aplicarea pentru funcții având 6÷7 variabile.

2.3.1.3 Minimizarea funcțiilor booleene prin metoda diagramele Karnaugh

Folosirea unei forme speciale a diagramele Venn, în scopul simplificării cu ajutorul acestora a funcțiilor booleene a fost sugerată pentru prima dată de către E.W. Veitch. La scurt timp M. Karnaugh propune, de asemenea, o formă modificată a diagramele Venn, în același scop. Astfel, au rezultat diagramele denumite *diagrame Karnaugh*.

Spre deosebire de metoda Quine-McCluskey, metoda diagramele Karnaugh este o *metodă globală*, în sensul că ambele etape din metoda Quine-McCluskey se efectuează simultan. Ca și în cazul metodei Quine-McCluskey, metoda diagramele Karnaugh pornește de la una dintre formele canonice (FCD sau FCC) ale funcției booleene. Pentru început, în prezentarea metodei, se va folosi FCD. Diagramele Karnaugh sunt folosite în mod curent pentru reprezentarea funcțiilor booleene cu un număr relativ mic de variabile [10,22,...]. Aceste diagrame sunt utile pentru minimizarea funcțiilor booleene deoarece permit evidențierea, cu ușurință, a unor identități de forma (legile absorbtiei):

$$a + ab = a, \quad ab + a\bar{b} = a, \quad a + \bar{a}b = a + b. \quad (2.35)$$

În general, o diagramă Karnaugh pentru o funcție booleană de n variabile se reprezintă sub forma unui pătrat sau dreptunghi, împărțit în 2^n compartimente (câmpuri sau locații), fiecare compartiment fiind rezervat unui termen canonic al funcției, respectiv unui dintre cele 2^n n -uple ale funcției sau vârfuri ale hipercubului n -dimensional din reprezentarea pe hipercub a funcției (v. Anexa A). În figura 2.7 sunt reprezentate diagramele Karnaugh pentru funcții de două variabile (v. fig. 2.7,a), trei variabile (v. fig. 2.7,b) și patru variabile (v. fig. 2.7,c). O diagramă Karnaugh este astfel organizată încât două compartimente vecine, pe o linie sau pe o coloană, corespund la doi termeni canonici care diferă numai printr-o singură variabilă, care apare într-unul dintre ei negată,

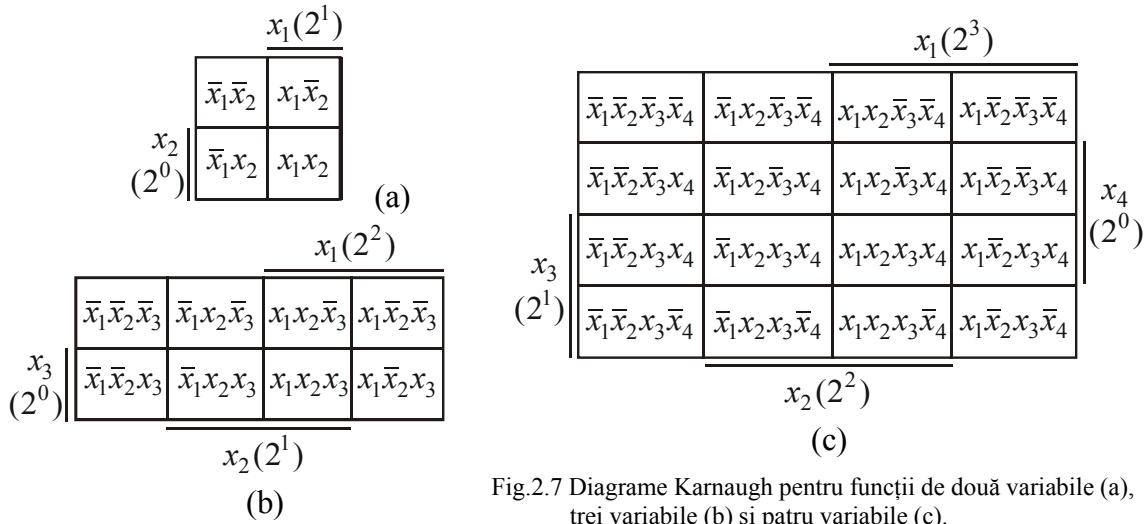


Fig.2.7 Diagrame Karnaugh pentru funcții de două variabile (a), trei variabile (b) și patru variabile (c).

iar în celălalt adevărată (deci au proprietatea de adiacență), respectiv la două n -uple adiacente. Se consideră vecine și deci au aceeași proprietate și compartimentele aflate la capetele opuse ale unei linii, respectiv coloane. Diagrama se notează (codifică) fie indicând domeniul fiecărei variabile (v. fig. 2.8,a), fie indicând pe linie și coloană n -uple de zerouri și unități corespondente unui compartiment din diagramă și ordinea variabilelor (v. fig. 2.8,b). De fapt, în acest ultim caz se folosește pentru codificarea diagramei, aşa

cum s-a mai precizat (v. §1.2.3), codul Gray (binar reflectat) care are proprietatea de adiacență. Pentru a se putea reprezenta ușor funcțiile booleene date în mod convențional (simbolic) prin indicii zecimali ai termenilor canonici, se poate nota fiecare compartiment prin indicele termenului canonic corespunzător, ținând cont de o anumită ordine a variabilelor (v. fig. 2.8,a,b).

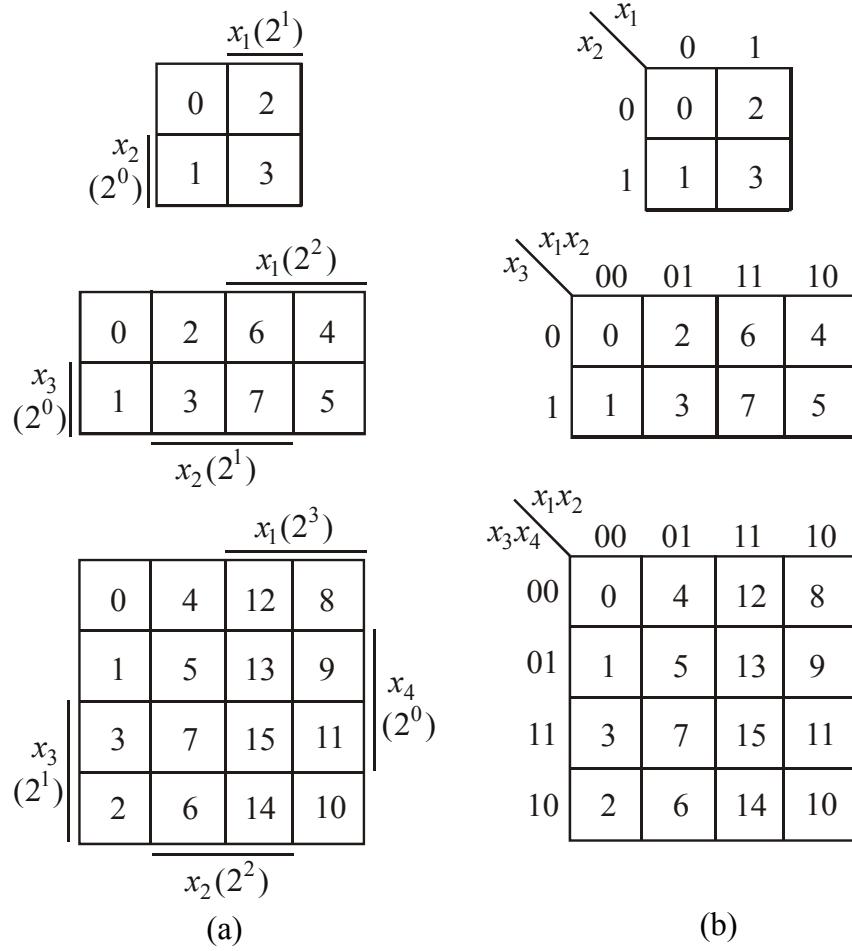


Fig.2.8 Modalități de codificare a diagramelor Karnaugh.

O funcție booleană, dată sub FCD, poate fi reprezentată pe o diagramă Karnaugh marcând, de exemplu cu 1, compartimentele corespunzătoare termenilor canonici ai funcției și cu 0 celelalte compartimente. Acest mod de reprezentare a funcțiilor booleene este avantajos pentru minimizare deoarece doi termeni canonici care diferă numai prin aceea că într-unul dintre aceștia una dintre variabile apare negată, iar în celălalt adevărată, de exemplu $x_1\bar{x}_2x_3\bar{x}_4$ și $x_1\bar{x}_2x_3x_4$, apar pe diagramă în compartimente vecine, deci se recunoaște ușor că sunt adiacenți. În conformitate cu relația (2.36) acești doi termeni se pot înlocui cu un termen normal în care variabila, prin care cei doi termeni canonici diferă, lipsește (este o variabilă redundantă):

$$x_1\bar{x}_2x_3\bar{x}_4 + x_1\bar{x}_2x_3x_4 = x_1\bar{x}_2x_3(\bar{x}_4 + x_4) = x_1\bar{x}_2x_3. \quad (2.36)$$

În reprezentarea pe hipercub a unei funcții booleene, doi termeni canonici care îndeplinesc condiția de mai sus corespund la două noduri adiacente, deci definesc o latură a

hipercubului n -dimensional (v. Anexa A). Din acest motiv se spune că două compartimente vecine sau adiacente pe diagrama Karnaugh reprezintă un *subcub 1-dimensional* (v. fig. 2.9,a). Un grup de patru compartimente, dintre care fiecare este vecin cu alte două compartimente din același grup, formează un *subcub 2-dimensional* (v. fig. 2.9,b). În acest caz cei patru termeni canonici corespunzători acestor compartimente au o parte comună formată din două variabile. În baza relației (2.37) acești patru termeni pot fi înlocuiți cu partea lor comună:

$$x_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 = \bar{x}_1\bar{x}_3\bar{x}_4 + x_1\bar{x}_3\bar{x}_4 = \bar{x}_3\bar{x}_4. \quad (2.37)$$

În cazul general, termenii canonici care formează un subcub 2-dimensional se pot înlocui cu un termen normal având cu două variabile mai puțin decât termenii canonici. Pe o diagramă de patru variabile se pot forma și *subcuburi 3-dimensionale* care cuprind 8 compartimente astfel grupate încât fiecare dintre ele este vecin cu alte trei din același grup (v. fig. 2.9,c).

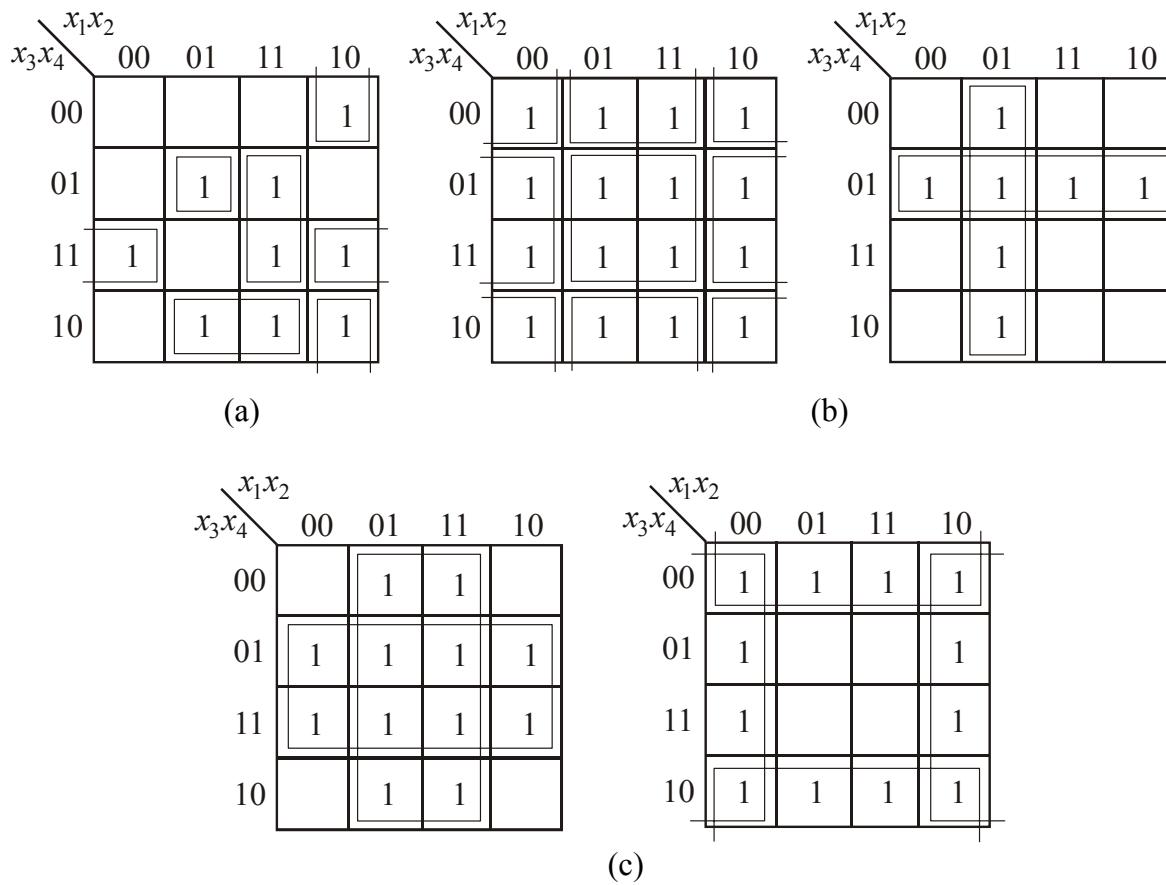


Fig.2.9 Reprezentarea pe diagrame Karnaugh a subcuburilor 0-dimensionale (a), 1-dimensionale (a), 2-dimensionale (b) și 3-dimensionale (c).

Termenii canonici reprezentăți de compartimentele din diagrama de patru variabile care formează un subcub 3-dimensional au o parte comună formată dintr-o singură variabilă. Prin urmare, pot fi înlocuiți cu un termen normal care are o singură variabilă, cum se vede în exemplul de mai jos:

$$\begin{aligned} & \bar{x}_1x_2\bar{x}_3x_4 + \bar{x}_1x_2x_3x_4 + x_1x_2x_3x_4 + x_1x_2\bar{x}_3x_4 + \bar{x}_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2x_3\bar{x}_4 + \\ & + x_1x_2x_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 = x_2. \end{aligned} \quad (2.38)$$

Pe o diagramă care reprezintă o anumită funcție booleană se pot forma în modul arătat mai sus subcuburi de diverse dimensiuni. În cazul general al unei funcții booleene de n variabile, dacă o parte dintre termenii canonici ai funcției formează un subcub k -dimensional, aceștia se pot înlocui cu un singur termen normal având $n-k$ variabile. Un subcub care nu este inclus într-un subcub de dimensiune mai mare se numește *implicant prim* al funcției date. Formând suma booleană a tuturor implicanților primi ai unei funcții date, se obține o formă disjunctivă a acesteia, care, în cazul general, este mult mai simplă decât FCD a aceleiași funcții. Pentru a găsi implicanții primi ai unei funcții reprezentată pe diagrama Karnaugh, compartimentele marcate cu 1 trebuie astfel grupate încât să se obțină subcuburi cu dimensiunea cea mai mare posibilă. Astfel, pe o diagramă de patru variabile se va căuta să se formeze în primul rând subcuburi 3-dimensionale, apoi în ordine, subcuburi 2-dimensionale și 1-dimensionale. O parte (dar nu toate) dintre compartimentele notate cu 1 ale unui subcub poate face parte din mai multe subcuburi de diverse dimensiuni.

Exemplu. Se dă funcția booleană de patru variabile (v. fig. 2.10):

$$f^{\text{FCD}}(x_1, x_2, x_3, x_4) = P_1 + P_5 + P_6 + P_7 + P_8 + P_9 + P_{12} + P_{13}. \quad (2.39)$$

Procedând în modul arătat mai sus se obțin implicanții primi ai acestei funcții și anume subcuburile 2-dimensionale $x_1\bar{x}_3$ și \bar{x}_3x_4 și subcuburile 1-dimensionale $\bar{x}_1x_2x_4$ și $\bar{x}_1x_2x_3$.

Pentru mai multă claritate, compartimentele care reprezintă un implicant prim sunt înscrise într-un contur, iar în dreptul conturului se scrie termenul normal corespondent implicantului prim încadrat (cu asterisc s-au marcat compartimentele care nu mai sunt incluse în alte subcuburi). Folosind pentru descrierea funcției implicanții primi în locul termenilor canonici se obține forma normală disjunctivă a funcției dată de relația (2.40):

$$f(x_1, x_2, x_3, x_4) = x_1\bar{x}_3 + \bar{x}_3x_4 + \bar{x}_1x_2x_4 + \bar{x}_1x_2x_3. \quad (2.40)$$

Nu toți implicanții primi ai unei funcții sunt însă necesari pentru definirea acesteia. Unii dintre implicanții primi pot fi termeni redundanți la care se poate renunța, obținându-se astfel o formă mai simplă a funcției. Prin urmare, se pune problema alegerii unui număr minim de implicanți primi din mulțimea implicanților primi ai unei funcții booleene care să includă însă toți termenii canonici ai funcției. Problema alegerii acestui set de implicanți este o problemă de acoperire cu cost minim. Expresia unei funcții booleene în care intră un număr minim dintre implicanții săi primi se numește, aşa cum s-a

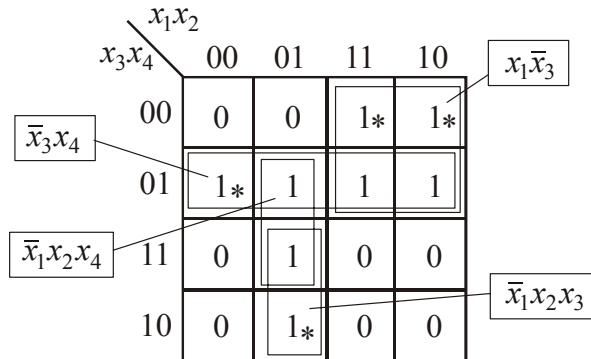


Fig.2.10 Determinarea implicanților primi pe diagrama Karnaugh.

mai precizat, forma minimă disjunctivă (FMD) a funcției. Dacă un termen canonic al funcției este inclus într-un singur implicant prim, din mulțimea de implicantii primi ai funcției, acela devine *implicant prim esențial* și trebuie să apară în mod obligatoriu în FMD a funcției. Pentru găsirea FMD trebuie deci determinați în primul rând implicantii primi esențiali. Pentru aceasta se marchează pe diagramă într-un mod special, de exemplu cu asterisc, acele celule notate cu 1 care sunt incluse într-un singur implicant prim (v. fig. 2.10). Implicantii primi care conțin compartimente marcate cu asterisc devin esențiali. Compartimentele marcate cu 1, rămasse neacoperite de către implicantii primi esențiali, se caută să se acopere folosind un număr cât mai mic din implicantii funcției care au mai rămas. În cazul exemplului din figura 2.10, toate compartimentele marcate cu 1, adică toți termenii canonici ai funcției, sunt acoperiți de implicantii primi esențiali $x_1\bar{x}_3$, \bar{x}_3x_4 și $\bar{x}_1x_2x_3$. Implicantul $\bar{x}_1x_2x_4$ reprezintă un termen redundant deoarece ambii termeni canonici pe care îi include sunt cuprinși în implicantul prim esențial \bar{x}_3x_4 , respectiv $\bar{x}_1x_2x_3$. Din cele prezentate mai sus rezultă că FMD a funcției, dată de relația (2.39), este:

$$f^{\text{FMD}}(x_1, x_2, x_3, x_4) = x_1\bar{x}_3 + \bar{x}_3x_4 + \bar{x}_1x_2x_3. \quad (2.41)$$

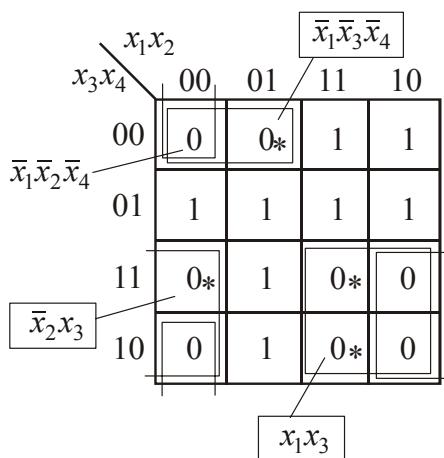


Fig.2.11 Obținerea FMC.

O funcție booleană are și o formă minimă conjunctivă (FMC). Pentru a vedea care dintre aceste două forme minime, disjunctivă sau conjunctivă, conduce la o rețea cu cost mai mic trebuie găsite ambele. O metodă simplă de determinare a FMC este aceea a căutării FMD pentru negata funcției date și negarea acesteia. Pentru exemplificare se ia tot funcția dată prin diagrama Karnaugh din figura 2.10. Negata acestei funcții se obține luându-se în considerare compartimentele marcate cu zero. Aplicând această metodă de minimizare pentru funcția $\bar{f}(x_1, x_2, x_3, x_4)$ (v. fig. 2.11), unde $f(x_1, x_2, x_3, x_4)$ este dată de relația (2.39), se obține pentru aceasta forma (2.42) (termenul $\bar{x}_1\bar{x}_2\bar{x}_4$ este redundant):

$$\bar{f}^{\text{FMD}}(x_1, x_2, x_3, x_4) = \bar{x}_2x_3 + x_1x_3 + \bar{x}_1\bar{x}_3\bar{x}_4, \quad (2.42)$$

de unde rezultă prin negare:

$$\bar{\bar{f}}^{\text{FMD}}(x_1, x_2, x_3, x_4) = f^{\text{FMC}}(x_1, x_2, x_3, x_4) = (x_2 + \bar{x}_3)(\bar{x}_1 + \bar{x}_3)(x_1 + x_3 + x_4). \quad (2.43)$$

Diagramele pentru mai mult de patru variabile se construiesc din diagrame de patru variabile, considerate diagrame elementare (v. fig. 2.12). Se pot construi însă diagrame Karnaugh pentru un număr mai mare de variabile și considerând ca diagrame elementare, diagramele de trei variabile. În cazul diagramelor pentru mai mult de patru variabile se consideră vecine, două compartimente și atunci când ocupă aceeași poziție în două diagrame elementare vecine, adică în două diagrame elementare alăturate sau aflate la extre-

mități pe o aceeași linie sau coloană.

Diagramele Karnaugh pentru mai multe variabile se pot construi și sub forma unei singure diagrame elementare dacă se folosesc pentru notarea liniilor și coloanelor coduri binare reflectate (v. fig. 2.12,b). În acest fel se asigură ca oricare două compartimente alăturate sau aflate la extremitățile unei linii sau coloane să fie adiacente.

Având în vedere relațiile (2.33) și algoritmul de trecere de la reprezentarea tabelară a funcției booleene la reprezentarea algebraică (v. algoritmul 1.2), se poate transpună în diagrama Karnaugh și minimizarea funcțiilor date prin FCC. În acest caz se reunesc în diagrama Karnaugh compartimentele notate cu 0 care sunt adiacente, eliminându-se variabilele redundante. Implicanții primi esențiali se obțin sub forma termenilor disjunctivi minimali. În final, se obține FMC realizată cu termenii minimali disjunctivi corespunzători reuniriilor compartimentelor adiacente notate cu zero.

Exemplu. Să se minimizeze, prin metoda diagramelor Karnaugh funcția f dată prin FCC:

$$\begin{aligned} f^{\text{FCC}}(x_1, x_2, x_3, x_4) = & (x_1 + x_2 + x_3 + x_4)(x_1 + \bar{x}_2 + x_3 + x_4)(\bar{x}_1 + x_2 + x_3 + x_4) \times \\ & \times (x_1 + \bar{x}_2 + x_3 + \bar{x}_4)(\bar{x}_1 + \bar{x}_2 + x_3 + \bar{x}_4)(\bar{x}_1 + x_2 + x_3 + \bar{x}_4) \times \\ & \times (x_1 + x_2 + \bar{x}_3 + \bar{x}_4)(x_1 + \bar{x}_2 + \bar{x}_3 + \bar{x}_4)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3 + \bar{x}_4) \times \\ & \times (\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_4). \end{aligned} \quad (2.44)$$

În figura 2.13,a este reprezentată diagrama Karnaugh pentru funcția considerată. Reunind compartimentele adiacente notate cu zero se obține următoarea expresie minimă conjunctivă:

$$f^{\text{FMC}}(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_4)(\bar{x}_1 + x_2 + x_3)(\bar{x}_2 + \bar{x}_4)(\bar{x}_3 + \bar{x}_4). \quad (2.45)$$

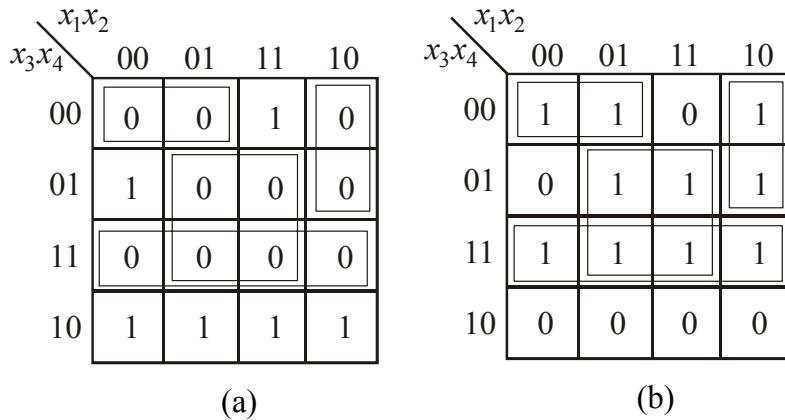


Fig.2.13 Reprezentări ale funcției f din exemplu în diagrame Karnaugh.

Forma minimă conjunctivă se poate obține și dacă se consideră negata FCC a funcției. Evident FCC negată corespunde diagramei Karnaugh complementate (0 se înlocuiește cu 1 și invers). Aplicând diagramei complementate metodologia corespunzătoare FCD, se obține expresia minimă pentru funcția negată. În figura 2.13,b este prezentată diagrama Karnaugh complementată pentru funcția din exemplul dat. Pentru acest caz rezultă:

$$\bar{f}^{\text{FMD}}(x_1, x_2, x_3, x_4) = \bar{x}_1 \bar{x}_3 \bar{x}_4 + x_1 \bar{x}_2 \bar{x}_3 + x_2 x_4 + x_3 x_4. \quad (2.46)$$

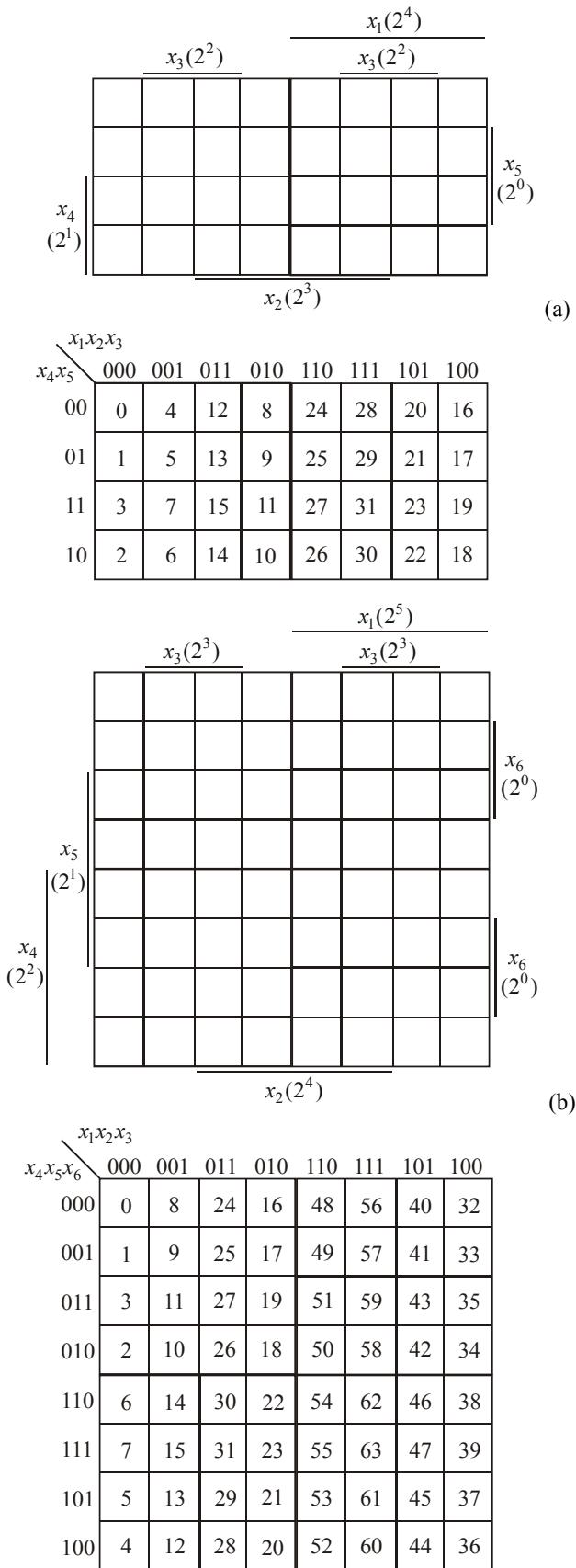


Fig.2.12 Diagramme Karnaugh pentru 5 variabile (a) și pentru 6 variabile (b).

Negând expresia obținută rezultă:

$$f^{\text{FMC}}(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_4)(\bar{x}_1 + x_2 + x_3)(\bar{x}_2 + \bar{x}_4)(\bar{x}_3 + \bar{x}_4). \quad (2.47)$$

Deoarece diagramele Karnaugh sunt cu atât mai ușor de utilizat cu cât numărul variabilelor care definesc compartimentele acestora este mai mic, în practică se folosește adesea posibilitatea de reprezentare a unei funcții de n variabile pe diagrame de $n-1$ sau chiar mai puține variabile. Această tehnică denumită în continuare *condensarea diagrameelor* (sau *metoda diagrameelor VID* – variabile incluse /înglobate în diagramă), se bazează pe ideea reprezentării într-un compartiment al diagramei a unor subcuburi 1-dimensionale, 2-dimensionale, și.a.m.d. Deoarece nu toate funcțiile permit o astfel de reprezentare, fiind necesare uneori și subcuburi de dimensiuni mai mici, se precizează în interiorul compartimentului care dintre subcuburile incluse trebuie luate, folosind variabilele neutilizate la notarea diagramei.

Exemplu. Se dă funcția booleană de trei variabile:

$$f^{\text{FCD}}(x_1, x_2, x_3) = x_1 x_2 \bar{x}_3 + x_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + \bar{x}_1 \bar{x}_2 \bar{x}_3, \quad (2.48)$$

reprezentată pe diagrama din figura 2.14,a și se urmărește reprezentarea ei pe diagrama de două variabile x_1 și x_2 , din figura 2.14,b (trebuie menționat că oricare dintre cele trei variabile ale funcției poate fi inclusă în diagramă). Forma minimă disjunctivă a funcției este :

$$f^{\text{FMD}}(x_1, x_2, x_3) = x_1 x_2 + x_1 x_3 + \bar{x}_1 \bar{x}_2 \bar{x}_3. \quad (2.49)$$

Urmărind pe coloane, valorile funcției înscrise în diagrama din figura 2.14,a se constată:

- pe coloana 1, $x_1 = x_2 = 0$, funcția ia valoarea 1 pentru $x_3 = 0$ și valoarea 0 pentru $x_3 = 1$, prin urmare, în diagrama de două variabile se poate scrie în compartimentul de pe prima coloană expresia \bar{x}_3 , care evaluată dă valoarea 1 numai pentru $x_3 = 0$;
- pe coloana 2, $x_1 = 0$, $x_2 = 1$, funcția ia valoarea 0 independent de valoarea lui x_3 , prin urmare se scrie 0 în diagrama de două variabile;
- pe coloana 3, $x_1 = 1$, $x_2 = 1$, funcția ia valoarea 1 independent de valoarea lui x_3 , prin urmare se scrie 1 în diagrama de două variabile;
- pe coloana 4, $x_1 = 1$, $x_2 = 0$, funcția ia valoarea 0 pentru $x_3 = 0$ și valoarea 1 pentru $x_3 = 1$, prin urmare, în diagrama de două variabile se scrie expresia x_3 .

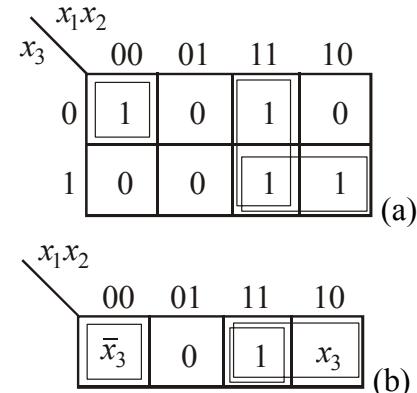


Fig.2.14 Condensarea unei diagrame Karnaugh de trei variabile.

Funcția reprezentată pe o diagramă condensată se poate scrie sub o formă normală disjunctivă exprimând subcuburile corespunzătoare compartimentelor marcate cu 1 prin variabilele cu care se notează diagrama, iar la compartimentele cu variabile sau expresii

adăugând termenului corespunzător compartimentului condiția înscrisă în compartiment. Astfel, funcția reprezentată pe diagrama din figura 2.14,b poate fi exprimată:

$$f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 + x_1 \bar{x}_2 x_3. \quad (2.50)$$

Se pot forma și pe diagrama condensată subcuburi de dimensiuni mai mari prin gruparea compartimentelor vecine, dar folosind unele reguli suplimentare, deoarece în unele cazuri sunt necesare numai părți ale unor subcuburi. De exemplu, în figura 2.14,b o parte a subcubului exprimat prin $x_1 x_2$, împreună cu subcubul exprimat prin $x_1 \bar{x}_2 x_3$, încadrate printr-un contur, dau subcubul exprimat prin $x_1 x_3$ (partea comună a celor două compartimente și condiția).

Metoda de minimizare cu ajutorul diagramelor Karnaugh prezintă dezavantajul că nu poate fi aplicată decât pentru funcții cu un număr relativ mic de variabile (cel mult 7, 8). Pentru minimizarea funcțiilor booleene cu număr mai mare de variabile se recurge la alte metode, algebrice sau tabelare.

2.3.1.4 Minimizarea funcțiilor booleene prin metoda consensurilor

Minimizarea funcțiilor booleene prin metoda Quine-McCluskey precum și prin metodele globale (de exemplu, metoda diagramelor Karnaugh) rezultate din această metodă impune cunoașterea funcțiilor sub una din formele canonice. Această necesitate este determinată de posibilitatea omiterii unor implicanți primi dacă se pleacă de la o formă normală a unei funcții booleene. Rezultă că în cazul funcțiilor booleene date sub formă normală, minimizarea lor este posibilă dacă, în prealabil, sunt aduse la una dintre formele canonice. Acest lucru este posibil prin operații de dezvoltare a formelor normale în forme canonice. Înlăturarea acestui neajuns se poate face bazând minimizarea nu pe compunerea vecinilor ca la metoda Quine-McCluskey, ci pe implicația $ab \rightarrow ax + b\bar{x}$, pe care o vom denumi *implicația consensului*, produsul ab fiind denumit *consensul implicanților* ax și $b\bar{x}$. Să remarcăm faptul că implicația consensului este o generalizare a relației $a \rightarrow ax + a\bar{x}$; într-adevăr, se poate verifica ușor că pentru $a = b$, implicația consensului se reduce la relația de echivalență $a = ax + a\bar{x}$.

O metodă bazată pe implicația consensului a fost dată de Tison [20,23].

Evident, pentru ca implicația $ab \rightarrow ax + b\bar{x}$ să nu fie banală este necesar ca produsul ab să fie nenul, adică a și b să conțină variabile monoforme, înțelegând prin variabile monoforme variabilele care apar sub o singură formă, fie directă, fie negată, atât în a cât și în b , dacă apar în amândouă. Înținând seama de cele arătate rezultă că metoda lui Tison este de fapt o generalizare a metodei Quine-McCluskey.

Noțiunea de consens poate fi la rândul ei generalizată, trecând de la consensul a doi termeni la consensul a n termeni.

Definiție. Un termen a_0 este *consensul termenilor* a_1, a_2, \dots, a_n dacă $a_0 \rightarrow \sum_{i=1}^n a_i$ și dacă $a_0 \not\rightarrow \sum_{i=1}^n a'_i$, unde $\sum_{i=1}^n a'_i$ se obține din $\sum_{i=1}^n a_i$ prin eliminarea, cel puțin, a unui termen oarecare.

Consecință. Consensul termenilor a_1, a_2, \dots, a_n , dacă există, este diferit de zero.

Convenim să facem notația $a_0 = \mathbf{C}\{a_1, a_2, \dots, a_n\}$.

Definiție. Expresia $\sum_{i=1}^n c_i$ se numește *reziduul reuniunii* $\sum_{i=1}^n a_i$ în raport cu a_0 . Termenul c_i este reziduul lui a_i în raport cu a_0 .

Fie $\mathbf{Q} = \{Q_1, Q_2, \dots, Q_r\}$, $\mathbf{Q} \subset \{a_1, a_2, \dots, a_n\}$ mulțimea termenilor a_i care conțin variabila x și $\mathbf{P} = \{P_1, P_2, \dots, P_s\}$, $\mathbf{P} \subset \{a_1, a_2, \dots, a_n\}$, mulțimea termenilor a_i care conțin variabila \bar{x} . Notăm cu Q'_i restul din Q_i după ce s-a înlăturat x și prin P'_i restul din P_i după ce s-a înlăturat \bar{x} .

Definiție. Q'_i , $i \in \mathbf{I}_r$, se vor numi *reziduurile în raport cu x* , iar P'_i , $i \in \mathbf{I}_s$, *reziduurile în raport cu \bar{x}* .

Definiție. Consensurile astfel obținute vor fi denumite *consensi în raport cu x* . Uneori pentru a scoate în evidență acest lucru se va scrie $\mathbf{C}_x\{Q_i, P_j\}$.

Pentru determinarea expresiei unei funcții booleene care să conțină toți implicantii primi, numită și *expresia caracteristică* a funcției respective, se folosește următoarea metodă:

Algoritm 2.5.

1. Se determină mai întâi variabilele biforme și se aranjează într-o ordine arbitrară x_1, x_2, \dots, x_p .

2. Se stabilește tabelul reziduurilor în raport cu x_1 și \bar{x}_1 (se iau în considerare termenii care conțin efectiv pe x_1 și \bar{x}_1).

3. Se face produsul, termen cu termen, al reziduurilor în raport cu x_1 cu reziduurile în raport cu \bar{x}_1 . Termenii diferenți de zero astfel obținuți constituie consensi în raport cu x_1 . Se adaugă aceste consensi la termenii inițiali și după ce se elimină termenii care se absorb, rămâne o nouă mulțime de implicantă la care se aplică pașii 2 și 3 în raport cu o altă variabilă din mulțimea x_1, x_2, \dots, x_p .

4. Procesul începează după aplicarea pașilor 2 și 3 la toate variabilele biforme x_1, x_2, \dots, x_p .

Exemplu. Să se determine expresia care conține toți implicantii primi ai funcției:

$$f = x_1x_5 + x_2x_6 + x_3\bar{x}_7 + x_4\bar{x}_5\bar{x}_6x_7. \quad (2.51)$$

1. Variabilele biforme sunt x_5 , x_6 și x_7 .

2.1. Reziduul în raport cu x_5 este x_1 . Reziduul în raport cu \bar{x}_5 este $x_4\bar{x}_6x_7$.

3.1. Consensul în raport cu x_5 este $x_1x_4\bar{x}_6x_7$. Noua expresie a lui f este:

$$f = x_1x_5 + x_2x_6 + x_3\bar{x}_7 + x_4\bar{x}_5\bar{x}_6x_7 + x_1x_4\bar{x}_6x_7. \quad (2.52)$$

2.2. Reziduul în raport cu x_6 este x_2 . Reziduurile în raport cu \bar{x}_6 sunt $x_4\bar{x}_5x_7$, $x_1x_4x_7$.

3.2. Consensurile în raport cu x_6 sunt $x_2x_4\bar{x}_5x_7$, $x_1x_2x_4x_7$. Noua expresie a lui f este:

$$f = x_1x_5 + x_2x_6 + x_3\bar{x}_7 + x_4\bar{x}_5\bar{x}_6x_7 + x_1x_4\bar{x}_6x_7 + x_1x_2x_4x_7 + x_2x_4\bar{x}_5x_7. \quad (2.53)$$

2.3. Reziduurile în raport cu x_7 sunt $x_4\bar{x}_5\bar{x}_6$, $x_1x_4\bar{x}_6$, $x_1x_2x_4$, $x_2x_4\bar{x}_5$. Reziduul în raport cu \bar{x}_7 este x_3 .

3.3. Consensurile în raport cu x_7 sunt $x_3x_4\bar{x}_5\bar{x}_6$, $x_1x_3x_4\bar{x}_6$, $x_1x_2x_3x_4$, $x_2x_3x_4\bar{x}_5$. Noua expresie a lui f este:

$$\begin{aligned} f = & x_1x_5 + x_2x_6 + x_3\bar{x}_7 + x_4\bar{x}_5\bar{x}_6x_7 + x_1x_4\bar{x}_6x_7 + x_1x_2x_4x_7 + x_2x_4\bar{x}_5x_7 + \\ & + x_3x_4\bar{x}_5\bar{x}_6 + x_1x_3x_4\bar{x}_6 + x_1x_2x_3x_4 + x_2x_3x_4\bar{x}_5, \end{aligned} \quad (2.54)$$

care este expresia caracteristică a funcției respective.

Pentru obținerea expresiilor minimale nu se mai poate aplica algoritmul prezentat la metoda Quine-McCluskey, deoarece funcția nu este scrisă sub forma normală disjunctivă, ca reuniune a tuturor constituenților unității pe care-i cuprinde. Pentru eliminarea acestui inconvenient se va prezenta o metodă bazată pe teoria consensurilor. Înainte de aceasta se vor prezenta relațiile și implicațiile de minimalitate.

Definiție. Fie $\mathbf{P} = \{P_1, P_2, \dots, P_r\}$ mulțimea implicantilor primi ai funcției f și $\mathbf{Q} = \{Q_1, Q_2, \dots, Q_s\}$ o submulțime a lui \mathbf{P} cu proprietatea că $f = \sum_{i=1}^s Q_i$, $f \not\rightarrow \sum_{i=1}^s' Q_i$, unde $\sum_{i=1}^s' Q_i$ se obține din $\sum_{i=1}^s Q_i$ prin eliminarea cel puțin a unui Q_i . Vom numi aceste relații *relațiile de minimalitate*.

Definiție. O mulțime \mathbf{Q} , ale cărei elemente îndeplinesc relațiile de minimalitate, se spune că are *proprietatea de minimalitate*.

Prin minimizarea funcției f se înțelege determinarea expresiilor care îndeplinesc relațiile de minimalitate.

Lemă. Relațiile de minimalitate sunt echivalente cu următoarele implicații:

$$P_j \rightarrow \sum_{i=1}^s Q_i, (\forall) j \in I_r, (\exists) j \in I_r, P_j \not\rightarrow \sum_{i=1}^s' Q_i. \quad (2.55)$$

Definiție. Implicațiile din lemă vor fi denumite *implicații de minimalitate*.

Algoritmul de realizare minimală este următorul:

Algoritm 2.6.

1. Se aduce funcția f la forma caracteristică f^c , de reuniune a tuturor implicantilor săi primi. Fie \mathbf{P} mulțimea implicantilor primi ai funcției f .

2. Se aplică algoritmul de determinare a consensurilor la funcția caracteristică sub următoarea formă, puțin modificată:

Fie P_1, P_2, \dots, P_n termenii expresiei caracteristice f^c .

2.1. Se determină variabilele biforme. Fie acestea x_1, x_2, \dots, x_p . Se face $x = x_1$.

2.2. Se determină consensurile în raport cu x . Printre aceste consensi se vor găsi termeni din f^c sau consensi noi. Fiecare consens în raport cu x , astfel determinat, se marchează cu indicii termenilor din f^c .

Dacă, de exemplu, $P_k = \mathbf{C}(P_i, P_j)$, atunci marcajul lui P_k va fi perechea (i, j) . În general, dacă $P_k = \mathbf{C}(P_{i_1}, P_{i_2}, \dots, P_{i_k})$, atunci marcajul va fi (i_1, i_2, \dots, i_k) . La fel se procedează cu consensurile noi.

Tinând seama că f^c conține toți implicantii primi ai funcției f , consensurile noi vor fi implicantii ai termenilor din f^c , dar nu vor fi înlăturați, întrucât folosesc la marcarea consensurilor la pașii următori.

2.3. Se adaugă consensurile noi la termenii din f^c .

2.4. Dacă $x = x_p$, se trece la pasul 3, astfel, se face $x = x_{i+1}$ și se trece la pasul 2.2.

3. Se asociază termenilor funcției f^c variabile booleene. Fie y_i variabila asociată termenului P_i , $i = 1, 2, \dots, n$. Marcajului (i_1, i_2, \dots, i_k) îi se asociază produsul $S_i = y_{i_1} y_{i_2} \dots y_{i_k}$.

4. Se construiesc expresiile $E_i = y_i + \sum_{j=1}^{n_i} S_i^j$, unde S_i^j corespunde marcajului j asociat implicantului prim P_i , în ipoteza că există mai multe marcaje iar n_i este numărul acestor marcaje.

5. Notăm prin $E_i^{(1)}$, $i = 1, 2, \dots, n$, expresia obținută în pasul 4.

6. Se construiesc expresiile $E_i^{(2)}$, $i = 1, 2, \dots, n$, din $E_i^{(1)}$, prin substituire, după următoarea regulă. Dacă în $E_i^{(1)}$ apare y_k , atunci în $E_i^{(1)}$ se înlocuiește y_k prin $E_k^{(1)}$, pentru orice k pentru care $y_k \in E_i^{(1)}$. Expresia lui $E_i^{(1)}$ obținută după aceste substituții se va nota prin $E_i^{(2)}$. La fel se procedează cu toate expresiile $E_i^{(1)}$, $i = 1, 2, \dots, n$.

7. În general, se obțin expresiile $E_i^{(k+1)}$ din $E_i^{(k)}$ la fel cum s-au obținut expresiile $E_i^{(2)}$ din $E_i^{(1)}$. Procesul începează atunci când $E_i^{(k)} = E_i^{(k+1)}$ pentru orice i .

8. Se construiește expresia $E = \prod_{i=1}^n E_i$ și se aduce la forma caracteristică. Implicantii de lungime ai lui E corespund expresiilor minimale astfel: dacă $y_1 y_2 \dots y_p$ este un implicant prim de lungime minimă a lui E , atunci expresia $f_1 = \sum_{i=1}^p P_i$ este expresia minimă a lui f , unde P_i corespunde variabilelor y_i , $i = 1, 2, \dots, p$. În general, pot exista mai multe expresii minime și un număr oarecare de expresii reduse.

Exemplu.

$$f^c = x_1 x_3 + x_2 x_3 + x_3 \bar{x}_4 + x_1 x_4 + x_2 x_4 + \bar{x}_3 x_4 + \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 \bar{x}_4. \quad (2.56)$$

Se fac notațiile $P_1 = x_1 x_3$, $P_2 = x_2 x_3$, $P_3 = x_3 \bar{x}_4$, $P_4 = x_1 x_4$, $P_5 = x_2 x_4$, $P_6 = \bar{x}_3 x_4$, $P_7 = \bar{x}_1 \bar{x}_2 \bar{x}_3$, $P_8 = \bar{x}_1 \bar{x}_2 \bar{x}_4$. Variabilele biforme sunt x_1, x_2, x_3, x_4 .

1. Consensurile în raport cu x_1 sunt date de produsul:

$$(x_3 + x_4)(\bar{x}_2 \bar{x}_3 + \bar{x}_2 \bar{x}_4) = \bar{x}_2 x_3 \bar{x}_4 + \bar{x}_2 \bar{x}_3 x_4. \quad (2.57)$$

Se adaugă consensurile $P_9 = \bar{x}_2x_3\bar{x}_4$ și $P_{10} = \bar{x}_2\bar{x}_3x_4$, cu marcajele (1,8) și respectiv (4,7).

2. Consensurile în raport cu x_2 sunt date de produsul:

$$(x_3 + x_4)(\bar{x}_1\bar{x}_3 + \bar{x}_1\bar{x}_4 + x_3\bar{x}_4 + \bar{x}_3x_4) = x_3\bar{x}_4 + \bar{x}_3x_4. \quad (2.58)$$

Consensurile $P_3 = x_3\bar{x}_4$ și $P_6 = \bar{x}_3x_4$ se marchează prin $(2,9) \rightarrow (1,2,8)$ și respectiv $(5,10) \rightarrow (4,5,7)$.

3. Consensurile în raport cu x_3 sunt date de produsul:

$$(x_1 + x_2 + \bar{x}_4)(x_4 + \bar{x}_1\bar{x}_2) = x_1x_4 + x_2x_4 + \bar{x}_1\bar{x}_2\bar{x}_4. \quad (2.59)$$

Consensurile $P_4 = x_1x_4$, $P_5 = x_2x_4$ și $P_8 = \bar{x}_1\bar{x}_2\bar{x}_4$, primesc marcajele (1,6), (2,6) și respectiv (3,7).

4. Consensurile în raport cu x_4 sunt date de produsul:

$$(x_1 + x_2 + \bar{x}_3)(x_3 + \bar{x}_1\bar{x}_2) = x_1x_3 + x_2x_3 + \bar{x}_1\bar{x}_2\bar{x}_3. \quad (2.60)$$

Consensurile $P_1 = x_1x_3$, $P_2 = x_2x_3$ și $P_7 = \bar{x}_1\bar{x}_2\bar{x}_3$, primesc marcajele (3,4), (3,5) și respectiv (6,8).

Expresiile E_i asociate implicantilor primi P_i , $i = 1, 2, \dots, 8$ sunt:

$$E_1 = y_1 + y_3y_4, E_2 = y_2 + y_3y_5, E_3 = y_3 + y_1y_2y_8, E_4 = y_4 + y_1y_6, \quad (2.61)$$

$$E_5 = y_5 + y_2y_6, E_6 = y_6 + y_4y_5y_7, E_7 = y_7 + y_6y_8, E_8 = y_8 + y_3y_7. \quad (2.62)$$

Întrucât nu se pot face substituiri, se scrie E :

$$\begin{aligned} E &= \prod_{i=1}^8 E_i = (y_1 + y_3y_4)(y_2 + y_3y_5)(y_3 + y_1y_2y_8)(y_4 + y_1y_6) \times \\ &\quad \times (y_5 + y_2y_6)(y_6 + y_4y_5y_7)(y_7 + y_6y_8)(y_8 + y_3y_7) = \\ &= y_1y_2y_6y_8 + y_3y_4y_5y_7 + y_1y_3y_5y_6y_7 + y_1y_3y_5y_6y_8 + y_1y_2y_4y_5y_7y_8. \end{aligned} \quad (2.63)$$

Există deci două expresii minimale:

$$f_1 = x_1x_3 + x_2x_3 + \bar{x}_3x_4 + \bar{x}_1\bar{x}_2\bar{x}_4, \quad (2.64)$$

$$f_2 = x_3\bar{x}_4 + x_1x_4 + x_2x_4 + \bar{x}_1\bar{x}_2\bar{x}_3 \quad (2.65)$$

și alte trei expresii reduse:

$$f_3 = x_1x_3 + x_3\bar{x}_4 + x_2x_4 + x_3x_4 + \bar{x}_1\bar{x}_2\bar{x}_3, \quad (2.66)$$

$$f_4 = x_1x_3 + x_3\bar{x}_4 + x_2x_4 + \bar{x}_3x_4 + \bar{x}_1\bar{x}_2\bar{x}_4, \quad (2.67)$$

$$f_5 = x_1x_3 + x_2x_3 + x_1x_4 + x_2x_4 + \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2\bar{x}_4. \quad (2.68)$$

Exemplu.

$$\begin{aligned} f^c = & x_1x_5 + x_2x_6 + x_3\bar{x}_7 + x_4\bar{x}_5\bar{x}_6x_7 + x_1x_4\bar{x}_6x_7 + x_2x_4\bar{x}_5x_7 + x_1x_2x_4x_7 + \\ & + x_3x_4\bar{x}_5\bar{x}_6 + x_1x_3x_4\bar{x}_6 + x_2x_3x_4\bar{x}_5 + x_1x_2x_3x_4. \end{aligned} \quad (2.69)$$

Se fac notațiile $P_1 = x_1x_5$, $P_2 = x_2x_6$, $P_3 = x_3\bar{x}_7$, $P_4 = x_4\bar{x}_5\bar{x}_6x_7$, $P_5 = x_1x_4\bar{x}_6x_7$, $P_6 = x_2x_4\bar{x}_5x_7$, $P_7 = x_1x_2x_4x_7$, $P_8 = x_3x_4\bar{x}_5\bar{x}_6$, $P_9 = x_1x_3x_4\bar{x}_6$, $P_{10} = x_2x_3x_4\bar{x}_5$, $P_{11} = x_1x_2x_3x_4$.

Variabilele biforme sunt x_5, x_6, x_7 .

Consensurile în raport cu x_5 sunt date de produsul:

$$\begin{aligned} x_1(x_4\bar{x}_6x_7 + x_2x_4x_7 + x_3x_4\bar{x}_6 + x_2x_3x_4) = \\ = x_1x_4\bar{x}_6x_7 + x_1x_2x_4x_7 + x_1x_3x_4\bar{x}_6 + x_1x_2x_3x_4. \end{aligned} \quad (2.70)$$

Consensurile P_5 , P_7 , P_9 și P_{11} se marchează prin (1,4), (1,6), (1,8) și respectiv (1,10).

Consensurile în raport cu x_6 sunt date de produsul:

$$\begin{aligned} x_2(x_4\bar{x}_5x_7 + x_1x_4x_7 + x_3x_4\bar{x}_5 + x_1x_3x_4) = \\ = x_2x_4\bar{x}_5x_7 + x_1x_2x_4x_7 + x_2x_3x_4\bar{x}_5 + x_1x_2x_3x_4. \end{aligned} \quad (2.71)$$

Consensurile P_6 , P_7 , P_{10} și P_{11} se marchează prin (2,4), (2,5), (2,8) și respectiv (2,9).

Consensurile în raport cu x_7 sunt date de produsul:

$$\begin{aligned} x_3(x_4\bar{x}_5\bar{x}_6 + x_1x_4\bar{x}_6 + x_2x_4\bar{x}_5 + x_1x_2x_4) = \\ = x_3x_4\bar{x}_5\bar{x}_6 + x_1x_3x_4\bar{x}_6 + x_2x_3x_4\bar{x}_5 + x_1x_2x_3x_4. \end{aligned} \quad (2.72)$$

Consensurile P_8 , P_9 , P_{10} și P_{11} se marchează prin (3,4), (3,5), (3,6) și respectiv (3,7).

Consensurile P_1 , P_2 , P_3 și P_4 rămân nemarcate.

Expresiile E_i aplicate implicantilor primi sunt:

$$E_1 = y_1, E_2 = y_2, E_3 = y_3, E_4 = y_4, E_5 = y_5 + y_1y_4, E_6 = y_6 + y_2y_4, \quad (2.73)$$

$$E_7 = y_7 + y_1y_6 + y_2y_5, E_8 = y_8 + y_3y_4, E_9 = y_9 + y_1y_8 + y_3y_5, \quad (2.74)$$

$$E_{10} = y_{10} + y_2y_8 + y_3y_6, E_{11} = y_{11} + y_1y_{10} + y_2y_9 + y_3y_7. \quad (2.75)$$

După substituții, expresiile E_i , $i = 1, \dots, 11$, devin:

$$\begin{aligned}
 E_1 &= y_1, E_2 = y_2, E_3 = y_3, E_4 = y_4, E_5 = y_5 + y_1y_4, \\
 E_6 &= y_6 + y_2y_4, E_7 = y_7 + y_1y_6 + y_2y_5 + y_1y_2y_4, E_8 = y_8 + y_3y_4, \\
 E_9 &= y_9 + y_1y_8 + y_3y_5 + y_1y_3y_4, E_{10} = y_{10} + y_2y_8 + y_3y_6 + y_2y_3y_4, \\
 E_{11} &= y_{11} + y_1y_{10} + y_2y_9 + y_3y_7 + y_1y_2y_8 + y_1y_3y_6 + y_2y_3y_6 + y_2y_3y_5 + y_1y_2y_3y_4.
 \end{aligned} \tag{2.76}$$

Se verifică imediat că:

$$E = \prod_{i=1}^{11} E_i = y_1y_2y_3y_4 \tag{2.77}$$

și deci există o singură expresie minimă:

$$f_1 = x_1x_5 + x_2x_6 + x_3\bar{x}_7 + x_4\bar{x}_5\bar{x}_6x_7. \tag{2.78}$$

2.3.1.5 Minimizarea funcțiilor booleene incomplet definite

Minimizarea funcțiilor booleene incomplet definite are mare importanță deoarece cea mai mare parte a comenziilor discrete conțin situații de nedefinire. Luarea în considerare a stărilor de nedefinire (nedeterminare), prin atribuirea de valori 0 sau 1 funcției, conduce în general la forme minime mai economice decât în cazul neconsiderării lor. Folosirea diferită a stărilor indiferente la obținerea formelor minime disjunctivă și conjunctivă conduce la rezultate diferite. Din acest motiv, în cazul funcțiilor incomplet definite este bine să se minimizeze în ambele forme și să se păstreze forma cea mai avantajoasă. Sinteză optimă a unei rețele de comutare a cărei comportare este dată printr-o funcție booleană parțial sau incomplet definită se face considerând că rețeaua trebuie să realizeze acea funcție, din clasa funcțiilor booleene corespondente funcției incomplet definite, care are cea mai simplă FMD sau FMC.

Minimizarea funcțiilor booleene incomplet definite se poate face cu ajutorul oricărereia dintre metodele de minimizare prezentate anterior dar aplicate cu o serie de modificări. Astfel, pentru a se obține FMD a acestei funcții, în cazul folosirii metodei de minimizare cu ajutorul diagramelor Karnaugh, se procedează în felul următor:

Regulă.

1. Se reprezintă funcția booleană dată în diagramă, notând cu 1 pozițiile corespunzătoare n -uplelor pentru care valoarea funcției este 1, cu 0 pozițiile corespunzătoare n -uplelor pentru care valoarea funcției este 0 și cu d sau asterisc n -uplele pentru care valoarea funcției nu este precizată.

2. Se încearcă gruparea compartimentelor notate cu 1 și formarea de subcuburi de dimensiuni cât mai mari, folosindu-se în acest scop, dacă ajută la realizarea acestui obiectiv și compartimentele notate cu d (*), considerându-le marcate cu 1.

3. Se procedează în continuare ca și la minimizarea funcției booleene complet definite cu observația că reprezintă implicații primi esențiali numai acele subcuburi care conțin cel puțin un compartiment notat cu 1 care nu mai este inclus într-un alt subcub (subcuburile care conțin însă un compartiment notat cu d (*), care nu este conținut în alte sub-

cuburi nu reprezintă un implicant prim esențial). În continuare se caută o acoperire minimă a termenilor canonici neinclusi în implicantii primi esențiali.

Exemplu. Se cere găsirea FMD care corespunde funcției booleene incomplet definite dată prin tabelul 2.5. Se urmărește un cost C_P minim.

Tab.2.5 Tabelul de adevăr al funcției booleene incomplet definite din exemplu.

x_1	x_2	x_3	x_4	f	x_1	x_2	x_3	x_4	f
0	0	0	0	1	1	0	0	0	1
0	0	0	1	1	1	0	0	1	0
0	0	1	0	1	1	0	1	0	d
0	0	1	1	d	1	0	1	1	1
0	1	0	0	d	1	1	0	0	0
0	1	0	1	1	1	1	0	1	0
0	1	1	0	0	1	1	1	0	d
0	1	1	1	d	1	1	1	1	1

Această funcție este descrisă de relațiile (2.79) și (2.80):

$$f^{\text{FCD}}(x_1, x_2, x_3, x_4) = P_0 + P_1 + P_2 + P_5 + P_8 + P_{11} + P_{15}, \quad (2.79)$$

$$\bar{x}_1\bar{x}_2x_3x_4 = \bar{x}_1x_2\bar{x}_3\bar{x}_4 = \bar{x}_1x_2x_3x_4 = x_1\bar{x}_2x_3\bar{x}_4 = x_1x_2x_3\bar{x}_4 = d, \quad (2.80)$$

unde, relația (2.79) reprezintă forma canonica disjunctivă a părții definite din funcție, iar relația (2.80) precizează că o parte dintre combinațiile de valori ale variabilelor de intrare, corespunzătoare părții nedefinite din funcție, nu apar în funcționare, prin urmare pot fi considerate combinații indiferente.

Pentru minimizarea acestei funcții se întocmește diagrama din figura 2.15,a. Din această diagramă rezultă că termenul $\bar{x}_2\bar{x}_4$ este implicant prim esențial. Celelalte unități din diagramă neincluse în acest implicant pot fi acoperite considerând implicantii $\bar{x}_1\bar{x}_3$ sau \bar{x}_1x_4 și x_3x_4 sau x_1x_3 . Funcția are deci patru FMD ce pot fi scrise prescurtat ca mai jos:

$$f^{\text{FMD}}(x_1, x_2, x_3, x_4) = \bar{x}_2\bar{x}_4 + \left\{ \begin{array}{l} \bar{x}_1\bar{x}_3 \\ \bar{x}_1x_4 \end{array} \right\} + \left\{ \begin{array}{l} x_3x_4 \\ x_1x_3 \end{array} \right\}. \quad (2.81)$$

Pentru obținerea FMC a funcției date se consideră zerourile din diagramă împreună cu compartimentele notate cu d , în scopul obținerii FMD a negatei funcției, care apoi prin negare dă FMC a funcției. Obținerea FMD a negatei funcției (2.79), (2.80) se face pe diagrama din figura 2.15,b. Din figura 2.15,b rezultă forma minimă a negatei funcției, dată de relația:

$$\bar{f}^{\text{FMD}}(x_1, x_2, x_3, x_4) = x_2 \bar{x}_4 + x_1 \bar{x}_3 x_4, \quad (2.82)$$

de unde rezultă:

$$f^{\text{FMC}}(x_1, x_2, x_3, x_4) = (\bar{x}_2 + x_4)(\bar{x}_1 + x_3 + \bar{x}_4). \quad (2.83)$$

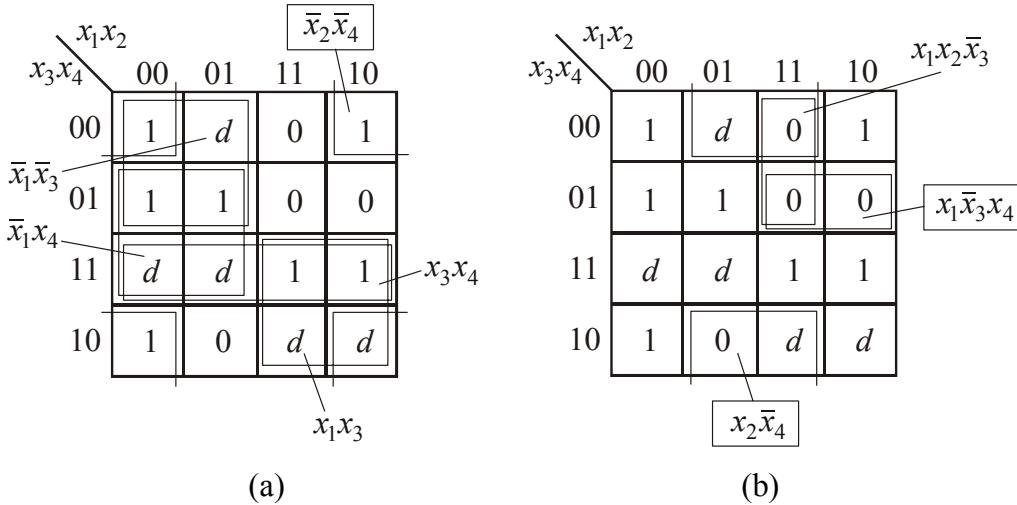


Fig.2.15 Minimizarea funcției booleene incomplet definite din exemplu.

Prin urmare, cea mai avantajoasă formă minimă a funcției este forma minimă conjunctivă.

Și în cazul în care minimizarea funcțiilor incomplet definite se face folosind metoda Quine-McCluskey, apar față de cazul aplicării acestei metode funcțiilor complet definite, o serie de mici modificări care vor fi ilustrate prin exemplul tratat mai jos.

Exemplu. Se cere să se găsească FMD a funcției booleene incomplet definite, $f(x_1, x_2, x_3, x_4, x_5)$ dată prin relațiile (2.84), (2.85), respectiv prin tabelul 2.6, folosind metoda Quine-McCluskey.

Așa cum s-a mai precizat, funcția este descrisă de următoarele relații:

$$f^{\text{FCD}}(x_1, x_2, x_3, x_4, x_5) = P_3 + P_4 + P_5 + P_9 + P_{11} + P_{20} + P_{21} + P_{28} + P_{29}, \quad (2.84)$$

$$P_{12} = P_{13} = P_{14} = P_{15} = P_{19} = P_{23} = P_{27} = P_{31} = *. \quad (2.85)$$

Se procedează la fel ca și în cazul funcțiilor complet definite, împărțind echivalenții binari ai termenilor canonici ai funcției în grupe, în raport de numărul de unități pe care aceștia îi conțin (v. tab. 2.7). Termenii indiferenți sunt introduși și ei în tabel în grupele corespunzătoare, pe același principiu, dar se marchează de la început cu un semn distinct, de exemplu, cu un asterisc. Este necesar un marcat special pentru a se putea distinge și implicații de „clasă superioară” obținuți numai din termeni indiferenți. Și aceștia se marchează, deja în momentul formării, cu asterisc. Implicații formați prin gruparea unor termeni indiferenți, cu termenii funcției aplicați în 1, se consideră implicații obișnuite. Implicații obișnuite acoperiți de către implicații de „clasă superioară” formați în urma grupării, se marchează prin bifare. Toți implicații rămași nemarcați în urma încheierii procedurii de grupare sunt implicații primi ai funcției. În tabelul 2.7 aceștia au fost

notații simbolic cu literele a, b, c, d, e și f .

Tab.2.6 Tabelul de adevăr al funcției booleane incomplet definite din exemplu.

	x_1	x_2	x_3	x_4	x_5	f		x_1	x_2	x_3	x_4	x_5	f
0	0	0	0	0	0	0	16	1	0	0	0	0	0
1	0	0	0	0	1	0	17	1	0	0	0	1	0
2	0	0	0	1	0	0	18	1	0	0	1	0	0
3	0	0	0	1	1	1	19	1	0	0	1	1	*
4	0	0	1	0	0	1	20	1	0	1	0	0	1
5	0	0	1	0	1	1	21	1	0	1	0	1	1
6	0	0	1	1	0	0	22	1	0	1	1	0	0
7	0	0	1	1	1	0	23	1	0	1	1	1	*
8	0	1	0	0	0	0	24	1	1	0	0	0	0
9	0	1	0	0	1	1	25	1	1	0	0	1	0
10	0	1	0	1	0	0	26	1	1	0	1	0	0
11	0	1	0	1	1	1	27	1	1	0	1	1	*
12	0	1	1	0	0	*	28	1	1	1	0	0	1
13	0	1	1	0	1	*	29	1	1	1	0	1	1
14	0	1	1	1	0	*	30	1	1	1	1	0	0
15	0	1	1	1	1	*	31	1	1	1	1	1	*

Înțocmind tabelul 2.8 se constată că implicantii primi a, b și c sunt esențiali și că ei acoperă toți termenii funcției aplicați în 1 (termenii canonici). Prin urmare, FMD a funcției este:

$$f^{\text{FMD}}(x_1, x_2, x_3, x_4, x_5) = x_3\bar{x}_4 + \bar{x}_3x_4x_5 + \bar{x}_1x_2x_5. \quad (2.86)$$

Tab.2.8 Tabelul acoperirilor pentru funcția f .

Implicantii primi		Termeni canonici								
Notatie simbolica	Indici	3	4	5	9	11	20	21	28	29
a	4,5,12,13,20,21,28,29	*	*	*		*	*	*	*	*
b	3,11,19,27	*			*	*				
c	9,11,13,15				*	*				
d	11,15,27,31				*			*		*
e	21,23,29,31						*			*
f	13,15,29,31								*	*

CIRCUITE LOGICE COMBINATORIALE

Tab.2.7 Tabelul implicantilor primi pentru functia f .

Subcuburi 0-dimensionale							Subcuburi 1-dimensionale					Subcuburi 2-dimensionale					Subcuburi 3-dimensionale													
Grupa	Indici	x_1	x_2	x_3	x_4	x_5	Grupa	Indici	x_1	x_2	x_3	x_4	x_5	Grupa	Indici	x_1	x_2	x_3	x_4	x_5	Grupa	Indici	x_1	x_2	x_3	x_4	x_5			
1	4	0	0	1	0	0	✓	1	4,5	0	0	1	0	—	✓	1	4,5,20,21	—	0	1	0	—	✓	1	4,5,20,21,	—	—	1	0	— (a)
2	3	0	0	0	1	1	✓		4,20	—	0	1	0	0	✓		4,5,12,13	0	—	1	0	—	✓							
	5	0	0	1	0	1	✓		4,12	0	—	1	0	0	✓		4,20,12,28	—	—	1	0	0	✓							
	9	0	1	0	0	1	✓	2	3,11	0	—	0	1	1	✓	2	3,11,19,27	—	—	0	1	1	(b)							
	20	1	0	1	0	0	✓		3,19	—	0	0	1	1	✓		5,21,13,29	—	—	1	0	1	✓							
	12*	0	1	1	0	0	✓		5,21	—	0	1	0	1	✓		9,11,13,15	0	1	—	—	1	(c)							
3	11	0	1	0	1	1	✓		5,13	0	—	1	0	1	✓		20,21,28,29	1	—	1	0	—	✓							
	21	1	0	1	0	1	✓		9,11	0	1	0	—	1	✓		12,28,13,29	—	1	1	0	—	✓							
	28	1	1	1	0	0	✓		9,13	0	1	—	0	1	✓		12,13,14,15*	0	1	1	—	—								
	13*	0	1	1	0	1	✓		20,21	1	0	1	0	—	✓	3	11,15,27,31	—	1	—	1	1	(d)							
	14*	0	1	1	1	0	✓		20,28	1	—	1	0	0	✓		21,29,23,31	1	—	1	—	1	(e)							
	19*	1	0	0	1	1	✓		12,28	—	1	1	0	0	✓		13,29,15,31	—	1	1	—	1	(f)							
4	29	1	1	1	0	1	✓		12,13*	0	1	1	0	—	✓		19,23,27,31*	1	0	0	1	1								
	15*	0	1	1	1	1	✓		12,14*	0	1	1	—	0	✓															
	23*	1	0	1	1	1	✓	3	11,15	0	1	—	1	1	✓															
	27*	1	1	0	1	1	✓		11,27	—	1	0	1	1	✓															
5	31*	1	1	1	1	1	✓		21,29	1	—	1	0	1	✓															
									21,23	1	0	1	—	1	✓															
									28,29	1	1	1	0	—	✓															
									13,29	—	1	1	0	1	✓															
									13,15*	0	1	1	—	1	✓															
									14,15*	0	1	1	1	—	✓															
									19,23*	1	0	—	1	1	✓															
									19,27*	1	—	0	1	1	✓															
									4	29,31	1	1	1	—	1	✓														
									15,31*	—	1	1	1	1	✓															
									23,31*	1	—	1	1	1	✓															
									27,31*	1	1	—	1	1	✓															

2.3.1.6 Minimizarea sistemelor de funcții booleene

Comportarea unui circuit de comutare combinațional cu n intrări și m ieșiri este descrisă de un sistem de m funcții booleene de n variabile. Pentru a se realiza un astfel de circuit cu un număr minim de circuite de comutație elementare de un anumit tip, care să fie la rândul lor cât mai simple, nu este suficient să se minimizeze fiecare funcție în parte, ci trebuie să se aducă întreg sistemul de funcții la o formă minimă printr-o minimizare corelată a funcțiilor acestuia.

Formele minime pentru un sistem de funcții booleene sunt acele expresii booleene disjunctive sau conjunctive în care apare un număr minim de termeni respectiv factori normali diferenți, având un număr minim de variabile. Acestor forme le corespunde o rețea de comutare cu două niveluri cu număr minim de elemente logice, prin urmare cu cost minim. Pentru obținerea formelor minime ale unui sistem de funcții booleene dat, se procedează, aşa cum s-a mai precizat, la minimizarea corelată a funcțiilor acestuia, adică la determinarea setului minim de implicantă primă care acoperă (inclusiv) toți termenii canonici ai tuturor funcțiilor sistemului [22,23].

Una dintre metodele de minimizare corelată a mai multor funcții booleene, f_1, f_2, \dots, f_n , se bazează pe determinarea implicantelor primă ai funcțiilor f_1, f_2, \dots, f_n și ai funcțiilor produs $f_1 \cdot f_2, f_1 \cdot f_3, \dots, f_{n-1} \cdot f_n, f_1 \cdot f_2 \cdot f_3, f_1 \cdot f_2 \cdot f_4, \dots, f_{n-2} \cdot f_{n-1} \cdot f_n, \dots, f_1 \cdot f_2 \cdot f_3 \cdot \dots \cdot f_n$. Având acest set de implicantă primă, se calculează acoperirile posibile pentru fiecare dintre funcții iar apoi se alege cea mai avantajoasă combinație de acoperiri din punct de vedere al costului, care reprezintă acoperirea minimală a sistemului.

Pentru obținerea acoperirii minime după această metodă se parcurg următoarele etape [22]:

Regulă.

1. Se calculează funcțiile produs. De exemplu, se cere minimizarea sistemului de funcții:

$$\begin{aligned} f_1^{\text{FCD}}(x_1, x_2, x_3) &= \sum(1,5,6,7), \\ f_2^{\text{FCD}}(x_1, x_2, x_3) &= \sum(1,4,5,6), \\ f_3^{\text{FCD}}(x_1, x_2, x_3) &= \sum(0,2,5,6,7), \end{aligned} \tag{2.87}$$

unde, sub semnul sumei booleene s-au dat indicii zecimale ai termenilor canonici prezenți în FCD a fiecărei funcții.

În prima etapă se calculează funcțiile produs:

$$\begin{aligned} f_1 \cdot f_2 &= \sum(1,5,6), \quad f_1 \cdot f_3 = \sum(5,6,7), \\ f_2 \cdot f_3 &= \sum(5,6), \quad f_1 \cdot f_2 \cdot f_3 = \sum(5,6). \end{aligned} \tag{2.88}$$

Funcțiile (2.88) conțin termenii canonici comuni pentru cele două, respectiv trei funcții, făcându-se produsul logic.

2. Se determină implicantă primă ai fiecărei dintre funcțiile (2.87) și (2.88).

În cazul exemplului considerat, pentru determinarea implicantilor primi ai acestor functii se folosesc diagramele Karnaugh, prezентate în figura 2.16. În același scop se pot folosi și metodele Quine-McCluskey respectiv a consensurilor.

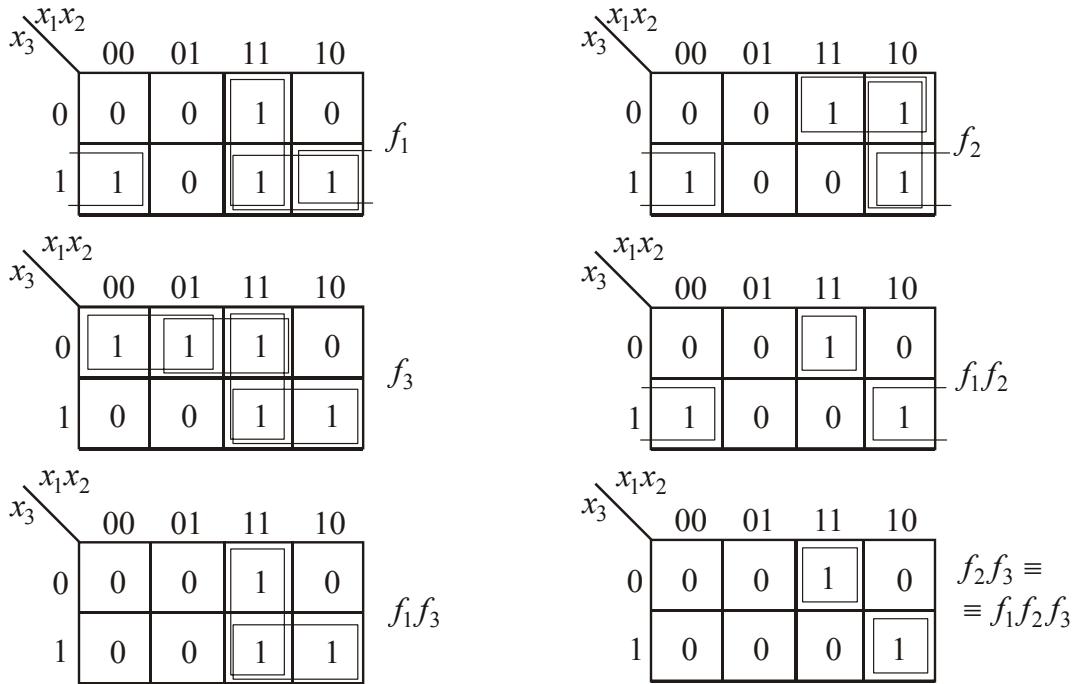


Fig.2.16 Minimizarea sistemelor de funcții booleene.

Din figura 2.16 se obțin, pentru funcțiile considerate, implicantii primi din tabelul 2.9.

Tab.2.9 Tabelul implicantilor primi pentru sistemul de funcții din exemplu.

Funcția	Implicantii primi			Funcția	Implicantii primi		
	Indicii	Expresia	Notăția		Indicii	Expresia	Notăția
f_1	1,5	\bar{x}_2x_3	-	f_3	6,7	x_1x_2	-
	6,7	x_1x_2	-		5,7	x_1x_3	-
	5,7	x_1x_3	-		1,5	\bar{x}_2x_3	e
f_2	1,5	\bar{x}_2x_3	-	$f_1 \cdot f_2$	6	$x_1x_2\bar{x}_3$	-
	4,6	$x_1\bar{x}_3$	i		6,7	x_1x_2	d
	4,5	$x_1\bar{x}_2$	h	$f_1 \cdot f_3$	5,7	x_1x_3	c
f_3	0,2	$\bar{x}_1\bar{x}_3$	g	$f_2f_3 =$	6	$x_1x_2\bar{x}_3$	b
	2,6	$x_2\bar{x}_3$	f	$= f_1f_2f_3$	5	$x_1\bar{x}_2x_3$	a

3. Se notează simbolic implicantii primi ai sistemului, pe coloana a patra a tabelului 2.9, începând cu ultimul implicant prim al ultimei funcții produs. Implicantii primi care apar de mai multe ori se notează o singură dată.

4. Se întocmește un tabel al acoperirilor funcțiilor sistemului, în care se înscriu pe linii toți implicantii primi găsiți la punctul 2, iar pe coloane termenii canonici ai fiecareia dintre funcțiile sistemului, luate în ordine.

Tabelul 2.10 reprezintă tabelul acoperirilor sistemului de funcții (2.87).

Tab.2.10 Tabelul acoperirilor sistemului de funcții din exemplu.

Notație	Indici	Funcții	Termeni canonici											
			Funcția f_1				Funcția f_2				Funcția f_3			
1	5	6	7	1	4	5	6	0	2	5	6	7		
a	5	f_1, f_2, f_3	*				*			*			*	
b	6	f_1, f_2, f_3		*				*				*		*
c	5,7	f_1, f_3	*			*					*			*
d	6,7	f_1, f_3		*	*	*					*		*	*
e	1,5	f_1, f_2	*	*				*						
f	2,6	f_3								*			*	
g	0,2	f_3						*			*			
h	4,5	f_2					*		*					
i	4,6	f_2						*						

5. Se completează tabelul acoperirilor marcând, de exemplu cu asterisc, în dreptul termenilor canonici ai uneia dintre funcții, pe cei inclusi în implicantul prim de pe o anumită linie. Trebuie menționat că acest lucru este posibil numai dacă implicantul prim respectiv este implicant prim al funcției considerate sau al unei funcții produs al acesteia. Pentru a respecta această condiție s-a prevăzut în tabelul 2.10 o coloană în care se specifică funcțiile în acoperirile cărora poate intra fiecare implicant prim.

6. Pe baza tabelului 2.10 se determină acoperirile fiecărei funcții, conform celor arătate la punctul 2.3.1.2. Pentru exemplul considerat, acoperirea funcției f_1 , notată cu $A(f_1)$, este:

$$A(f_1) = ed + ecb = A_1 + A_2. \quad (2.89)$$

De asemenea:

$$A(f_2) = ei + ehb = B_1 + B_2, \quad (2.90)$$

iar

$$A(f_3) = gcb + gcd + gcf + gad = C_1 + C_2 + C_3 + C_4. \quad (2.91)$$

Prin urmare, pentru funcția f_1 s-au găsit două acoperiri, pentru f_2 de asemenea două acoperiri, iar pentru f_3 patru acoperiri. Pentru a se forma o acoperire a sistemului se ia câte una dintre acoperirile fiecărei funcții. Pentru a se găsi acoperirea minimală a sistemului se continuă cu punctul următor.

7. Se scriu toate acoperirile posibile ale sistemului de funcții și se alege dintre acestea acoperirea cu număr minim de elemente. Numărul acoperirilor posibile este dat de produsul numerelor acoperirilor fiecăreia dintre funcții. Pentru exemplul considerat există $2 \times 2 \times 4 = 16$ acoperiri (enumerate în tabelul 2.11). Numărul elementelor unei acoperiri a sistemului este egal cu numărul implicanților primi distincți care intră în

acoperirile tuturor funcțiilor din sistem.

Din tabelul 2.11 rezultă că sistemul de funcții considerat are patru acoperiri minime cu câte cinci elemente, dintre care, din punctul de vedere al costului C_R , cea mai avantajoasă este acoperirea $A_1B_1C_2$.

Tab.2.11 Lista tuturor acoperirilor posibile pentru sistemul de funcții din exemplu.

Nr. crt.	Acoperirea	Elementele acoperirii	Nr. elem.	Costul C_R
1	$A_1B_1C_1$	<i>edigcb</i>	6	13
2	$A_1B_1C_2$	<i>edigc</i>	5	10
3	$A_1B_1C_3$	<i>edigcf</i>	6	12
4	$A_1B_1C_4$	<i>ediga</i>	5	11
5	$A_1B_2C_1$	<i>edhbgc</i>	6	13
6	$A_1B_2C_2$	<i>edhbgc</i>	6	13
7	$A_1B_2C_3$	<i>edhbpcf</i>	7	15
8	$A_1B_2C_4$	<i>edhbga</i>	6	14
9	$A_2B_1C_1$	<i>ecbig</i>	5	11
10	$A_2B_1C_2$	<i>ecbigd</i>	6	13
11	$A_2B_1C_3$	<i>ecbigf</i>	6	13
12	$A_2B_1C_4$	<i>ecbigad</i>	7	16
13	$A_2B_2C_1$	<i>ecbhg</i>	5	11
14	$A_2B_2C_2$	<i>ecbhgd</i>	6	13
15	$A_2B_2C_3$	<i>ecbhgf</i>	6	13
16	$A_2B_2C_4$	<i>ecbhgad</i>	7	16

8. Pe baza acoperirii cu cost minim găsită la punctul 7, se scriu expresiile minime ale sistemului de funcții.

Pentru exemplul tratat, corespunzător acoperirii $A_1B_1C_2$, se obțin expresiile:

$$\begin{aligned}
 f_1^{\text{FMD}}(x_1, x_2, x_3) &= \bar{x}_2 x_3 + x_1 x_2, \\
 f_2^{\text{FMD}}(x_1, x_2, x_3) &= \bar{x}_2 x_3 + x_1 \bar{x}_3, \\
 f_3^{\text{FMD}}(x_1, x_2, x_3) &= \bar{x}_1 \bar{x}_3 + x_1 x_3 + x_1 x_2.
 \end{aligned} \tag{2.92}$$

Procedura de minimizare a sistemului de funcții incomplet definite diferă de metoda prezentată pentru sistemele de funcții complet definite, în ceea ce privește modalitatea de stabilire a setului de implicantă primă pentru funcțiile sistemului. În acest caz, datorită valorilor indiferente ale funcțiilor, apar implicantă primă care pot, sau nu, să implice o anumită funcție, după cum se ia în considerare valoarea 0 sau 1 a acesteia, pentru termenii/combinațiile indiferenți/indiferente. În [23] se prezintă o metodă de minimizare a

acestor sisteme de funcții, în care, pentru determinarea implicațiilor primi ai sistemului de funcții, s-a folosit metoda consensurilor (v. §2.3.1.4) [20].

2.3.2 Implementarea funcțiilor booleene

Circuitele logice constituie modelul fizic al funcțiilor logice bivalente. Orice element fizic, organ sau sistem care în funcționarea sa poate avea două stări stabile, distincte sub acțiunea unei comenzi exterioare este capabil, în principiu, să materializeze variabile sau funcții booleene. În funcționarea lor, elementele capabile de o funcționare binară pot implica intervenția unor fenomene de naturi diferite: mecanice, electrice, magnetice, electronice, pneumatice, hidraulice, etc.

Elementele fizice care stau la baza realizării circuitelor logice se pot împărți în două categorii:

- elemente dinamice sau cu piese în mișcare;
- elemente statice sau fără piese în mișcare.

Prima categorie este definită în special de contactele releelor electromagnetice, dar se mai pot aminti și alte elemente cum ar fi elementele pneumatice respectiv hidraulice cu piese în mișcare. Dacă stării unui contact (închis sau deschis) sau obturării sau nu a unui orificiu prin care circulă un fluid li se asociază valorile binare 0 sau 1, funcționarea lor permite materializarea variabilelor și funcțiilor booleene. Deoarece prin funcționarea elementelor dinamice se conectează sau se deconectează un circuit, electric sau fluidic, se spune că aceste elemente realizează o *logică de conexiuni*.

Elementele din a doua categorie, datorită avantajelor pe care le prezintă în raport cu primele au căpătat o utilizare foarte mare în domeniul circuitelor logice. Din această clasă fac parte: dispozitivele electronice de comutare statică, elementele pneumatice respectiv hidraulice fără piese în mișcare, elementele optoelectronice, etc. Elementelor din această categorie, care în funcționare se bazează pe fenomene electrice, li se asociază cifrele binare 0 și 1 fie nivelelor de tensiune, fie semnalelor sub formă de impulsuri. În cazul nivelelor de tensiune se atribuie convențional valoarea 0 unui nivel de tensiune în jurul valorii de zero volți iar valoarea 1 altui nivel în valoare absolută distinct față de primul. În cazul semnalelor sub formă de impulsuri, absenței impulsului i se atribuie cifra 0, iar prezenței acestuia cifra 1. Elementelor statice care au ca agent de lucru un fluid li se asociază valorile 0 și 1 nivelelor de presiune coborâtă sau nulă și respectiv unor nivele de presiune distincte de primele. Se spune că elementele statice cu acțiune discretă funcționează cu o *logică de nivele*.

2.3.2.1 Implementarea funcțiilor booleene cu relee electromagnetice

Circuitele logice realizate cu relee electromagnetice au fost primele tipuri și mult timp au fost singurele. Referitor la aceste circuite s-a dezvoltat și o teorie matematică puternică la care cercetătorii români și-au adus o importantă contribuție³.

Deși prezintă multe dezavantaje în raport cu elementele statice (piese în mișcare, durată redusă de funcționare, viteză mică de comutare, gabarite mari, întreținere periodică) aceste elemente se utilizează încă suficient de mult datorită unei serii de avantaje

³ Moisil, Gr.C., *Teoria algebrică a schemelor cu contacte și relee*, Ed. Tehnică, București, 1965.

Moisil, Gr.C., *Funcționarea reală a schemelor cu contacte și relee*, Ed. Academiei, București, 1965.

Livovschi, L., *Circuite cu contacte și relee*, Ed. Academiei, București, 1968.

(număr mare de contacte acționate simultan, robustețe la suprasarcini), îndeosebi în realizarea comenziilor de complexitate mică și medie.

Releul electromagnetic materializează o variabilă de stare sau de ieșire a unui automat. Acționarea sa se face prin punerea sub tensiune a înfășurării de comandă prin intermediu unui dipol care realizează funcția logică de comandă a releului și în care intervin elementele de comandă corespunzătoare variabilelor de intrare: butoane, limitatoare, etc. și variabile de stare sau ieșire materializate prin contacte ale releeelor respective.

Cea mai utilizată convenție de atașare a valorilor binare 0 și 1 celor două stări distincte ale unor contacte de releu este următoarea: când înfășurarea releului este alimentată contactelor închise li se atribuie valoarea 1 iar celor deschise valoarea 0; deci în stare de repaus contactelor normal închise (n.i.) valoarea 0 iar contactelor normal deschise (n.d.) valoarea 1.

În cazul în care implementarea se face cu elemente cu piese în mișcare (de tip releu electromagnetic, releu hidraulic, releu pneumatic), se transpun direct prin legături serie și paralel funcțiile řI, respectiv SAU, negarea realizându-se utilizând contacte (căi) normal închise. În figura 2.17 se prezintă materializarea sistemului funcțional complet de bază NU, řI, SAU. În mod curent înfășurării releului i se atașează o literă mare iar contactelor sale minusculele corespunzătoare, adăugându-se eventual o cifră care semnifică ordinea contactelor aceluiași releu.

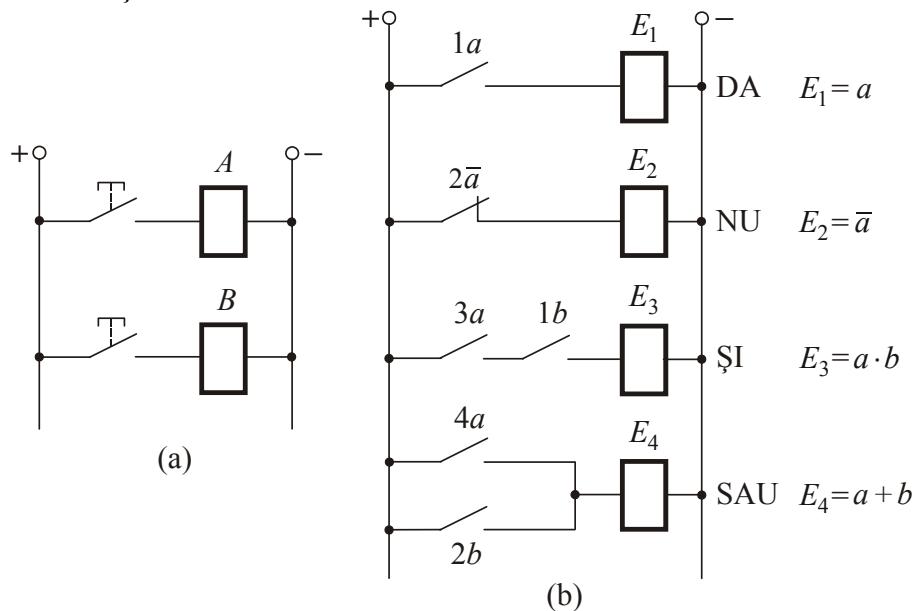


Fig.2.17 Materializarea funcțiilor logice elementare cu relee electromagnetice⁴.

Evident, folosind circuitele prezentate se poate materializa orice funcție logică, deci orice circuit logic în care nu intervine timpul (combinatorial). Folosirea releelor temporizate sau polarizate permite introducerea timpului în prelucrarea logică, obținând astfel posibilitatea realizării cu contacte și a circuitelor secvențiale.

Implementarea funcțiilor booleene cu reele electromagnetice pornește de la una dintre cele două forme minime (FMD respectiv FMC) ale acestora.

⁴ STAS 11381/21-89 – SEMNE CONVENTIONALE PENTRU CONTACTE ȘI APARATE AUXILIARE DE COMANDĂ. SCHEME ELECTRICE.

STAS 11381/23-89 – SEMNE CONVENTIONALE PENTRU CONTACTE ȘI APARATE AUXILIARE DE COMANDĂ. RELEEE ELECTRICE.

Exemplu. Să se implementeze cu relee următorul sistem de funcții booleene:

$$\begin{aligned} f_1^{\text{FMD}}(x_1, x_2, x_3, x_4) &= \\ &= x_1 \bar{x}_2 + x_3 x_4, \quad (2.93) \\ f_2^{\text{FMC}}(x_1, x_2, x_3, x_4) &= \\ &= x_1(x_2 + \bar{x}_4). \end{aligned}$$

Soluția este prezentată în figura 2.18.

Simplitatea deosebită în realizarea circuitelor logice cu contacte este însă marcată de un mare neajuns și anume că nu sunt *unidirectionale*, fapt ce poate conduce la realizarea unor legături false, fiind necesare precauții pentru evitarea acestora.

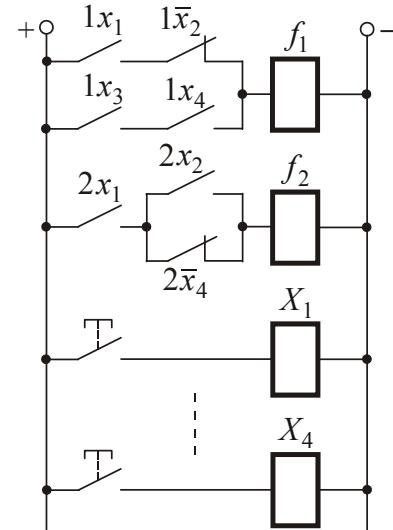


Fig.2.18 Realizarea cu relee a sistemului de funcții din exemplu.

2.3.2.2 Implementarea funcțiilor booleene cu circuite integrate

Pozibilitatea dispozitivelor semiconductoare de a funcționa în regim de comutație, cu două stări distincte și stabile, le conferă acestora proprietatea de element cu acțiune discretă apt de a materializa funcții logice. Circuitele logice electronice se bazează tocmai pe funcționarea în regim de comutație a dispozitivelor semiconductoare.

Pentru realizarea cu circuite logice electronice a funcțiilor booleene este necesară, așa cum s-a mai menționat, atribuirea valorilor unei mărimi fizice electrice (tensiune sau curent) mulțimii de două elemente $\{0,1\}$, care reprezintă valorile variabilelor booleene. Cele două valori 0 și 1 sunt puse în corespondență cu două domenii disjuncte ale mărimii fizice alese (v. fig. 2.19).

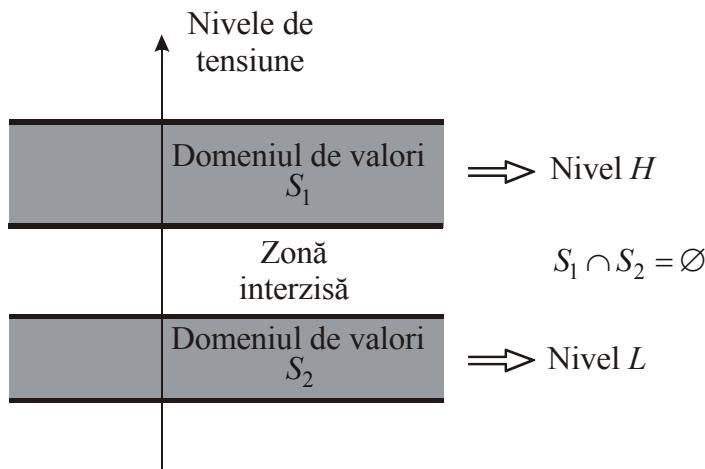


Fig.2.19 Reprezentarea variabilelor booleene prin tensiuni.

Stabilirea unor valori precise pentru cele două nivele logice nu este convenabilă întrucât circuitul care trebuie să realizeze acest lucru devine mai complicat. Este absolut necesară condiția disjuncției celor două domenii de valori ($S_1 \cap S_2 = \emptyset$) întrucât

elementele comune ar crea confuzii de interpretare în domeniul valorilor funcțiilor booleene.

Reprezentarea valorilor funcțiilor booleene prin nivele de tensiune este mai răspândită și în cele ce urmează ne vom referi la această reprezentare. Nivelele de tensiune din cele două domenii de valori S_1 și S_2 respectă relația:

$$\text{pentru } \forall v_1 \in S_1 \text{ și } \forall v_2 \in S_2, \text{ există } v_1 > v_2. \quad (2.94)$$

Datorită acestui fapt tensiunile din intervalul S_1 se mai numesc *nivele H (High)* iar cele din intervalul S_2 se numesc *nivele L (Low)*. Deoarece între mulțimea valorilor funcțiilor booleene $\{0,1\}$ și mulțimea domeniilor de tensiune $\{S_1, S_2\}$ se pot stabili corespondențele din figura 2.20, este necesară stabilirea unei convenții de asociere pentru a defini funcția logică realizată de un anumit circuit.

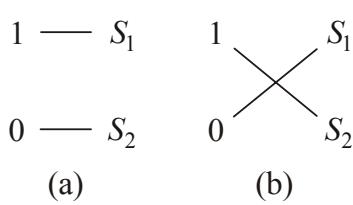


Fig.2.20 Nivele logice și tensiuni: (a) logică pozitivă, (b) logică negativă.

Situată în care valorile maxime de tensiune corespund la 1 logic iar cele minime la 0 logic definește ceea ce se numește *logică pozitivă* (v. fig. 2.20,a), iar situația în care valorile maxime de tensiune corespund la 0 logic iar cele minime la 1 logic definește *logica negativă* (v. fig. 2.20,b).

În practică se întâlnesc ambele tipuri de corespondențe. În plus, este posibilă folosirea ambelor reprezentări logice, pozitivă și negativă, într-un singur sistem logic (folosind în continuare numai două tensiuni). Folosirea ambelor reprezentări în același sistem este cunoscută ca *reprezentarea logică mixtă*. În acest caz este necesară o notație specială pentru a numi semnalele logice. Trebuie observat să schimbarea convenției este echivalentă cu o negare a variabilei booleene.

Stabilirea nivelelor de tensiune corespunzătoare domeniilor S_1 și S_2 depinde de modul de realizare al circuitului, de felul tranzistoarelor folosite, de tehnologia utilizată în cazul circuitelor integrate, de tensiunile de alimentare întrebuințate. De exemplu, pentru familia de circuite logice integrate TTL, 1 logic este reprezentat de o tensiune nominală de 3,4V, deși ea poate lua valori între 2,4V și 5V. Adesea se asociază +5V lui 1 logic. Tensiunea de alimentare a circuitelor TTL este tot de +5V. Un 0 logic este reprezentat de 0,2V nominal, dar poate fi generat cu valori între 0V și 0,4V. În general, se asociază 0V lui 0 logic. Pot fi mici variații între nivelul de tensiune nominală declarată la diverse circuite pentru 0 logic și 1 logic. Tensiuni coborând până la 2V vor fi recunoscute ca 1 logic, iar tensiuni urcând până la 0,8V vor fi recunoscute ca 0 logic, permitând în sistem un "zgomot" electric străin⁵. Suprapunerea parțială dintre gamele de tensiune recunoscute și gamele de tensiune generate este prezentată în figura 2.21.

Evoluția circuitelor logice a dus în timp la dezvoltarea unor familii de circuite în care se încearcă realizarea unui compromis optim între diversele performanțe și cost.

După modul de realizare, circuitele logice electronice se pot împărți în două mari categorii: circuite logice realizate din elemente discrete și circuite logice integrate. Dezvoltarea explozivă a tehnologiilor de integrare a dispozitivelor semiconductoare a

⁵ Marginea (marja) de zgomot este nivelul de tensiune prezent ca și zgomot electric care poate fi tolerat în sistem. În TTL, marjele de zgomot, atât la nivel logic 0 cât și la nivel logic 1, sunt de +0,4V.

făcut ca circuitele din cea de a doua categorie să se impună net, datorită avantajelor lor incontestabile, motiv pentru care în lucrare nu vor fi tratate decât problemele legate de implementarea funcțiilor booleene cu astfel de circuite.

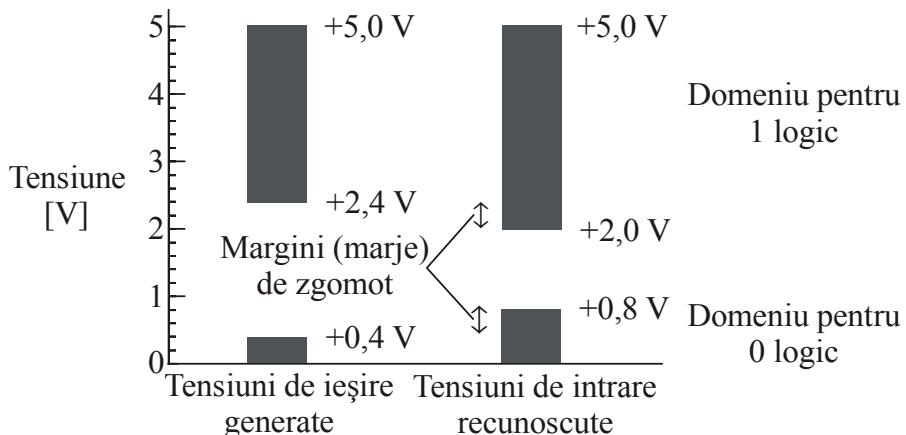


Fig.2.21 Tensiuni TTL.

Circuitele logice integrate sunt realizate, în general, în tehnologia *TTL* (Transistor Transistor Logic), respectiv realizate în tehnologia *CMOS* (Complementary Metal Oxide Semiconductor).

Metodele matematice prezentate anterior sunt necesare dar insuficiente pentru proiectarea cu circuite integrate. Unul din scopurile importante ale acestia este obținerea unor scheme cu număr minim de circuite integrate și cu un cost cât mai scăzut.

Abordând din acest punct de vedere problema implementării se impune cu necesitate parcurgerea a două etape:

- cunoașterea tipurilor de circuite integrate combinaționale;
- stăpânirea metodelor specifice de proiectare cu aceste circuite și a criteriilor de alegere a celor care conduc la o schema minimă.

Există mai multe metode de implementare a CLC-urilor, diferențiate după nivelul de complexitate al circuitelor integrate utilizate.

2.3.2.2.1 Implementarea cu circuite integrate SSI

Circuitele integrate pe scară mică, *SSI* (Small Scale Integration), conțin până la 100 de tranzistoare/cip (sau, până la 12 porti/capsulă). Din această categorie de circuite integrate fac parte diferite tipuri de porti logice: ȘI (AND), SAU (OR), INVERSOR (INVERTER), ȘI-NU (NAND), SAU-NU (NOR), SAU-EXCLUSIV (XOR), ȘI-SAU-NU (AOI), etc.

Prima familie de porti integrate care a reprezentat un succes tehnologic, a fost cea numită *TTL*, apărută în anii '60 și folosită încă pentru sistemele digitale de complexitate redusă și medie și pentru interfațarea cu circuitele cu un grad mai mare de integrare. În multe cazuri, tehnologia inițială *TTL* a fost înlocuită de tehnologia *CMOS*, dar funcționalitatea circuitelor a rămas aceeași.

Portile logice integrate pot exista într-un circuit integrat fie ca elemente de-sine-stătătoare (prezentând deci intrări și ieșiri proprii), fie în calitate de elemente componente într-o structură de circuit logic mai complex.

Pentru implementarea funcțiilor logice cu circuite integrate *SSI* este necesară minimizarea matematică a acestora.

Operația de convertire a expresiilor funcțiilor logice exprimate sub formă minimă în scheme logice echivalente cu porți logice, reprezintă o problemă de optimizare, a cărei rezolvare depinde de experiența proiectantului în contextul unor restricții impuse privind: tipurile de porți disponibile; natura semnalelor disponibile; tipul de poartă care asigură utilizarea numărului minim de capsule; numărul intrărilor fiecărei porți; numărul de porți într-o capsulă; numărul de nivele logice; întârzierile impuse în propagare și prețul circuitelor utilizate [2,15,22,23,26,27,29,30,33].

Exemplu. Să se implementeze un circuit de „anticoincidență“ cu trei intrări (circuit care dă 1 logic la ieșire când cele trei variabile de intrare nu sunt identice, toate 0 sau toate 1), în următoarele variante:

- cu porți řI (AND), SAU(OR) și INVERSOARE;
- cu porți řI-NU (NAND);
- cu porți SAU-NU (NOR);
- cu porți SAU-EXCLUSIV (XOR) și řI-NU (NAND);
- cu porți řI-SAU-NU (AOI) și řI-NU (NAND);
- cu porți řI-SAU-NU (AOI) și INVERSOARE.

Care este relația între minimizarea matematică a funcției și minimizarea implementării?

Prin implementarea unei funcții sau a unui sistem de funcții booleene se înțelege realizarea acestora cu circuite integrate, adică, proiectarea și desenarea schemei cu simbolurile logice asociate circuitelor și calcularea numărului de circuite integrate necesare.

În tabelul 2.12 este reprezentat tabelul de adevăr, iar în figura 2.22 diagrama Karnaugh pentru funcția logică tratată în exemplu.

Tab.2.12 Tabelul de adevăr pentru funcția din exemplu.

	x_1	x_2	x_3	f		x_1	x_2	x_3	f
0	0	0	0	0	4	1	0	0	1
1	0	0	1	1	5	1	0	1	1
2	0	1	0	1	6	1	1	0	1
3	0	1	1	1	7	1	1	1	0

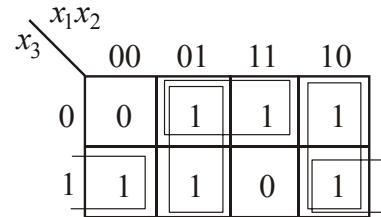


Fig.2.22 Diagrama Karnaugh pentru funcția din exemplu.

Forma minimă disjunctivă a funcției este următoarea:

$$f^{\text{FMD}}(x_1, x_2, x_3) = \bar{x}_1 x_2 + x_1 \bar{x}_2 + x_2 \bar{x}_3 + \bar{x}_2 x_3. \quad (2.95)$$

a) Implementarea cu porți řI, SAU și INVERSOARE este prezentată în figura 2.23.a. Se desenează circuitele prin simboluri logice și în dreptul fiecărui se notează numărul asociat din cadrul seriei de circuite integrate digitale utilizate (în lucrare se vor folosi circuite integrate digitale din seria 74xxx, produse ale firmei Texas Instruments, [34], v. Anexa D; excepție fac o parte dintre circuitele integrate utilizate în §2.3.2.2.3). Fracția dinaintea numărului indică cât reprezintă poarta respectivă din cadrul circuitului integrat. Se utilizează 1×7404, 1×7408 și 1×7432 deci, în total 3 circuite integrate din care rămân însă disponibile 3 INVERSOARE și 1 poartă SAU. Se observă că acest tip de implementare este dezavantajos: se utilizează trei tipuri de circuite integrate și, în plus nu se poate aplica eficient la funcții logice cu mai mulți termeni, deoarece nu există circuite integrate care să conțină porți SAU cu mai mult de 2 intrări.

b) Implementarea cu porți ȘI-NU prezintă față de prima variantă de implementare avantajul că, folosind doar două tipuri de circuite integrate, atinge un grad destul de bun de utilizare a acestora.

Pentru ca implementarea cu porți ȘI-NU să devină evidentă, se transformă expresia funcției f cu ajutorul teoremelor lui De Morgan:

$$f^{\text{FMD}}(x_1, x_2, x_3) = \overline{x_1\bar{x}_2 + \bar{x}_1x_2 + x_2\bar{x}_3 + \bar{x}_2x_3} = \overline{x_1\bar{x}_2} \cdot \overline{\bar{x}_1x_2} \cdot \overline{x_2\bar{x}_3} \cdot \overline{\bar{x}_2x_3}. \quad (2.96)$$

Schema este prezentată în figura 2.23,b. Se utilizează 2×7400 și 1×7420 din care rămân disponibile 1 poartă ȘI-NU cu 2 intrări și 1 poartă ȘI-NU cu 4 intrări.

La porțile logice ȘI-NU o intrare nefolosită trebuie să fie legată, fie împreună cu o altă intrare folosită, fie la un potențial corespunzător constantei logice 1.

c) Implementarea cu porți logice SAU-NU asigură, de asemenea, datorită uniformității, un grad mai bun de utilizare al circuitelor integrate în raport cu prima variantă de implementare. În schimb, față de varianta a două de implementare apare un nivel logic suplimentar. Și în acest caz este necesară transformarea expresiei funcției f , cu ajutorul teoremelor lui De Morgan, pentru înlocuirea funcției ȘI:

$$\begin{aligned} f^{\text{FMD}}(x_1, x_2, x_3) &= \overline{x_1\bar{x}_2 \cdot \bar{x}_1x_2 \cdot x_2\bar{x}_3 \cdot \bar{x}_2x_3} = \overline{(\bar{x}_1 + x_2)(x_1 + \bar{x}_2)(\bar{x}_2 + x_3)(x_2 + \bar{x}_3)} = \\ &= \overline{(\bar{x}_1 + x_2)} + \overline{(x_1 + \bar{x}_2)} + \overline{(\bar{x}_2 + x_3)} + \overline{(x_2 + \bar{x}_3)}. \end{aligned} \quad (2.97)$$

Implementarea este aceea din figura 2.23,c; se utilizează trei circuite integrate, 2×7402 și 1×74260 , rămânând liberă o poartă SAU-NU cu 5 intrări.

La porțile logice SAU-NU o intrare nefolosită trebuie să fie legată, fie împreună cu altă intrare folosită, fie la un potențial corespunzător constantei logice 0.

d) Se observă că expresia funcției f se poate scrie și sub forma:

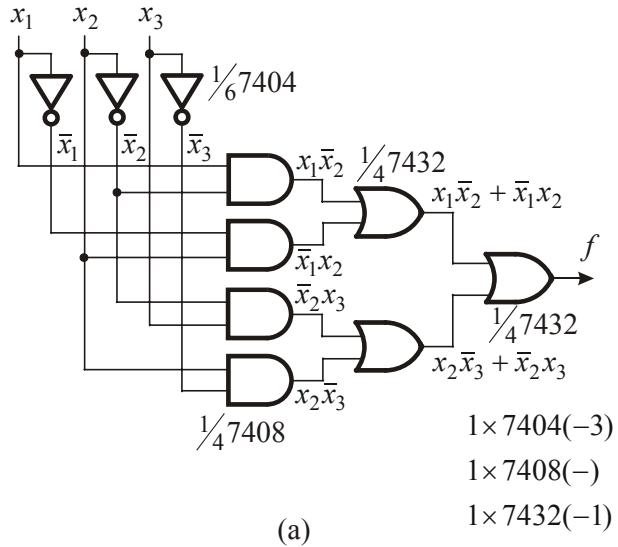
$$\begin{aligned} f^{\text{FMD}}(x_1, x_2, x_3) &= x_1\bar{x}_2 + \bar{x}_1x_2 + x_2\bar{x}_3 + \bar{x}_2x_3 = \overline{(x_1 \oplus x_2) + (x_2 \oplus x_3)} = \\ &= \overline{(x_1 \oplus x_2) \cdot (x_2 \oplus x_3)}. \end{aligned} \quad (2.98)$$

Deci implementarea (v. fig. 2.23,d) se face doar cu două circuite integrate: 1×7486 și 1×7400 , din care rămân neutilizate 2 porți SAU-EXCLUSIV și 1 poartă ȘI-NU cu două intrări. Este cea mai avantajoasă soluție de implementare de până în acest moment, doar că o astfel de implementare nu se poate face decât în cazul în care în expresia funcției de implementat se pune în evidență funcția SAU-EXCLUSIV.

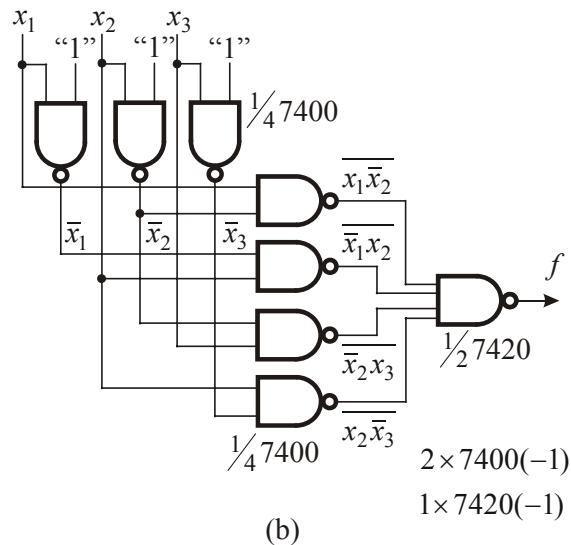
e) Varianta de implementare cu porți ȘI-SAU-NU și ȘI-NU este prezentată în figura 2.23,e. Pentru a se pune în evidență operația ȘI-SAU-NU expresia funcției f poate fi transformată în felul următor:

$$f^{\text{FMD}}(x_1, x_2, x_3) = \overline{x_1\bar{x}_2 + \bar{x}_1x_2 + x_2\bar{x}_3 + \bar{x}_2x_3} = \overline{(x_1\bar{x}_2 + \bar{x}_1x_2)} \cdot \overline{(x_2\bar{x}_3 + \bar{x}_2x_3)}. \quad (2.99)$$

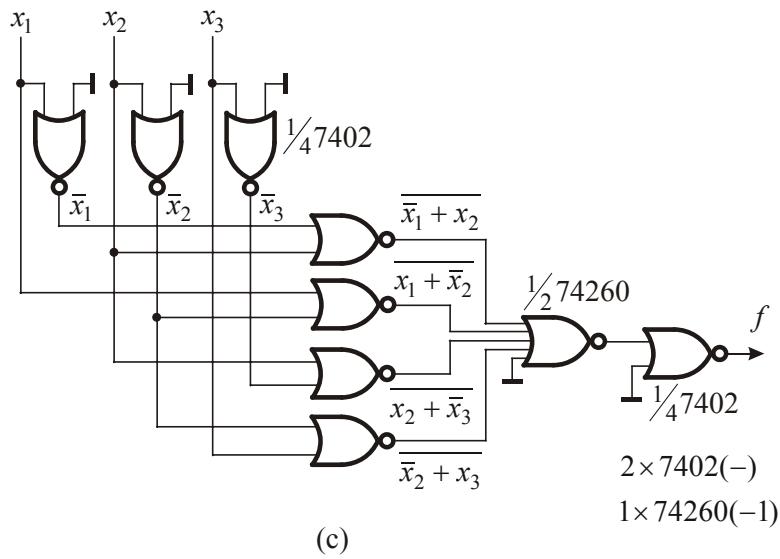
Sunt necesare, pentru implementare, două circuite integrate: 1×7400 și 1×7451 , utilizate în totalitate. Din punct de vedere al numărului de circuite integrate folosite și al gradului de utilizare al acestora este cea mai avantajoasă soluție de implementare pentru funcția f .



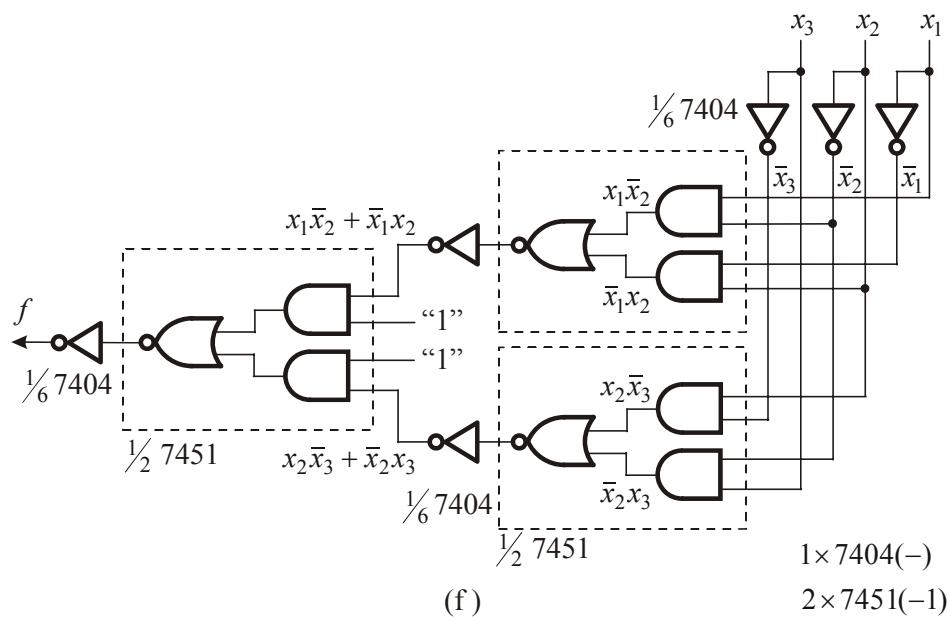
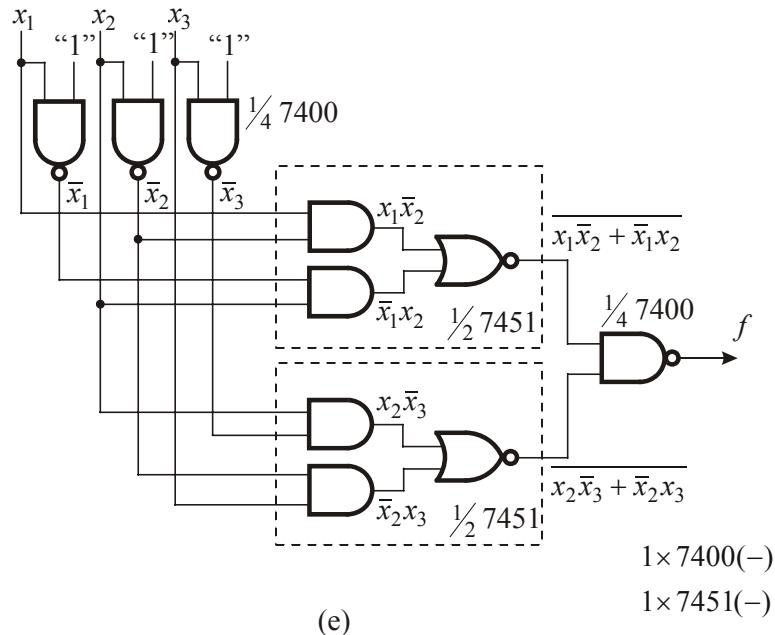
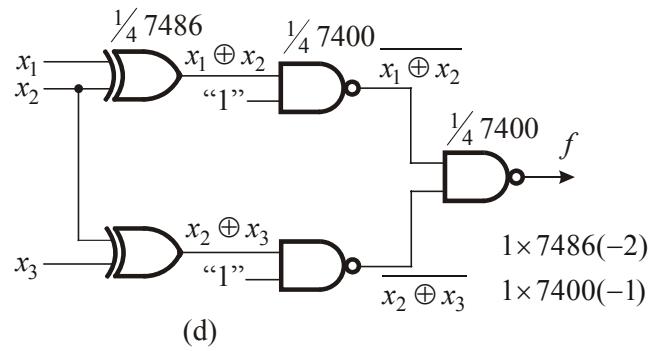
(a)



(b)



(c)


 Fig.2.23 Implementarea funcției f din exemplu cu diferite tipuri de porți logice.

f) Cea de a doua variantă de implementare care utilizează porti řI-SAU-NU, împreună, de această dată, cu INVERSOARE (v. fig. 2.23,f) folosește trei circuite integrate: 2×7451 , 1×7404 , rămânând neutilizată o poartă řI-SAU-NU 2×2 .

Se demonstrează, analizând exemplul de mai sus, că prin realizarea unei funcții booleene, dată sub formă minimă, cu diverse tipuri de porti logice, se obține un număr variabil de circuite integrate, sau, cu alte cuvinte, că deși funcția f este dată sub formă minimizată matematic, nu toate implementările conduc la un număr minim de circuite integrate. Înseamnă că, ulterior minimizării matematice este necesară alegerea aceluia tip de poartă logică care conduce la schema logică cea mai redusă ca dimensiuni. Această etapă de proiectare nu mai este formalizată matematic și pentru parcurgerea ei, cât mai rapid, este necesară formarea unor deprinderi, a unei experiențe. Pentru minimizarea implementării unei funcții logice trebuie să se aibă în vedere următoarele aspecte:

- să se utilizeze circuite integrate cât mai puțin dispersate ca tipuri;
- să se aleagă tipul de circuit integrat cel mai potrivit pentru implementare, în raport cu forma funcției de implementat.

În cazul CLC-urilor definite de un ansamblu (sistem) de funcții logice, implementarea cu porti logice minimă se obține luând în considerare ansamblul funcțiilor și nu implementând fiecare funcție independent și punând în evidență, în expresiile funcțiilor a cât mai multor termeni comuni, chiar dacă în acest fel implementarea nu mai pornește de la formele minimizate matematice.

Exemplu. Să se implementeze cu porti řI-NU circuitul logic combinațional definit de următoarele funcții:

$$\begin{aligned} f_1^{\text{FCD}}(x_1, x_2, x_3, x_4) &= P_0 + P_1 + P_2 + P_3 + P_{10} + P_{11} + P_{14} + P_{15}, \\ f_2^{\text{FCD}}(x_1, x_2, x_3, x_4) &= P_0 + P_1 + P_5, \\ f_3^{\text{FCD}}(x_1, x_2, x_3, x_4) &= P_1 + P_2 + P_3 + P_5, \end{aligned} \quad (2.100)$$

- minimizând fiecare funcție independent;
- minimizând ansamblul funcțiilor.

a) Diagramele Karnaugh pentru funcțiile f_1 , f_2 și f_3 sunt prezentate în figura 2.24.

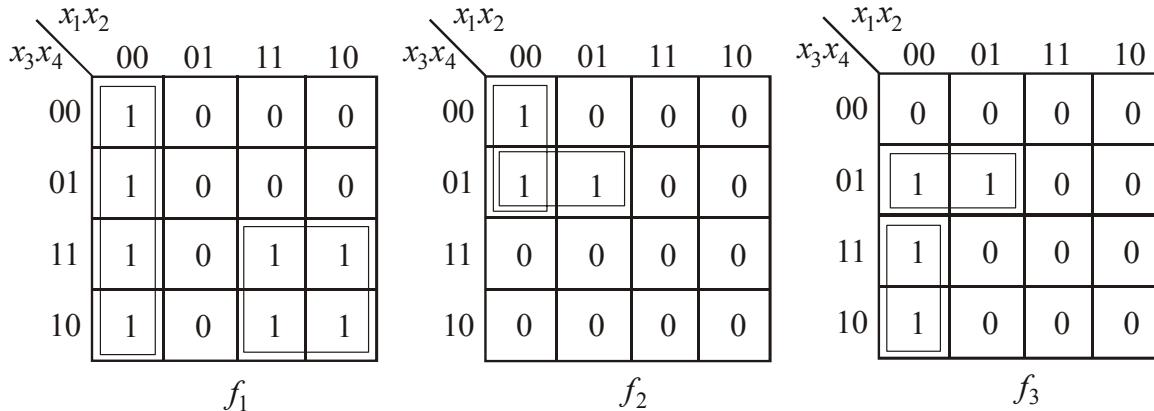


Fig.2.24 Diagramele Karnaugh pentru funcțiile f_1 , f_2 și f_3 .

Din diagrama de minimizare pentru funcția f_1 (v. fig. 2.24) rezultă:

$$f_1^{\text{FMD}}(x_1, x_2, x_3, x_4) = \overline{\overline{x_1} \overline{x_2} + x_1 x_3} = \overline{\overline{x_1} \overline{x_2}} \cdot \overline{x_1 x_3}, \quad (2.101)$$

care se implementează cu 3 porți řI-NU cu 2 intrări și 2 INVERSOARE (se vor folosi ca INVERSOARE porțile řI-NU rămase disponibile în circuitele integrate utilizate).

Forma minimă disjunctivă pentru funcția f_2 este următoarea:

$$f_2^{\text{FMD}}(x_1, x_2, x_3, x_4) = \overline{\overline{x_1} \overline{x_2} \overline{x_3} + \overline{x_1} \overline{x_3} x_4} = \overline{\overline{x_1} \overline{x_2} \overline{x_3}} \cdot \overline{\overline{x_1} \overline{x_3} x_4} \quad (2.102)$$

și sunt necesare, pentru implementare, 2 porți řI-NU cu 3 intrări, 1 poartă řI-NU cu 2 intrări și încă un INVERSOR, pentru \overline{x}_3 (variabilele x_1 și x_2 au fost deja negate pentru obținerea funcției f_1). De asemenea f_3 minimizată, este:

$$f_3^{\text{FMD}}(x_1, x_2, x_3, x_4) = \overline{\overline{x_1} \overline{x_2} x_3 + \overline{x_1} \overline{x_3} x_4} = \overline{\overline{x_1} \overline{x_2} x_3} \cdot \overline{\overline{x_1} \overline{x_3} x_4} \quad (2.103)$$

și se implementează cu 2 porți řI-NU cu 3 intrări și o poartă řI-NU cu 2 intrări.

Deci, în total sunt necesare 4 circuite integrate (v. fig. 2.25), 2×7400 și 2×7410 și rămân disponibile 3 porți řI-NU cu 2 intrări (s-au folosit ca INVERSOARE) și 2 porți řI-NU cu 3 intrări.

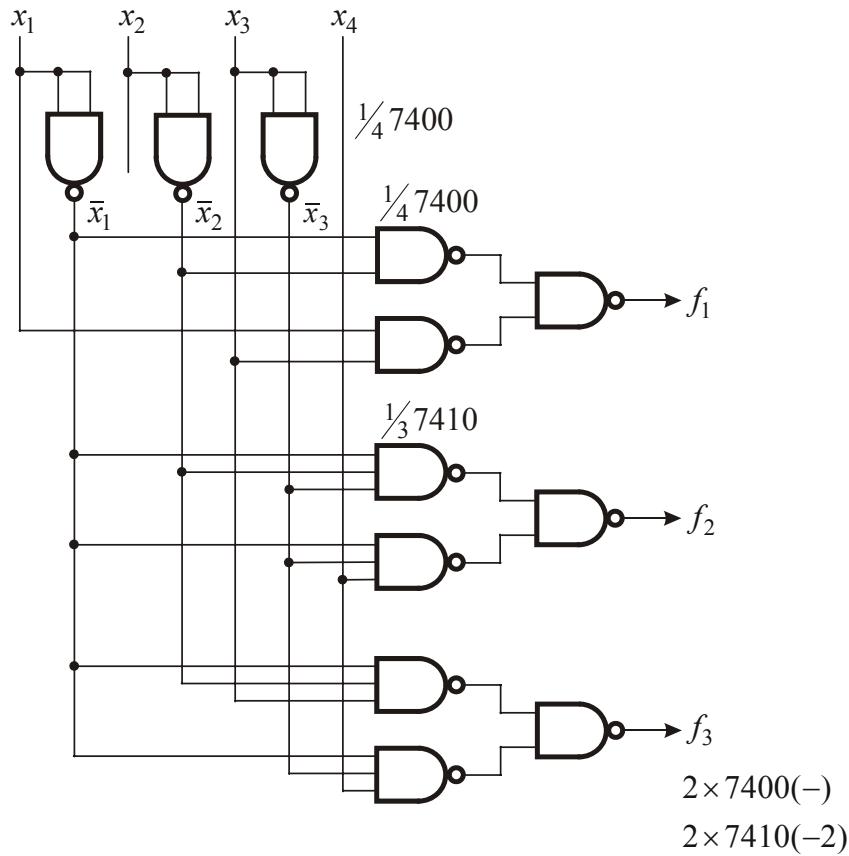


Fig.2.25 Implementarea individuală a funcțiilor ansamblului.

b) Pentru minimizarea ansamblului funcțiilor s-a folosit metoda prezentată mai sus, în §2.3.1.6.

Se calculează mai întâi funcțiile produs:

$$f_1 = \sum(0,1,2,3,10,11,14,15), \quad f_1 \cdot f_2 = \sum(0,1),$$

$$f_2 = \sum(0,1,5), \quad f_1 \cdot f_3 = \sum(1,2,3), \quad f_1 \cdot f_2 \cdot f_3 = \sum(1). \quad (2.104)$$

$$f_3 = \sum(1,2,3,5), \quad f_2 \cdot f_3 = \sum(1,5),$$

Se determină în continuare implicantii primi ai fiecărei dintre funcțiile (2.104), folosindu-se în acest scop diagramele Karnaugh (v. fig. 2.26).

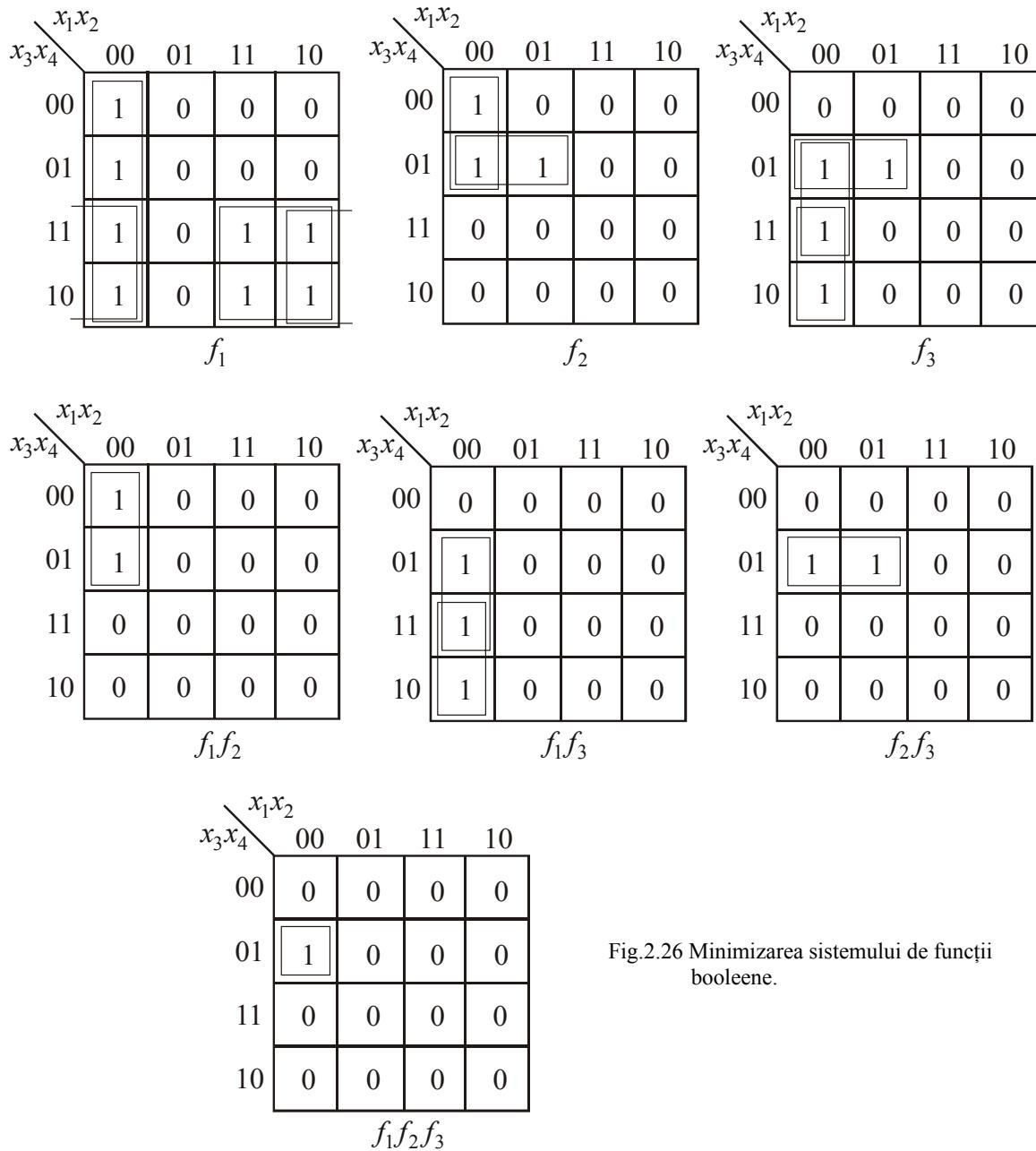


Fig.2.26 Minimizarea sistemului de funcții booleene.

Din figura 2.26 se obțin, pentru funcțiile considerate, implicantii primi din tabelul 2.13. Se întocmește, de asemenea, un tabel al acoperirilor funcțiilor sistemului din exemplul analizat (v. tab. 2.14).

Tab.2.13 Tabelul implicantilor primi pentru sistemul de funcții.

Funcția	Implicantii primi			Funcția	Implicantii primi		
	Indicii	Expresia	Notăția		Indicii	Expresia	Notăția
f_1	0,1,2,3	\bar{x}_1x_2	h	f_3	1,5	$\bar{x}_1\bar{x}_3x_4$	-
	2,3,10,11	\bar{x}_2x_3	g	f_1f_2	0,1	$\bar{x}_1\bar{x}_2\bar{x}_3$	e
	10,11,14,15	x_1x_3	f	f_1f_3	1,3 2,3	$\bar{x}_1\bar{x}_2x_4$ $\bar{x}_1\bar{x}_2x_3$	d c
f_2	0,1	$\bar{x}_1\bar{x}_2\bar{x}_3$	-	f_2f_3	1,5	$\bar{x}_1\bar{x}_3x_4$	b
	1,5	$\bar{x}_1\bar{x}_3x_4$	-	$f_1f_2f_3$	1	$\bar{x}_1\bar{x}_2\bar{x}_3x_4$	a
f_3	1,3 2,3	$\bar{x}_1\bar{x}_2x_4$ $\bar{x}_1\bar{x}_2x_3$	-				

Tab.2.14 Tabelul acoperirilor sistemului de funcții.

Implicantii primi			Termeni canonici													
Notăție	Indici	Funcții	f ₁						f ₂			f ₃				
			0	1	2	3	10	11	14	15	0	1	5	1	2	3
a	1	f_1, f_2, f_3		*							*	*	*	*		
b	1,5	f_2, f_3									*	*	*			*
c	2,3	f_1, f_3			*	*								*	*	*
d	1,3	f_1, f_3		*		*							*		*	*
e	0,1	f_1, f_2	*	*							*	*				
f	10,11,14,15	f_1					*	*	*	*						
g	2,3,10,11	f_1			*	*	*	*	*							
h	0,1,2,3	f_1	*	*	*	*										

În acest mod se obțin următoarele forme minime pentru funcțiile sistemului:

$$\begin{aligned} A(f_1) &= f + h = x_1x_3 + \bar{x}_1\bar{x}_2 = x_1x_3 + \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3 = \\ &= \overline{\overline{x}_1x_3} \cdot \overline{\bar{x}_1\bar{x}_2\bar{x}_3} \cdot \overline{\bar{x}_1\bar{x}_2x_3}, \end{aligned} \quad (2.105)$$

$$A(f_2) = b + e = \bar{x}_1\bar{x}_3x_4 + \bar{x}_1\bar{x}_2\bar{x}_3 = \overline{\overline{\bar{x}_1\bar{x}_3x_4}} \cdot \overline{\bar{x}_1\bar{x}_2\bar{x}_3}, \quad (2.106)$$

$$A(f_3) = b + c = \bar{x}_1\bar{x}_3x_4 + \bar{x}_1\bar{x}_2x_3 = \overline{\overline{\bar{x}_1\bar{x}_3x_4}} \cdot \overline{\bar{x}_1\bar{x}_2x_3}. \quad (2.107)$$

Se observă că dacă funcția f_1 nu se minimizează complet, implicantul prim $\bar{x}_1\bar{x}_2$ poate fi considerat ca fiind format din termenii normali $\bar{x}_1\bar{x}_2x_3$ și $\bar{x}_1\bar{x}_2\bar{x}_3$, care se regăsesc și în funcțiile f_2 , respectiv f_3 . Astfel, implementarea ansamblului de funcții, în care termenii

comuni se obțin cu ajutorul unei singure porți logice, se poate face numai cu 3 circuite integrate ($1 \times 7400, 2 \times 7410$), deci mai economic decât în cazul în care f_1 este dată în formă minimă (v. fig. 2.27).

Două sau mai multe porți logice (de exemplu, porți řI-NU) în structură obișnuită nu pot avea ieșirile cuplate în paralel. Pentru a se cupla, totuși, în paralel mai multe porți se utilizează circuite integrate de tipul cu *colector în gol/deschis, OC (Open Collector)* (v. Anexa B). Poarta cu colector deschis este, de asemenea, folosită pentru cuplarea bi (multi) direcțională a unor circuite logice (registre, memorii, etc.).

O alternativă favorabilă la cuplarea mai multor porți logice în paralel pentru transmiterea, de exemplu a informației pe o magistrală comună de date (BUS), o reprezintă subfamilia de circuite integrate digitale cu *trei stări logice, TSL (Three State Logic)* (v. Anexa B). Aceasta este utilă nu numai pentru cuplarea ieșirilor porților logice dar și a altor circuite integrate digitale (de ex., pentru extinderea capacitatei memorii ROM, etc.).

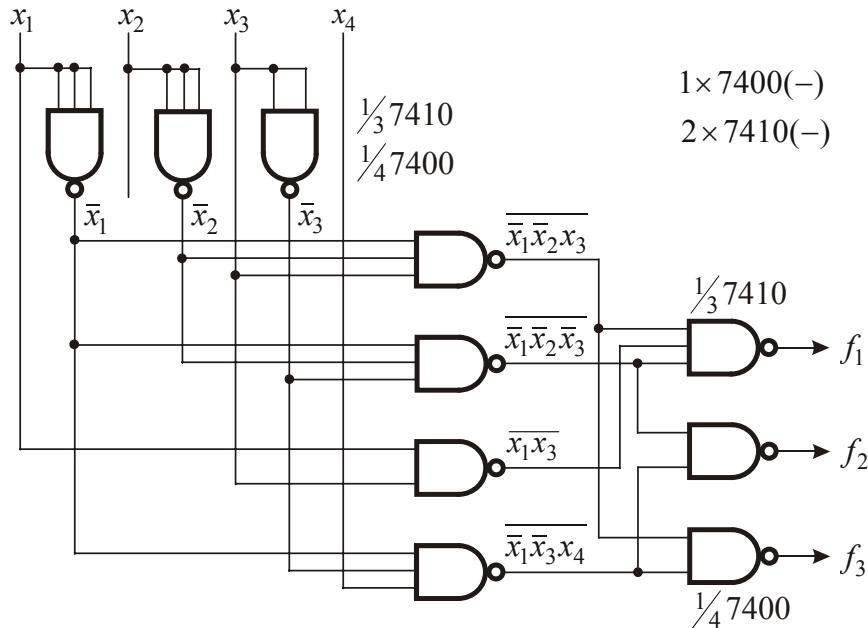


Fig.2.27 Implementarea ansamblului de funcții.

Observație. Intrările nefolosite ale circuitelor integrate digitale NU SE LASĂ ÎN GOL!! Ignorarea acestei reguli poate produce, în exploatarea echipamentului digital, surpreze dintre cele mai neplăcute. Astfel, intrările lăsate în gol adaugă capacitate parazite a căror încărcare produce întârzieri la intrare și prin urmare o creștere a timpilor de propagare și captează semnale parazite.

Există două categorii de intrări:

- intrări care nu pot fi decât la un anumit nivel logic; intrările nefolosite ale porților řI sau řI-NU trebuie legate la 1 logic, în timp ce intrările nefolosite ale porților SAU sau SAU-NU se leagă la 0 logic, etc.;

- intrări care pot fi legate fie la 0, fie la 1 logic, fără ca prin aceasta să fie afectată funcționarea circuitului (de exemplu, intrările porților nefolosite dintr-un cip; intrările de date ale numărătoarelor presetabile, evident, dacă nu sunt folosite; intrările multiple-xoarelor nefolosite dintr-un cip, etc.). În acest caz criteriu este minimizarea puterii absorbite de către un cip de la sursa de alimentare.

Intrările neutilitate mai pot fi, de asemenea, legate la:

– intrările utilizate care îndeplinesc aceeași funcție; este o soluție mai comodă, dar prezintă dezavantajul că se încarcă suplimentar circuitul care comandă intrările respective;

– ieșiri de circuite care au, în permanență, același nivel logic; în acest caz trebuie verificat că nu se depășește capacitatea de comandă a respectivelor ieșiri.

O altă problemă care trebuie avută în vedere în cazul implementării cu porți logice este aceea că numărul intrărilor este, în general, limitat. De exemplu, există porți řI-NU cu 1 (inversorul), 2, 3, 4 și 8 intrări. Numărul intrărilor pentru portile řI, SAU-NU, etc. este și mai mic. Apare prin urmare, necesitatea practică a măririi numărului de intrări. Există mai multe posibilități pentru realizarea acestui lucru:

– se cuplează ieșirile a două sau mai multe porți cu un număr de intrări la intrarea altelor porți;

– se realizează funcția logică řI-cablat cu ajutorul portilor cu colectorul în gol (open collector, v. Anexa B);

– se folosește un circuit special de expandare care atacă bornele de expandare prevăzute la unele circuite, atacul făcându-se la nivelul SAU.

2.3.2.2.2 Implementarea cu circuite integrate MSI

Implementarea cu circuite integrate pe scară medie, *MSI* (*Medium Scale Integration*), care conțin până la 1000 de tranzistoare/cip (sau, având între 12 și 100 de porți/capsulă), utilizează circuite universale ca: decodificatorul respectiv demultiplexorul și multiplexorul care generează, în interior, toți termenii canonici de 2, 3, respectiv 4 variabile.

Decodificatorul și demultiplexorul (se vor trata împreună pentru că se asemănă din punct de vedere constructiv și funcțional) realizează, așa cum s-a precizat mai sus, doar nivelul logic řI. Aceste circuite transmit termenii canonici pe care îi generează, în exterior, unde sunt „adunați“ în porti logice (nivelul logic SAU) corespunzător cu funcțiile logice ce trebuie implementate. Multiplexorul realizează constructiv ambele nivele logice sau cu alte cuvinte, „adună“, în interior, termenii canonici pe care îi generează. Din acest motiv aceste circuite trebuie prevăzute cu intrări de validare care să permită transmisarea, spre exterior, doar a celor termeni canonici ce intervin în expresia funcției de implementat.

În aceste cazuri, deoarece se pornește direct de la termeni canonici, minimizarea matematică a funcțiilor logice nu mai este necesară. Există însă metode specifice de reducere a schemelor logice, la MSI minimizarea implementării presupunând, pe lângă obținerea unui număr minim de circuite integrate și utilizarea unor circuite integrate cu un grad mai scăzut de complexitate.

• Circuite multiplexoare

Multiplexorul (MUX-ul) este unul dintre cele mai importante circuite MSI. Este denumit uneori și „selector“ deoarece este utilizat și ca un comutator de selectare a unuitor căi [2,15,22,23,26÷30,33].

Multiplexorul/selectorul este un circuit logic combinatorial care are, în cazul general, 2^n intrări de date, $I_0, I_1, \dots, I_{2^n-1}$, n intrări de selecție (adresă), S_0, S_1, \dots, S_{n-1} și o ieșire Z (v. fig. 2.28,a). Starea ieșirii circuitului la un moment dat este aceeași cu starea intrării I_k , unde indicele k este echivalentul zecimal al numărului binar reprezentat de

stările 0 și 1 ale intrărilor de selecție:

$$k = (S_{n-1}S_{n-2} \dots S_1S_0). \quad (2.108)$$

Ieșirea selectează deci intrarea indicată de variabilele de adresă.

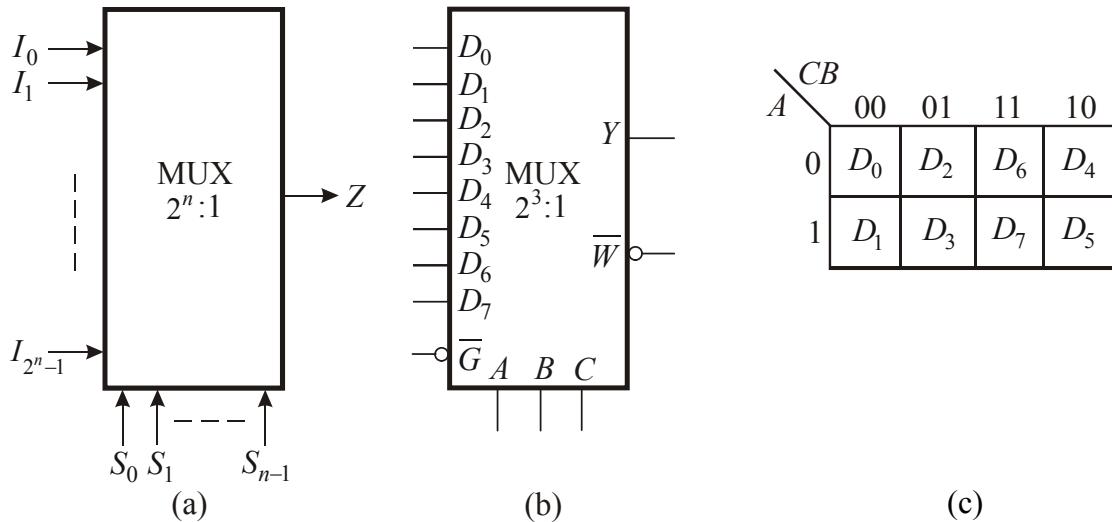


Fig.2.28 Circuit multiplexor: (a) simbolul de reprezentare al unui MUX 2ⁿ :1; (b), (c) simbolul de reprezentare⁶ și diagrama Karnaugh pentru un MUX 2³ :1.

Pentru ca la ieșire să apară întotdeauna numai intrarea selectată trebuie ca selecția să se facă după stabilizarea intrărilor de adresă. Din acest motiv, multiplexoarele sunt prevăzute cu o intrare suplimentară, intrarea de autorizare/validare sau strobare⁷, \bar{G} , care condiționează selecția fiecărei intrări. Ea are de fapt rolul de a comanda inhibarea respectiv dezinhibarea funcționării circuitului. Această intrare suplimentară poate fi folosită și la extinderea numărului de intrări, prin legarea mai multor circuite de multiplexare. Intrarea s-a notat cu \bar{G} deoarece semnalul de autorizare permite selectarea numai atunci când are valoarea logică 0.

Se consideră, în continuare, un multiplexor de 8 căi (MUX 2³ :1) cu opt intrări de date, D_0, D_1, \dots, D_7 , trei intrări de selecție, A, B și C , o intrare de strobare, \bar{G} , activă pe 0 logic și două ieșiri complementare, Y și \bar{W} (v. fig. 2.28,b). Comportarea sa este descrisă în tabelul 2.15. Pe baza tabelului 2.15 se poate întocmi diagrama Karnaugh din figura 2.28,c, în care în locul valorilor logice 0 și 1 s-au înscris variabilele corespunzătoare intrărilor de date care pot avea fie valoare 0, fie valoarea 1. În acest caz, la scrierea expresiei funcției, termenii corespunzători unui anumit compartiment din diagramă se scriu adăugând variabilelor care definesc compartimentul respectiv și variabila înscrisă în interiorul său. Valoarea acestei variabile condiționează valoarea funcției. Având în vedere condițiile introduse în diagrama Karnaugh din figura 2.28,c, expresia ieșirii Y , este:

$$Y = G(\bar{C}\bar{B}\bar{A}D_0 + \bar{C}\bar{B}AD_1 + \bar{C}B\bar{A}D_2 + \bar{C}B\bar{A}D_3 + C\bar{B}\bar{A}D_4 + \\ + C\bar{B}AD_5 + C\bar{B}\bar{A}D_6 + CB\bar{A}D_7). \quad (2.109)$$

⁶ Există simboluri grafice standard și pentru dispozitivele logice MSI și LSI (v. §1.2.3.4), dar aceste simboluri nefiind universal acceptate pentru schemele logice, nu se vor folosi în această lucrare.

⁷ De la denumirea în limba engleză – “to strobe” – a selectă.

Tab.2.15 Tabelul de adevăr pentru un MUX $2^3 : 1$ ((a) – forma detaliată, respectiv (b) – forma compactă).

Intrări									Ieșiri				
Selectie			Strob	Date						Y	\bar{W}		
C	B	A	\bar{G}	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7		
*	*	*	1	*	*	*	*	*	*	*	*	0	1
0	0	0	0	0	*	*	*	*	*	*	*	0	1
0	0	0	0	1	*	*	*	*	*	*	*	1	0
0	0	1	0	*	0	*	*	*	*	*	*	0	1
0	0	1	0	*	1	*	*	*	*	*	*	1	0
0	1	0	0	*	*	0	*	*	*	*	*	0	1
0	1	0	0	*	*	1	*	*	*	*	*	1	0
0	1	1	0	*	*	*	0	*	*	*	*	0	1
0	1	1	0	*	*	*	1	*	*	*	*	1	0
1	0	0	0	*	*	*	*	0	*	*	*	0	1
1	0	0	0	*	*	*	*	1	*	*	*	1	0
1	0	1	0	*	*	*	*	*	0	*	*	0	1
1	0	1	0	*	*	*	*	*	1	*	*	1	0
1	1	0	0	*	*	*	*	*	*	0	*	0	1
1	1	0	0	*	*	*	*	*	*	1	*	1	0
1	1	1	0	*	*	*	*	*	*	*	0	0	1
1	1	1	0	*	*	*	*	*	*	*	1	1	0

Intrări				Ieșiri	
Selectie			Strob	Y	\bar{W}
C	B	A	\bar{G}		
*	*	*	1	0	1
0	0	0	0	D_0	\bar{D}_0
0	0	1	0	D_1	\bar{D}_1
0	1	0	0	D_2	\bar{D}_2
0	1	1	0	D_3	\bar{D}_3
1	0	0	0	D_4	\bar{D}_4
1	0	1	0	D_5	\bar{D}_5
1	1	0	0	D_6	\bar{D}_6
1	1	1	0	D_7	\bar{D}_7

Observație:

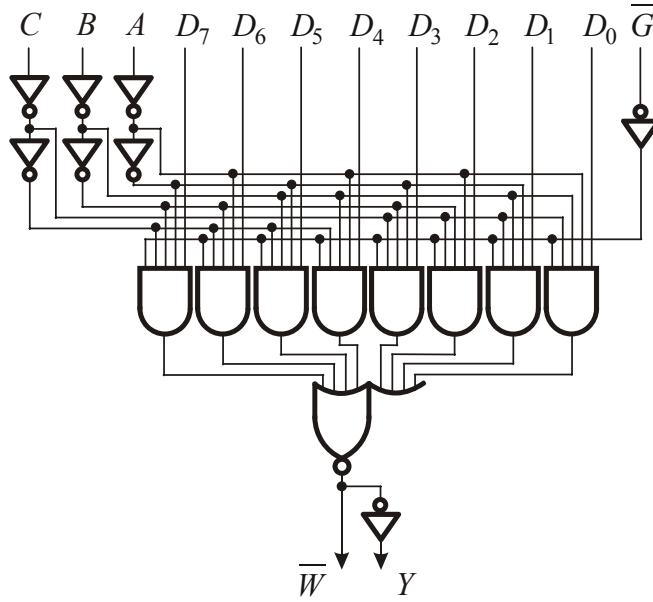
Utilizarea simbolului “*” indică faptul că intrarea este nesemnificativă.

(b)

Posibilitatea utilizării multiplexorului pentru implementarea funcțiilor logice rezultă din ecuația (2.109), care definește funcționarea acestuia. Se observă că, în această relație apar toți termenii canonici care se pot forma cu variabilele de selecție, înmulțiti fiecare cu câte o variabilă corespunzătoare unei intrări de date.

Pe baza expresiei (2.109) se poate obține schema logică a multiplexorului, prezentată în figura 2.29.

Principalele tipuri de multiplexoare utilizate, în mod curent, sunt: MUX-urile 16:1, 8:1, 2×(4:1), 4×(2:1).


 Fig.2.29 Schema logică a unui MUX $2^3 : 1$.

• Implementarea funcțiilor booleene cu multiplexoare

Possibilitatea de utilizare a multiplexorului MUX $2^n : 1$ ca CLC de n variabile, se explică prin faptul că, aşa cum s-a mai precizat, în structura sa se obțin toți termenii canonici de n variabile (nivelul logic řI), cât și suma logică completă între acești termeni (nivelul logic SAU). Deci, cu o astfel de structură, se poate implementa orice funcție booleană de n variabile, dată sub FCD [2,15,22,23,26÷30,33].

Spre deosebire de metodele clasice de implementare cu porți logice, la care se face, în prealabil, o minimizare, în cazul implementării cu multiplexoare se pornește deci de la FCD a funcției de implementat.

În continuare se vor prezenta, parcurgându-se o serie de exemple, particularitățile implementării funcțiilor booleene cu multiplexoare, utilizându-se în acest scop următoarele multiplexoare integrate [34]:

- 74150, MUX $2^4 : 1$, cu o intrare de validare/strobare (\bar{G}) și o ieșire (\bar{W}), ambele active pe 0 logic;
- 74151, MUX $2^3 : 1$, cu o intrare de validare/strobare activă pe 0 logic (\bar{G}) și două ieșiri complementare (Y și \bar{W});
- 74153, $2 \times$ MUX $2^2 : 1$, cu câte o intrare de validare/strobare activă pe 0 logic ($1\bar{G}, 2\bar{G}$) și cu câte o ieșire necomplementată ($1Y, 2Y$);
- 74352, $2 \times$ MUX $2^2 : 1$, cu câte o intrare de validare/strobare activă pe 0 logic ($1\bar{G}, 2\bar{G}$) și cu câte o ieșire complementată ($1\bar{Y}, 2\bar{Y}$).

Exemplu. Să se implementeze cu MUX funcția următoare:

$$f^{\text{FMD}}(x_1, x_2, x_3) = \bar{x}_1\bar{x}_3 + \bar{x}_1x_2 + x_1\bar{x}_2x_3. \quad (2.110)$$

Funcția trebuie adusă, mai întâi, la FCD, prin introducerea variabilelor absente:

$$\begin{aligned}
 f^{\text{FCD}}(x_1, x_2, x_3) &= \bar{x}_1(x_2 + \bar{x}_2)\bar{x}_3 + \bar{x}_1x_2(x_3 + \bar{x}_3) + x_1\bar{x}_2x_3 = \\
 &= \bar{x}_1x_2\bar{x}_3 + \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3 + \bar{x}_1x_2\bar{x}_3 \stackrel{\text{termen redundant}}{=} + x_1\bar{x}_2x_3 = \quad (2.111) \\
 &= \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3,
 \end{aligned}$$

respectiv:

$$f^{\text{FCD}}(x_1, x_2, x_3) = P_0 + P_2 + P_3 + P_5. \quad (2.112)$$

Funcția ce urmează a fi implementată având $n = 3$ variabile, se va folosi multiplexorul MUX $2^3 : 1$, de 3 biți sau 8 căi: 74151 (v. fig. 2.30,a). Pentru a se genera termenii canonici, variabilele funcției se aplică pe intrările de selecție în raport cu ponderile acestora (v. ec. (2.109)). În continuare, pentru a se selecta doar acei termeni canonici care intervin în expresia funcției, se utilizează (legându-se la un potențial corespunzător constantei logice 1) doar intrările de date corespunzătoare (în exemplul tratat: D_0, D_2, D_3, D_5), celelalte intrări de date, neutilizate, fiind legate la un potențial corespunzător constantei logice 0:

$$\begin{aligned}
 f^{\text{FCD}}(x_1, x_2, x_3) &= P_0 \cdot D_0^{(=1)} + P_1 \cdot D_1^{(=0)} + P_2 \cdot D_2^{(=1)} + P_3 \cdot D_3^{(=1)} + \\
 &\quad + P_4 \cdot D_4^{(=0)} + P_5 \cdot D_5^{(=1)} + P_6 \cdot D_6^{(=0)} + P_7 \cdot D_7^{(=0)} = \quad (2.113) \\
 &= P_0 + P_2 + P_3 + P_5.
 \end{aligned}$$

În acest mod la ieșirea Y se obține expresia funcției f . Intrarea \bar{G} se conectează la 0 logic pentru a permite funcționarea circuitului.

În cazul în care se dorește obținerea expresiei funcției f de la ieșirea \bar{W} (complementată) sau atunci când un multiplexor dispune doar de o astfel de ieșire se poate proceda în felul următor:

- se poate utiliza metoda de mai sus, dar pentru a nu se obține, la ieșire funcția negată, \bar{f} , se va adăuga o poartă logică inversoare, aşa cum este prezentat în figura 2.30,b (soluția nu este nici elegantă și nici economică);
- se „activează“ (1 logic), de această dată, intrările de date corespunzătoare termenilor canonici care nu intervin în expresia funcției, luându-se de fapt în considerare negata acesteia; pentru obținerea funcției adevărate este nevoie de încă o negare, care se realizează prin utilizarea ieșirii complementate (v. fig. 2.30,c):

$$\begin{aligned}
 f^{\text{FCD}}(x_1, x_2, x_3) &= \bar{f}^{\text{FCD}}(x_1, x_2, x_3) = \overline{P_1 + P_4 + P_6 + P_7} = \\
 &= \overline{\bar{x}_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 + x_1x_2\bar{x}_3 + x_1x_2x_3} = \\
 &= (x_1 + x_2 + \bar{x}_3)(\bar{x}_1 + x_2 + x_3)(\bar{x}_1 + \bar{x}_2 + x_3)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3) = \quad (2.114) \\
 &= (x_1x_3 + \bar{x}_1\bar{x}_3 + x_2)(\bar{x}_1 + \bar{x}_2) = \\
 &= \bar{x}_1\bar{x}_3 + \bar{x}_1x_2 + x_1\bar{x}_2x_3 + \bar{x}_1\bar{x}_2\bar{x}_3 = \\
 &= \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 = P_0 + P_2 + P_3 + P_5.
 \end{aligned}$$

Implementarea unui CLC descris de un ansamblu de funcții logice, cu multiplexoare este costisitoare deoarece necesită câte un circuit integrat MSI pentru fiecare funcție.

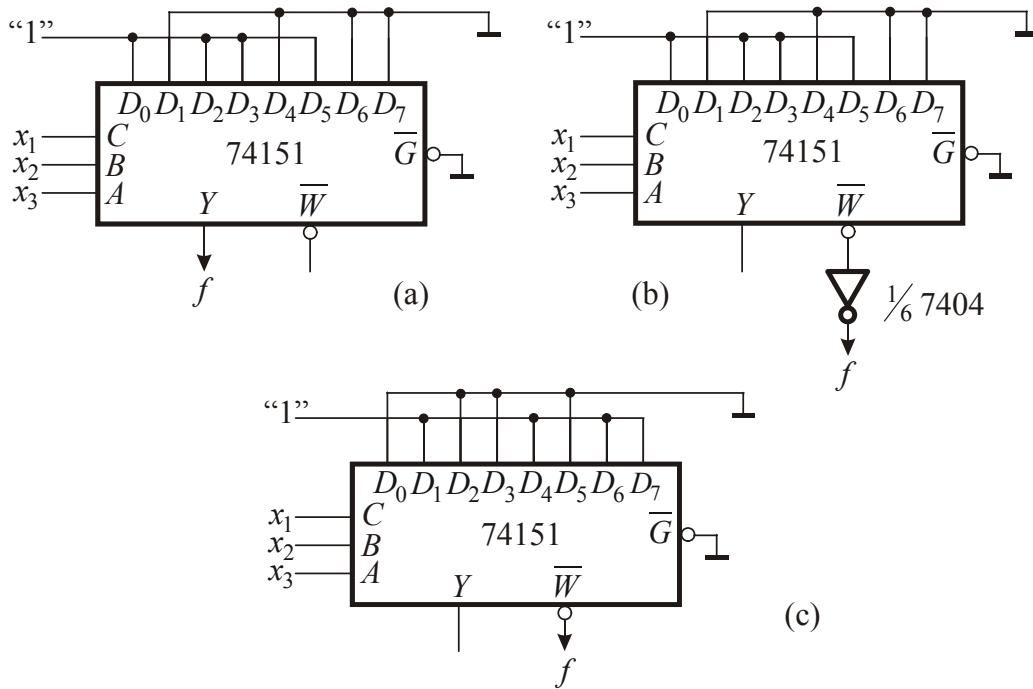


Fig.2.30 Modalități de implementare a funcției f cu MUX $2^3 : 1$, 74151: (a) utilizând ieșirea necomplementată Y ; (b), (c) utilizând ieșirea complementată \bar{W} .

Exemplu. Să se implementeze cu MUX-uri de 8 căi, CLC-ul descris de următorul sistem de funcții:

$$\begin{aligned} f_1^{\text{FCD}}(x_1, x_2, x_3) &= P_0 + P_1 + P_4 + P_6 + P_7, \\ f_2^{\text{FCD}}(x_1, x_2, x_3) &= P_0 + P_2 + P_3 + P_5. \end{aligned} \quad (2.115)$$

Soluția este prezentată în figura 2.31.

O funcție logică de n variabile poate fi implementată și cu un MUX $2^{n-1} : 1$. Această soluție de implementare este mai economică decât aceea în care se folosesc un MUX $2^n : 1$. În acest caz, metoda de implementare este următoarea: dintre cele n variabile ale funcției se separă $n-1$, având ponderile cele mai mari, care vor constitui intrările de selecție ale MUX-ului, variabila rămasă aplicându-se pe intrările de date, în acest scop, ea trebuind să fie comparată cu valorile funcției.

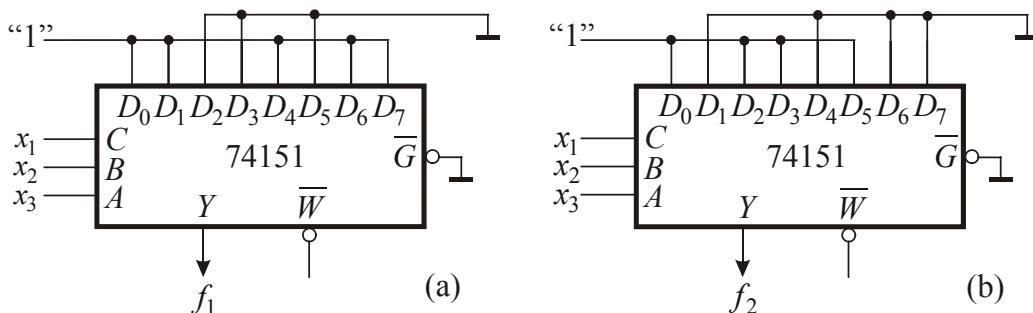


Fig.2.31 Implementarea cu MUX-uri $2^3 : 1$ a sistemului de funcții din exemplu.

Exemplu. Să se implementeze funcția având expresia dată de ecuația (2.112), cu un MUX $2^2 : 1$.

Pentru rezolvarea acestei probleme se pornește de la tabelul de adevăr al funcției (v. tab. 2.16) în care se separă două dintre variabile, x_1 și x_2 , care vor reprezenta, așa cum s-a precizat, intrările de selecție ale MUX-ului, în timp ce variabila x_3 , se compară cu valorile lui f , jucând rolul unei intrări de validare⁸:

- din primele două linii ale tabelului: $x_1x_2 = 00$; se observă că funcția $f = 1$, pentru $x_3 = 0$, deci $f = \bar{x}_3$ și la intrarea de date C_0 se va aplica \bar{x}_3 ;
- din următoarele două linii ale tabelului: $x_1x_2 = 01$; se observă că $f = 1$ indiferent de valoarea lui x_3 ; deci la intrarea C_1 a MUX-ului se aplică 1;
- pentru următoarele două linii din tabel: $x_1x_2 = 10$; funcția f are valoarea 1 pentru $x_3 = 1$ ($f = x_3$), deci la intrarea C_2 a MUX-ului se aplică x_3 ;
- ultimele două linii ale tabelului au: $x_1x_2 = 11$; funcția f este 0 indiferent de valoarea lui x_3 , deci la intrarea C_3 a MUX-ului se aplică 0.

Pentru implementare se folosește circuitul integrat 74153 (v. fig. 2.32,a).

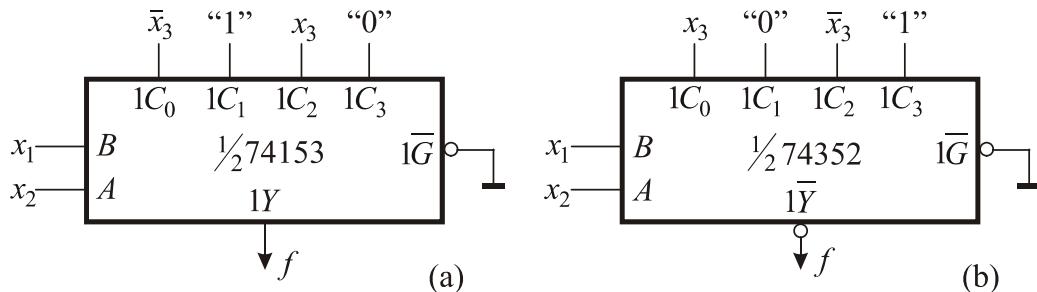


Fig.2.32 Implementarea cu MUX-uri $2^2 : 1$ a funcției din exemplu.

Avându-se în vedere expresia ieșirii multiplexorului se poate verifica corectitudinea raționamentului făcut:

$$\begin{aligned} f^{\text{FCD}}(x_1, x_2, x_3) &= Y = \overline{B}\overline{A}C_0 + \overline{B}AC_1 + B\overline{A}C_2 + BAC_3 = \\ &= \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2 \cdot 1 + x_1\bar{x}_2x_3 + x_1x_2 \cdot 0 = \\ &= P_0 + P_2 + P_3 + P_5. \end{aligned} \quad (2.116)$$

Dacă avem la dispoziție, pentru implementarea funcției, un MUX $2^2 : 1$ care nu are decât ieșire complementată (v. fig. 2.32,b) se procedează în mod asemănător, cu observația că, de această dată, se vor lua în considerare, din tabelul de adevăr, zerourile funcției:

- în primele două linii ale tabelului: $x_1x_2 = 00$, iar funcția are valoarea 0 pentru

Tab.2.16 Tabelul de adevăr al funcției.

	x_1	x_2	x_3	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

⁸ În cazul în care implementarea se face cu un multiplexor cu ieșirea necomplementată, se vor lua în considerare, din tabelul de adevăr, valorile 1 ale funcției.

$x_3 = 1$; deci $\bar{f} = x_3$, semnal ce se aplică pe intrarea de date C_0 ;

- pentru următoarele două linii: $x_1x_2 = 01$; funcția $f = 1$, indiferent de valoarea variabilei x_3 , deci la intrarea C_1 a MUX-ului se aplică 0;

- următoarele două linii ale tabelului au: $x_1x_2 = 10$; funcția \bar{f} se obține pentru \bar{x}_3 , rezultă deci $\bar{f} = \bar{x}_3$, semnal aplicat intrării de date C_2 ;

- pentru ultimele două linii: $x_1x_2 = 11$, iar funcția studiată are valoarea 0, indiferent de valoarea variabilei x_3 ; la borna C_3 se va conecta semnalul logic 1.

La aceleași rezultate se poate ajunge și algebric, pornind de la FCD a funcției date, respectiv a negației acestora și separând, pe același principiu, una dintre variabile. Astfel:

$$\begin{aligned} f^{\text{FCD}}(x_1, x_2, x_3) &= \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 = \\ &= (\bar{x}_1\bar{x}_2)\bar{x}_3 + (\bar{x}_1x_2)\bar{x}_3 + (\bar{x}_1x_2)x_3 + (x_1\bar{x}_2)x_3 = \\ &= P'_0 \cdot \bar{x}_3 + P'_1 \cdot 1 + P'_2 \cdot x_3 + P'_3 \cdot 0 \end{aligned} \quad (2.117)$$

și

$$\begin{aligned} \bar{f}^{\text{FCD}}(x_1, x_2, x_3) &= \bar{x}_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 + x_1x_2\bar{x}_3 + x_1x_2x_3 = \\ &= (\bar{x}_1\bar{x}_2)x_3 + (x_1\bar{x}_2)\bar{x}_3 + (x_1x_2)\bar{x}_3 + (x_1x_2)x_3 = \\ &= P'_0 \cdot x_3 + P'_1 \cdot 0 + P'_2 \cdot \bar{x}_3 + P'_3 \cdot 1. \end{aligned} \quad (2.118)$$

Într-o serie de aplicații sunt necesare multiplexoare având un număr mai mare de intrări/căi decât modulele integrate standard existente (este cazul implementării unor funcții booleene având un număr mai mare de variabile). Deși numărul maxim de intrări a multiplexoarelor integrate standard este de 16, se pot construi multiplexoare cu dimensiuni oricără de mari interconectând mai multe astfel de circuite în rețele arborescente. Extinderea capacitații de multiplexare se poate realiza prin diverse tehnici, toate având la bază conectarea în paralel sau în cascadă a unui număr corespunzător de multiplexoare standard.

Exemplu. Să se realizeze o extindere a capacitații de multiplexare de la 8 la 16 căi.

Pentru obținerea unui multiplexor de 16 căi se vor utiliza două multiplexoare de 8 căi (74151) conectate în paralel, pe intrările de selectare și strobare și porți logice (v. fig. 2.33). Extinderea multiplexării la 16 căi necesită o intrare suplimentară pentru selectare. În acest scop se vor folosi intrările \bar{G} de la cele două multiplexoare, la borna comună aplicându-se bitul cel mai semnificativ (al patrulea) al cuvântului de selectare. Cu schema logică prezentată în figura 2.33, MUX-ul I este selectat cu un cuvânt de forma 0xxx (deci de la 0÷7), iar MUX-ul II cu un cuvânt 1xxx (de la 8÷15).

Exemplu. Să se realizeze funcția f de 5 variabile cu 2 etaje de multiplexoare:

$$f^{\text{FCD}}(x_1, x_2, x_3, x_4, x_5) = P_0 + P_2 + P_4 + P_5 + P_{11} + P_{20} + P_{24} + P_{25} + P_{31}. \quad (2.119)$$

Soluția este evidentă dacă se întocmește tabelul de adevar al funcției (v. tab. 2.17).

Se separă variabilele funcției în două grupe, x_1x_2 respectiv $x_3x_4x_5$ și se implementează la primul etaj, cu 4 MUX-uri de 4 căi, grupele $x_3x_4x_5$ definite de fiecare configurație x_1x_2 . Rezultatele multiplexării de la primul etaj se adună apoi, cu ajutorul

altui MUX de 4 căi, în etajul al doilea. Pentru implementare se folosește circuitul integrat 74153 (2×MUX 2² :1).

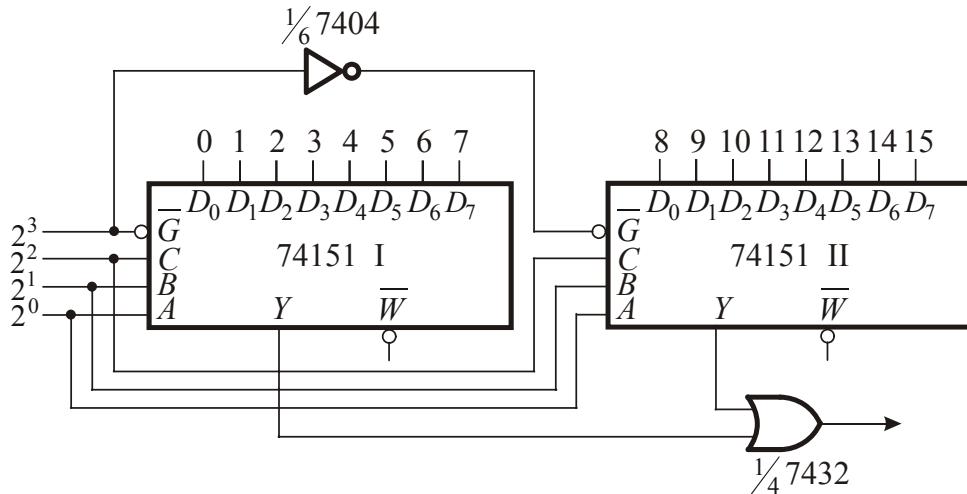


Fig.2.33 Realizarea unui MUX 2⁴ : 1 prin conectarea în paralel a două MUX-uri 2³ : 1.

Tab.2.17 Tabelul de adevăr al funcției f .

	x_1	x_2	x_3	x_4	x_5	f		x_1	x_2	x_3	x_4	x_5	f
0	0	0	0	0	0	1	16	1	0	0	0	0	0
1	0	0	0	0	1	0	17	1	0	0	0	1	0
2	0	0	0	1	0	1	18	1	0	0	1	0	0
3	0	0	0	1	1	0	19	1	0	0	1	1	0
4	0	0	1	0	0	1	20	1	0	1	0	0	1
5	0	0	1	0	1	1	21	1	0	1	0	1	0
6	0	0	1	1	0	0	22	1	0	1	1	0	0
7	0	0	1	1	1	0	23	1	0	1	1	1	0
8	0	1	0	0	0	0	24	1	1	0	0	0	1
9	0	1	0	0	1	0	25	1	1	0	0	1	1
10	0	1	0	1	0	0	26	1	1	0	1	0	0
11	0	1	0	1	1	1	27	1	1	0	1	1	0
12	0	1	1	0	0	0	28	1	1	1	0	0	0
13	0	1	1	0	1	0	29	1	1	1	0	1	0
14	0	1	1	1	0	0	30	1	1	1	1	0	0
15	0	1	1	1	1	0	31	1	1	1	1	1	1

1. Configurația $x_1x_2 = 00$.

$$x_3x_4 = 00 \quad f = 1 \text{ pentru } x_5 = 0, \quad 1C_0^1 = \bar{x}_5,$$

$$x_3x_4 = 01 \quad f = 1 \text{ pentru } x_5 = 0, \quad 1C_1^1 = \bar{x}_5,$$

$$x_3x_4 = 10 \quad f = 1 \quad \forall x_5, \quad 1C_2^1 = 1,$$

$$x_3x_4 = 11 \quad f = 0 \quad \forall x_5, \quad 1C_3^1 = 0.$$

2. Configurația $x_1x_2 = 01$.

$$x_3x_4 = 00 \quad f = 0 \quad \forall x_5, \quad 2C_0^2 = 0,$$

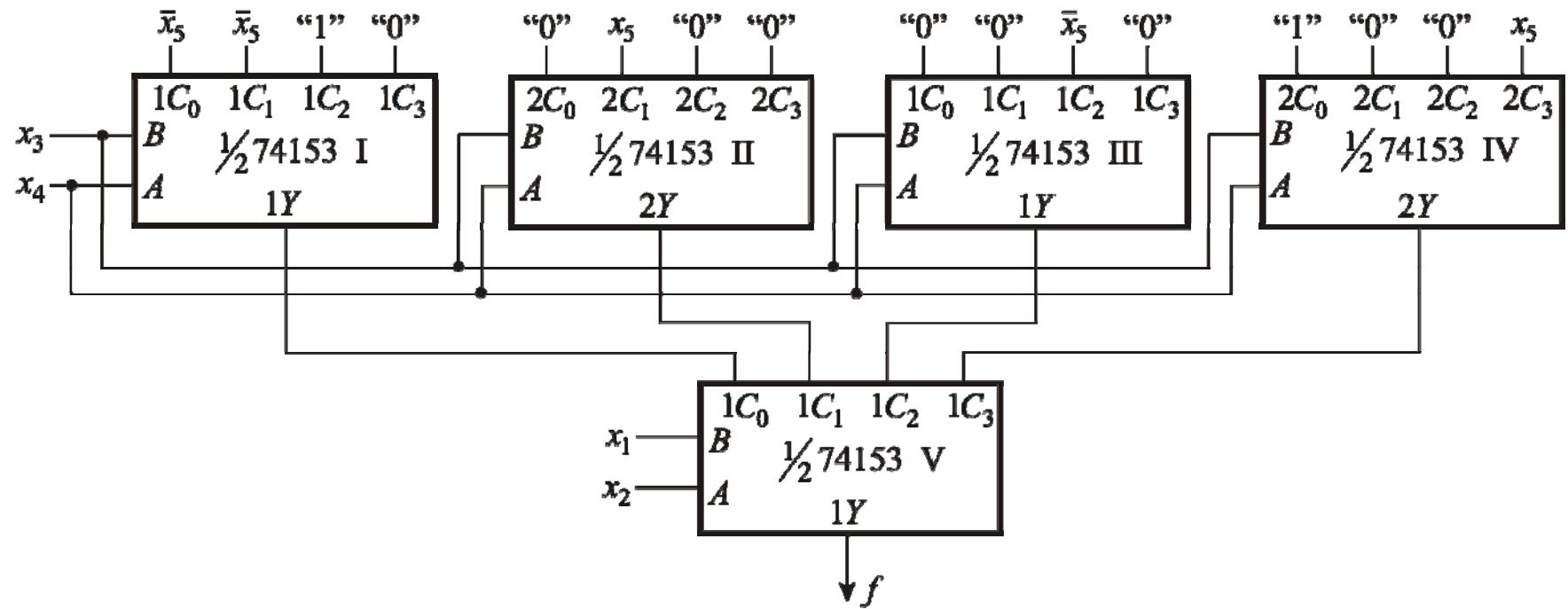


Fig.2.34 Implementarea cu MUX-uri $2^2 : 1$ a funcției f .

$$x_3x_4 = 01 \quad f = 1 \text{ pentru } x_5 = 1, \quad 2C_1^2 = x_5,$$

$$x_3x_4 = 10 \quad f = 0 \quad \forall x_5, \quad 2C_2^2 = 0,$$

$$x_3x_4 = 11 \quad f = 0 \quad \forall x_5, \quad 2C_3^2 = 0.$$

3. Configurația $x_1x_2 = 10$.

$$x_3x_4 = 00 \quad f = 0 \quad \forall x_5, \quad 1C_0^3 = 0,$$

$$x_3x_4 = 01 \quad f = 0 \quad \forall x_5, \quad 1C_1^3 = 0,$$

$$x_3x_4 = 10 \quad f = 1 \text{ pentru } x_5 = 0, \quad 1C_2^3 = \bar{x}_5,$$

$$x_3x_4 = 11 \quad f = 0 \quad \forall x_5, \quad 1C_3^3 = 0.$$

4. Configurația $x_1x_2 = 11$.

$$x_3x_4 = 00 \quad f = 1 \quad \forall x_5, \quad 2C_0^4 = 1,$$

$$x_3x_4 = 01 \quad f = 0 \quad \forall x_5, \quad 2C_1^4 = 0,$$

$$x_3x_4 = 10 \quad f = 0 \quad \forall x_5, \quad 2C_2^4 = 0,$$

$$x_3x_4 = 11 \quad f = 1 \text{ pentru } x_5 = 1, \quad 2C_3^4 = x_5.$$

Schema logică obținută în urma implementării funcției f este prezentată în figura 2.34.

Este evident că funcția se implementează mai economic cu un MUX de 16 cai (v. fig. 2.35). S-a tratat acest exemplu doar pentru ilustrarea simplificată a metodei care este aplicabilă în cazul funcțiilor cu șase sau mai multe variabile.

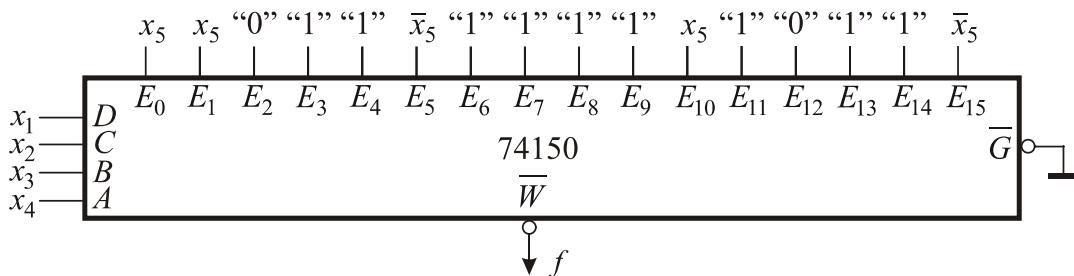


Fig.2.35 Implementarea cu MUX $2^4:1$ a funcției f .

• Circuite decodificatoare

Decodificatoarele (DCD-urile) reprezintă o clasă de circuite logice combinatoriale care, în cazul general, au n intrări, X_0, X_1, \dots, X_{n-1} , pe care se aplică cei n biți ai cuvântului de cod și m ieșiri ($m \leq 2^n$), Z_0, Z_1, \dots, Z_{m-1} (v. fig. 2.36,a); pentru fiecare cuvânt de cod aplicat la intrare se obține semnal logic doar pe una dintre ieșiri sau, se poate spune că decodificatorul servește la identificarea unui cod de intrare prin activarea unei singure liniilor de ieșire, corespunzătoare acestui cod [2,15,22,23,26÷30,33]. Decodificarea este necesară în numeroase aplicații cum sunt: adresarea memorii, afișarea numerică, multiplexarea datelor, etc.

În cele ce urmează va fi examinat decodificatorul BCD 8421-ZECIMAL, cu $n = 4$ intrări și $m = 10$ ieșiri, a cărui schemă-bloc este prezentată în figura 2.36,b.

Codul *BCD* (Binary Coded Decimal) 8421 este un cod binar zecimal ponderat care utilizează numai primele 10 combinații binare din cele 16, care se pot forma cu 4 biți în

sistemul de numerație binar natural. Cele 6 combinații de cod neutilizate (1010, 1011, 1100, 1101, 1110 și 1111) sunt combinații indiferente (v. Anexa C).

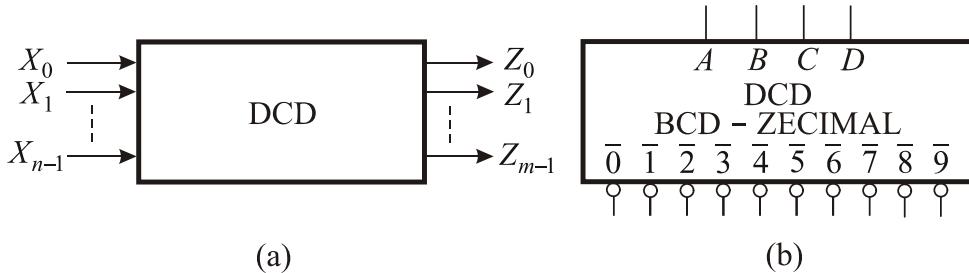


Fig.2.36 Schemele-bloc pentru un DCD cu n intrări și m ieșiri (a), respectiv pentru un DCD BCD-ZECIMAL (b).

Comportarea acestui tip de decodificator este descrisă în tabelul 2.18.

Tab.2.18 Tabelul de adevar pentru un DCD BCD-ZECIMAL.

Intrări BCD				Ieșiri - zecimal									
D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

DC BA	00 01 11 10					00 01 11 10					00 01 11 10				
	00	01	11	10	$\bar{0}$	00	01	11	10	$\bar{1} \dots$	00	01	11	10	$\dots \bar{9}$
00	0	1	*	1	0	1	*	1	1	1	1	1	*	1	1
01	1	1	*	1	1	1	*	1	1	1	1	1	*	1	1
11	1	1	*	*	1	1	*	1	1	1	1	1	*	1	1
10	1	1	*	*	1	1	*	1	1	1	1	1	*	1	1

Fig.2.37 Matricele de comandă pentru DCD-ul BCD-ZECIMAL.

Se observă, din tabelul de adevăr, că deoarece toate stările sunt decodificate explicit, datele false (combinăriile indiferente) fixate eventual pe intrările decodificatorului duc la stabilirea tuturor ieșirilor în starea falsă de 1 logic. Se spune că acest decodificator rejectează datele false de la intrare. Pe baza tabelului de adevăr se pot întocmi diagramele Karnaugh ale ieșirilor (v. fig. 2.37), numite și *matrice de comandă*. Pentru o decodificare simplă, de obicei, nu este necesar să se completeze câte o matrice de comandă pentru fiecare ieșire; în acest caz se va întocmi aşa numita *matrice de referință* sau de *decodificare*, în locațiile căreia sunt înscrise direct funcțiile de ieșire (v. fig. 2.38).

		DC	00	01	11	10
		BA	00	01	11	10
00			0	4	*	8
01			1	5	*	9
11			3	7	*	*
10			2	6	*	*

Fig.2.38 Matricea de referință pentru DCD-ul BCD-ZECIMAL.

Din aceste matrice rezultă expresiile booleene ale funcțiilor de ieșire:

$$\begin{aligned} \bar{0} &= \overline{\overline{DCB}}A, & \bar{5} &= \overline{\overline{DCB}}\bar{A}, \\ \bar{1} &= \overline{\overline{DCB}}A, & \bar{6} &= \overline{\overline{DCB}}\bar{A}, \\ \bar{2} &= \overline{\overline{DCB}}\bar{A}, & \bar{7} &= \overline{\overline{DCB}}A, \\ \bar{3} &= \overline{\overline{DCB}}A, & \bar{8} &= \overline{\overline{DCB}}\bar{A}, \\ \bar{4} &= \overline{\overline{DCB}}\bar{A}, & \bar{9} &= \overline{\overline{DCB}}A, \end{aligned} \quad (2.120)$$

care permit întocmirea schemei logice a decodificatorului, prezentă în figura 2.39.

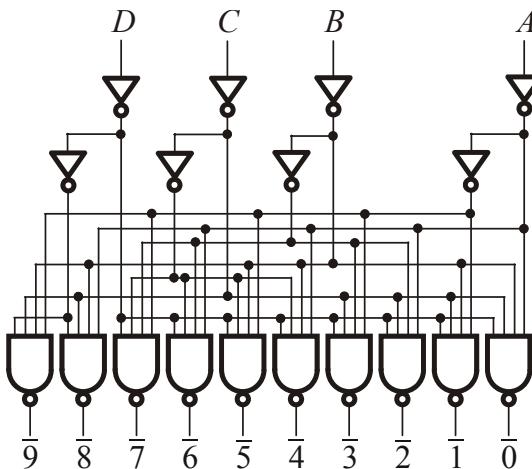


Fig.2.39 Schema logică a DCD-ului BCD-ZECIMAL.

Din ecuațiile (2.120) rezultă că decodificatorul realizează pe fiecare ieșire un termen canonic. Decodificatoarele complete, la care $m = 2^n$, generează la ieșire toți termenii canonici care se pot forma cu cele n variabile de intrare, sau, cu alte cuvinte, fiecare din cele 2^n ieșiri realizează una dintre funcțiile elementare minterm de n variabile, de aici rezultând și posibilitatea utilizării acestor circuite pentru implementarea funcțiilor logice.

Principalele tipuri de DCD-uri utilizate în mod curent sunt următoarele: DCD-ul BCD-ZECIMAL (4 intrări, 10 ieșiri), DCD-ul BCD Exces 3-ZECIMAL (4 intrări, 10 ieșiri), DCD-ul BCD Cod Gray Exces 3-ZECIMAL (4 intrări, 10 ieșiri); de asemenea, deoarece aşa cum se va vedea mai jos, constructiv și funcțional, decodificatoarele și demultiplexoarele se aseamănă, majoritatea circuitelor integrate conțin DCD/DMUX-uri, de

diverse dimensiuni.

Este de menționat că denumirea de decodificator este utilizată impropriu și pentru convertoarele de cod BCD-cod 7 segmente [27].

• Circuite demultiplexoare

Circuitele demultiplexoare (DMUX-urile) realizează operația inversă multiplexării și anume distribuie un semnal (0 sau 1) de pe o cale, pe mai multe căi, în funcție de adresa acelor căi, motiv pentru care se mai numesc și circuite distribuitoare [2,15,22,23,26÷30, 33].

Demultiplexorul este un circuit logic combinațional care are, în cazul general, o intrare de date I , n intrări de selecție (adresă), S_0, S_1, \dots, S_{n-1} și 2^n ieșiri, $Z_0, Z_1, \dots, Z_{2^n-1}$ (v. fig. 2.40,a). Expresia ieșirii Z_j , pentru $j = 0, 1, \dots, 2^n - 1$, este dată de relația:

$$Z_j = \begin{cases} 0, & \text{pentru } j \neq k, \\ I, & \text{pentru } j = k, \end{cases} \quad (2.121)$$

unde k este echivalentul zecimal al numărului binar reprezentat de stările 0 și 1 ale intrărilor de selectare:

$$k = (S_{n-1}S_{n-2}\dots S_1S_0). \quad (2.122)$$

Se va analiza în continuare un demultiplexor de 8 căi (DMUX 1:2³), cu o intrare de date/strobare, \bar{G} , activă pe 0 logic, trei intrări de selecție A , B și C și opt ieșiri, $\bar{Y}_0, \bar{Y}_1, \dots, \bar{Y}_7$, active pe 0 logic (v. fig. 2.40,b). Comportarea sa este descrisă în tabelul 2.19, prezentat mai jos.

Plecând de la tabelul de adevăr se întocmește matricea de referință din figura 2.40,c, din care se pot scrie expresiile ieșirilor circuitului:

$$\begin{aligned} \bar{Y}_0 &= \overline{\overline{CBAG}}, & \bar{Y}_4 &= \overline{\overline{CBA\bar{G}}}, \\ \bar{Y}_1 &= \overline{\overline{C\bar{B}AG}}, & \bar{Y}_5 &= \overline{\overline{C\bar{B}A\bar{G}}}, \\ \bar{Y}_2 &= \overline{\overline{C\bar{B}\bar{A}G}}, & \bar{Y}_6 &= \overline{\overline{CB\bar{A}G}}, \\ \bar{Y}_3 &= \overline{\overline{C\bar{B}A\bar{G}}}, & \bar{Y}_7 &= \overline{\overline{CBA\bar{G}}}. \end{aligned} \quad (2.123)$$

Din relațiile (2.123) se observă că la ieșirea demultiplexorului se obțin toți termenii canonici care se pot forma cu variabilele de selectare înmulțiti cu variabila corespunzătoare intrării de date/strobare (realizează numai nivelul logic ȘI).

Tabelele 2.18 și 2.19 precum și relațiile (2.120) și (2.123) pun în evidență similaritatea funcțională a decodificatoarelor și demultiplexoarelor, ceea ce face ca, în anumite situații, decodificatoarele să poată fi folosite ca demultiplexoare și invers.

În figura 2.41 se prezintă schema logică a unui demultiplexor de 8 căi.

Principalele tipuri de DMUX-uri integrate utilizate în mod curent sunt: DCD//DMUX-urile 1:16, 1:8, 2×(1:4).

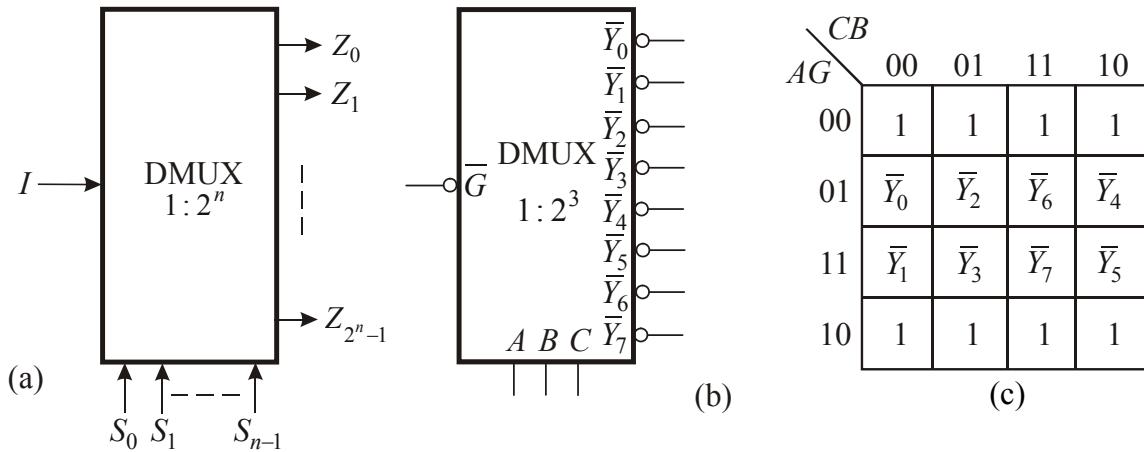


Fig.2.40 Circuitul demultiplexor: (a) schema-bloc pentru un DMUX $1:2^n$; (b), (c) schema-bloc și matricea de referință pentru un DMUX $1:2^3$.

Tab.2.19 Tabelul de adevăr pentru un DMUX $1:2^3$.

Intrări			Ieșiri								
Selectie			Strob/date	\bar{Y}_0	\bar{Y}_1	\bar{Y}_2	\bar{Y}_3	\bar{Y}_4	\bar{Y}_5	\bar{Y}_6	\bar{Y}_7
C	B	A	\bar{G}	1	1	1	1	1	1	1	1
*	*	*	0	0	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	1
0	1	0	0	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	0	1	1	1	1
1	0	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	0	1	1
1	1	0	0	1	1	1	1	1	1	0	1
1	1	1	0	1	1	1	1	1	1	1	0

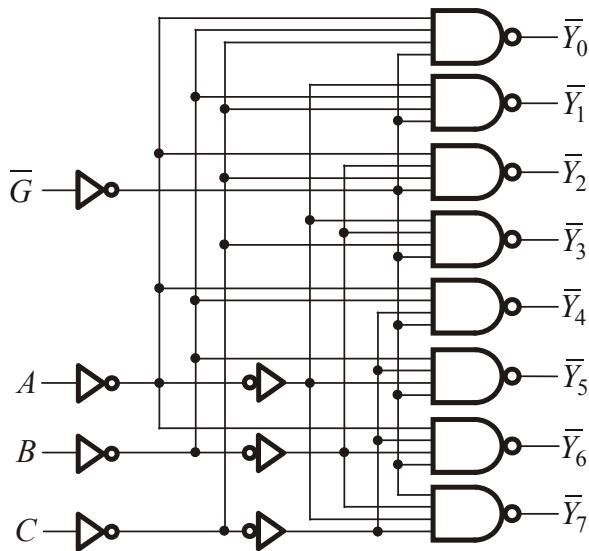


Fig.2.41 Schema logică a DMUX-ului $1:2^3$.

- Implementarea funcțiilor booleene cu decodificatoare și demultiplexoare

Orice funcție logică de n variabile, dată sub FCD, se poate implementa cu ajutorul unui DCD cu n intrări sau a unui DMUX $1:2^n$, dacă la ieșirea acestor circuite termenii canonici (nivelul logic řI), existenți în tabelul de adevăr al funcției sunt introdusi într-o poartă logică pentru realizarea nivelului logic SAU (acest nivel logic trebuie atașat din exterior). Este evident că și în cazul acestui tip de implementare nu este necesară minimizarea matematică a funcțiilor logice [2,15,22,23,26÷30,33].

Problemele specifice legate de implementarea cu decodificatoare, respectiv demultiplexoare vor fi puse în evidență prin tratarea unor exemple. În acest scop se vor folosi următoarele decodificatoare și demultiplexoare integrate [34]:

- 7442, DCD BCD-ZECIMAL, cu 4 intrări și 10 ieșiri active pe 0 logic;
- 74154, DMUX $1:2^4$, cu 2 intrări de validare/strobare (\bar{G}_1, \bar{G}_2) și 16 ieșiri, toate active pe 0 logic;
- 74138, DMUX $1:2^3$; la acest circuit este disponibilă o funcție combinațională suplimentară la intrarea de validare/date, folosind semnalele aplicate pe intrările de validare G_1 (activă pe 1 logic), \bar{G}_2A și \bar{G}_2B (active pe 0 logic); cele 8 ieșiri sunt active pe 0 logic;
 - 74155, $2\times$ DMUX $1:2^2$, cu câte o intrare de validare/strobare activă pe 0 logic ($1\bar{G}, 2\bar{G}$), cu două intrări de date ($1C$, activă pe 1 logic și $2\bar{C}$ activă pe 0 logic) și cu câte patru ieșiri active pe 0 logic;
 - 74139, $2\times$ DMUX $1:2^2$, cu două intrări de validare/strobare sau date, active pe 0 logic ($1\bar{G}, 2\bar{G}$) și cu câte patru ieșiri active pe 1 logic.

Exemplu. Să se implementeze cu DCD, respectiv DMUX și porți logice funcția:

$$f^{\text{FCD}}(x_1, x_2, x_3) = P_0 + P_2 + P_5 + P_6. \quad (2.124)$$

Deoarece funcția are $n=3$ variabile, este necesar un DCD cu $n=3$ intrări, respectiv un DMUX $1:2^3$. Se va folosi, într-o primă variantă, DCD-ul 7442, respectiv DMUX-ul $1:2^3$ 74138, iar în varianta a doua circuitul integrat 74155 ($2\times$ DMUX $1:2^2$). Trebuie avut în vedere că majoritatea decodificatoarelor și demultiplexoarelor integrate realizează funcțiile minterm negate.

DCD-ul 7442 are, aşa cum s-a precizat, 4 intrări și 10 ieșiri active pe 0 logic; din tabelul de adevăr al decodificatorului (v. tab. 2.18) se observă că intrarea având ponderea cea mai mare (D) este 0 pentru primele 8 ieșiri.

Problema utilizării DCD-ului 7442 poate fi pusă și în următorii termeni: cum se poate obține un DMUX de 8 căi folosindu-se un DCD 7442? În acest scop, dintre cele 4 intrări ale DCD-ului, C, B, A vor fi folosite ca intrări de selecție, iar cea de a patra (D), ca intrare de strobare/date, în timp ce dintre cele 10 ieșiri sunt necesare doar primele 8, celelalte două trebuind să fie neutralizate (v. fig. 2.42,a).

Pentru a se genera termenii canonici de trei variabile, variabilele funcției f trebuie aplicate pe intrările de date respectiv pe intrările de selecție ale DCD/DMUX-ului (v. ec. (2.120), (2.123)).

Selectarea doar a acelor termeni canonici care intervin în expresia funcției de imple-

mentat se poate face, într-o primă variantă, cu o poartă řI-NU la intrările căreia trebuie conectate ieșirile DCD/DMUX-ului corespunzătoare termenilor canonici menționati mai sus (v. fig. 2.42,a):

$$f^{\text{FCD}}(x_1, x_2, x_3) = \overline{P_0 \cdot P_2 \cdot P_5 \cdot P_6} = P_0 + P_2 + P_5 + P_6. \quad (2.125)$$

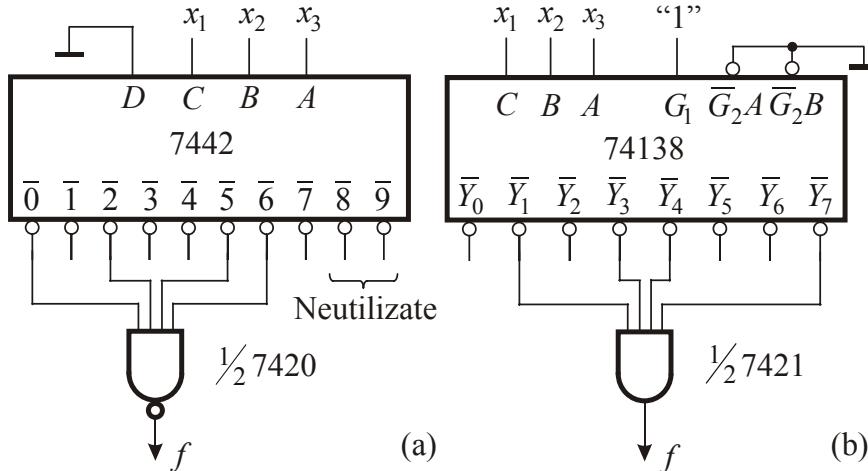


Fig.2.42 Realizarea funcției f din exemplu: (a) cu DCD 7442 și porți řI-NU; (b) cu DMUX 1:2³ 74138 și porți řI.

O soluție alternativă este folosirea unei porți řI la intrările căreia se conectează ieșirile DCD/DMUX-ului corespunzătoare termenilor canonici care nu intervin în expresia funcției considerate (se va lua deci în considerare negata funcției):

$$\bar{f}^{\text{FCD}}(x_1, x_2, x_3) = P_1 + P_3 + P_4 + P_7, \quad (2.126)$$

$$\begin{aligned} f^{\text{FCD}}(x_1, x_2, x_3) &= \bar{f}^{\text{FCD}}(x_1, x_2, x_3) = \overline{P_1 + P_3 + P_4 + P_7} = \\ &= \overline{P_1} \cdot \overline{P_3} \cdot \overline{P_4} \cdot \overline{P_7}. \end{aligned} \quad (2.127)$$

Această soluție de implementare este prezentată în figura 2.42,b.

Pentru cea de a doua variantă de implementare se va utiliza, aşa cum s-a menționat, circuitul integrat 74155 care conține două DMUX-uri 1:2². Comportarea acestor circuite este descrisă în tabelele 2.20,a,b. Prin câteva conexiuni simple, cele două DMUX-uri 1:2² se pot transforma într-un DMUX 1:2³, cu care se va realiza implementarea funcției. Astfel, cele două intrări de date 1C și 2C se vor conecta împreună devenind cea de a treia intrare de selectare (de ponderea cea mai mare) pentru DMUX-ul 1:2³, iar borna comună obținută prin legarea împreună a intrărilor de strobare 1G și 2G va reprezenta noua intrare de strobare sau date (v. fig. 2.43).

Deoarece intrarea de date 1C este activă pe 1 logic iar 2C pe 0 logic și având în vedere că dintre cele opt combinații binare care apar la intrare, primele patru au 0 pe poziția bitului celui mai semnificativ, iar următoarele patru 1, ieșirea 0 a DMUX-ului 1:2³ va coincide cu borna 2Y₀ și nu cu 1Y₀, pentru că al doilea DMUX 1:2² va fi autorizat să funcționeze mai întâi.

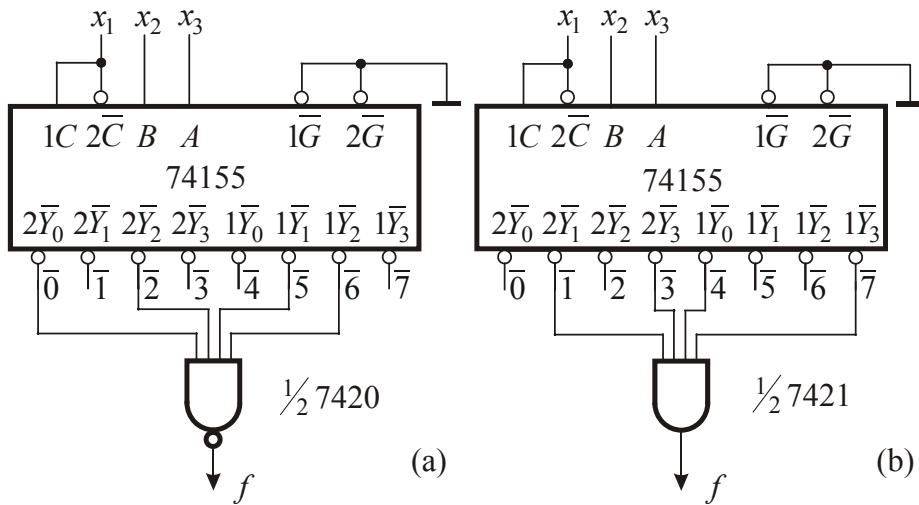
Tab.2.20 Tabelele de adevăr pentru $2 \times \text{DMUX}$ -uri $1:2^2$.

Intrări			Ieșiri				
Selectare		Strob	Date	$1\bar{Y}_0$	$1\bar{Y}_1$	$1\bar{Y}_2$	$1\bar{Y}_3$
B	A	$1\bar{G}$	$1C$				
*	*	1	*	1	1	1	1
0	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1
1	0	0	1	1	1	0	1
1	1	0	1	1	1	1	0
*	*	1	0	1	1	1	1

Intrări			Ieșiri				
Selectare		Strob	Date	$2\bar{Y}_0$	$2\bar{Y}_1$	$2\bar{Y}_2$	$2\bar{Y}_3$
B	A	$2\bar{G}$	$2\bar{C}$				
*	*	1	*	1	1	1	1
0	0	0	0	0	1	1	1
0	1	0	0	1	0	1	1
1	0	0	0	0	1	1	0
1	1	0	0	0	1	1	0
*	*	1	1	1	1	1	1

(a)

(b)


 Fig.2.43 Realizarea funcției f din exemplu: (a) cu porti SI-NU; (b) cu porti SI.

Realizarea nivelului logic SAU presupune utilizarea unei porti SI-NU (v. fig. 2.43, a) sau a unei porti SI (v. fig. 2.43,b), procedura fiind identică cu aceea expusă mai sus.

Pentru implementarea unui CLC definit de un sistem de funcții este necesar un singur DCD, respectiv DMUX și se adaugă câte o poartă logică pentru fiecare funcție din sistem.

Exemplu. Să se implementeze cu DMUX și porti logice CLC-ul descris de următorul sistem de funcții:

$$\begin{aligned}
 f_1^{\text{FMD}}(x_1, x_2, x_3, x_4) &= \bar{x}_1 \bar{x}_3 \bar{x}_4 + \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 x_4, \\
 f_2^{\text{FMD}}(x_1, x_2, x_3, x_4) &= x_1 \bar{x}_3 \bar{x}_4 + x_1 \bar{x}_2 x_3 + \bar{x}_1 \bar{x}_2 x_4 + \bar{x}_1 x_3 x_4, \\
 f_3^{\text{FMD}}(x_1, x_2, x_3, x_4) &= \bar{x}_1 \bar{x}_3 \bar{x}_4 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 x_4, \\
 f_4^{\text{FMD}}(x_1, x_2, x_3, x_4) &= x_1 x_4 + x_2 x_3 x_4.
 \end{aligned} \tag{2.128}$$

Mai întâi este necesar ca expresiile funcțiilor (2.128) să fie aduse la FCD:

$$\begin{aligned}
 f_1^{\text{FCD}}(x_1, x_2, x_3, x_4) &= P_0 + P_4 + P_5 + P_9, \\
 f_2^{\text{FCD}}(x_1, x_2, x_3, x_4) &= P_1 + P_3 + P_7 + P_8 + P_{10} + P_{11} + P_{12},
 \end{aligned} \tag{2.129}$$

$$f_3^{\text{FCD}}(x_1, x_2, x_3, x_4) = P_0 + P_3 + P_4 + P_5, \quad (2.129)$$

$$f_4^{\text{FCD}}(x_1, x_2, x_3, x_4) = P_7 + P_9 + P_{11} + P_{13} + P_{15}.$$

Pentru implementare se folosește DMUX-ul 1:2⁴, 74154, schema logică fiind prezentată în figura 2.44.

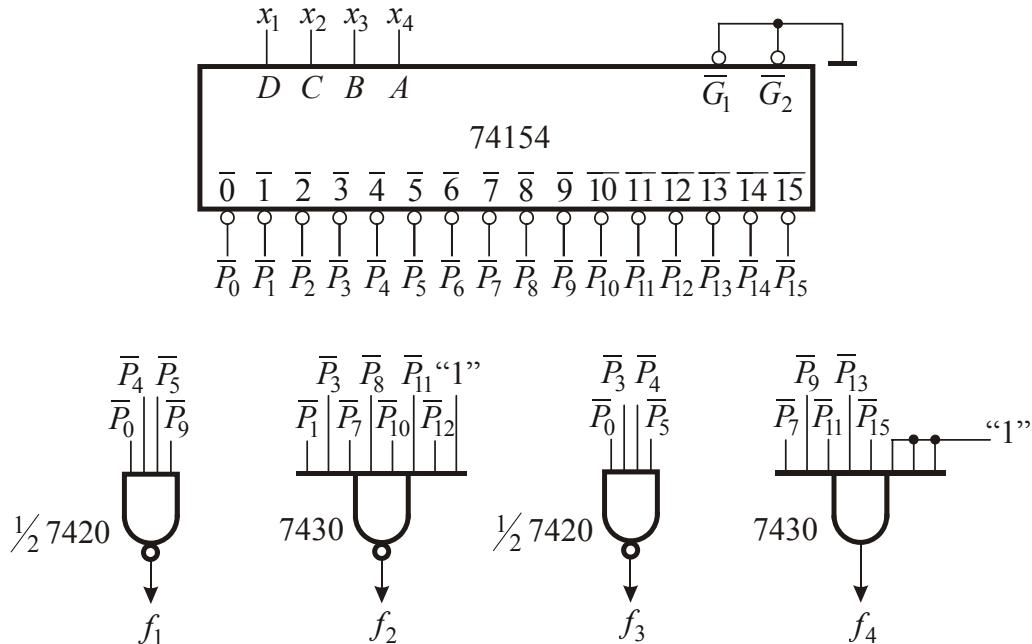


Fig.2.44 Implementarea cu DMUX 1:2⁴ a sistemului de funcții din exemplu.

Pentru implementarea cu circuite integrate MSI a sistemelor logice combinatoriale cu mai multe ieșiri se preferă soluția cu DCD/DMUX-uri și porți logice, în raport cu soluția de implementare cu MUX-uri, fiind mai economică.

O funcție logică având n variabile se poate realiza și cu un DCD respectiv DMUX de 2^{n-1} căi. Această metodă de implementare este mai economică decât aceea în care se folosește un DCD/DMUX de 2^n căi. Soluția acestei metode de implementare constă în a se separa, dintre cele n variabile ale funcției, de obicei, variabila având ponderea cea mai mare. Cele $n-1$ variabile rămase se aplică pe intrările DCD/DMUX-ului de 2^{n-1} căi, generând termeni canonici de $n-1$ variabile. Pentru a se obține însă termenii canonici (de n variabile) prezenti în expresia funcției de implementat, trebuie ca la termenii canonici de $n-1$ variabile să li se adauge variabila selectată. Acest lucru nu se poate face decât în exteriorul DCD/DMUX-ului, de obicei, prin utilizarea unei rețele de porți logice.

Exemplu. Să se realizeze cu un DMUX 1:2ⁿ⁻¹ o funcție cu n variabile de intrare. Se va considera exemplul concret al următoarei funcții de 3 variabile:

$$f^{\text{FCD}}(x_1, x_2, x_3) = P_1 + P_2 + P_6 + P_7 = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + x_1 x_2 \bar{x}_3 + x_1 x_2 x_3. \quad (2.130)$$

Tabelul de adevăr al funcției f este prezentat în tabelul 2.21.

Conform celor arătate mai sus, funcția f având 3 variabile se poate implementa cu un DCD/DMUX cu $2^3 = 8$ căi (7442 sau 74155). Problema cere însă să se realizeze cu

un DCD/DMUX de $2^2 = 4$ căi.

Tab.2.21 Tabelul de adevăr al funcției f .

	x_1	x_2	x_3	f
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Soluția constă, cum s-a mai precizat, în a se ține cont de cea de a treia variabilă (fie aceasta x_1), nu la intrarea DCD/DMUX-ului, ci la intrarea porțiilor logice ce selectează termenii canonici din expresia funcției. Pentru implementare se vor utiliza circuitele integrate 74139 ($2 \times$ DMUX 1: 2^2) cu ieșirile active în 1 logic (v. fig. 2.45,a), respectiv 74155 ($2 \times$ DMUX 1: 2^2) cu ieșirile active în 0 logic (v. fig. 2.45,b) și porți logice ȘI-NU.

Pentru a se demonstra corectitudinea schemelor logice obținute, se poate prelucra expresia funcției f , separând variabilele x_2 , x_3 și aplicând teoremele lui De Morgan, după cum urmează:

$$\begin{aligned} f^{\text{FCD}}(x_1, x_2, x_3) &= \bar{x}_1(\bar{x}_2 x_3) + \bar{x}_1(x_2 \bar{x}_3) + x_1(x_2 \bar{x}_3) + x_1(x_2 x_3) = \\ &= \bar{x}_1(\bar{x}_2 x_3) + x_2 \bar{x}_3 + x_1(x_2 x_3) = \\ &= \overline{\bar{x}_1 P'_1 + P'_2 + x_1 P'_3} = \overline{\bar{x}_1 P'_1 \cdot \overline{P'_2} \cdot \overline{x_1 P'_3}}, \end{aligned} \quad (2.131)$$

pentru cazul în care DCD/DMUX-ul pune la dispoziție ieșiri active pe 1 logic, respectiv:

$$\begin{aligned} f(x_1, x_2, x_3) &= \overline{\bar{x}_1 P'_1 \cdot \overline{P'_2} \cdot \overline{x_1 P'_3}} = \overline{(x_1 + \overline{P'_1}) \overline{P'_2} (\bar{x}_1 + \overline{P'_3})} = \\ &= \overline{(x_1 \bar{x}_1^{(=0)} + x_1 \overline{P'_3} + \bar{x}_1 \overline{P'_1} + \overline{P'_1} \overline{P'_3})} = \text{termen redundant} \overline{P'_2} = \\ &= \overline{(\bar{x}_1 \overline{P'_1} + x_1 \overline{P'_3}) \overline{P'_2}} = \overline{\bar{x}_1 \overline{P'_1} \overline{P'_2} + x_1 \overline{P'_2} \overline{P'_3}} = \\ &= \overline{\bar{x}_1 \overline{P'_1} \overline{P'_2} \cdot x_1 \overline{P'_2} \overline{P'_3}}, \end{aligned} \quad (2.132)$$

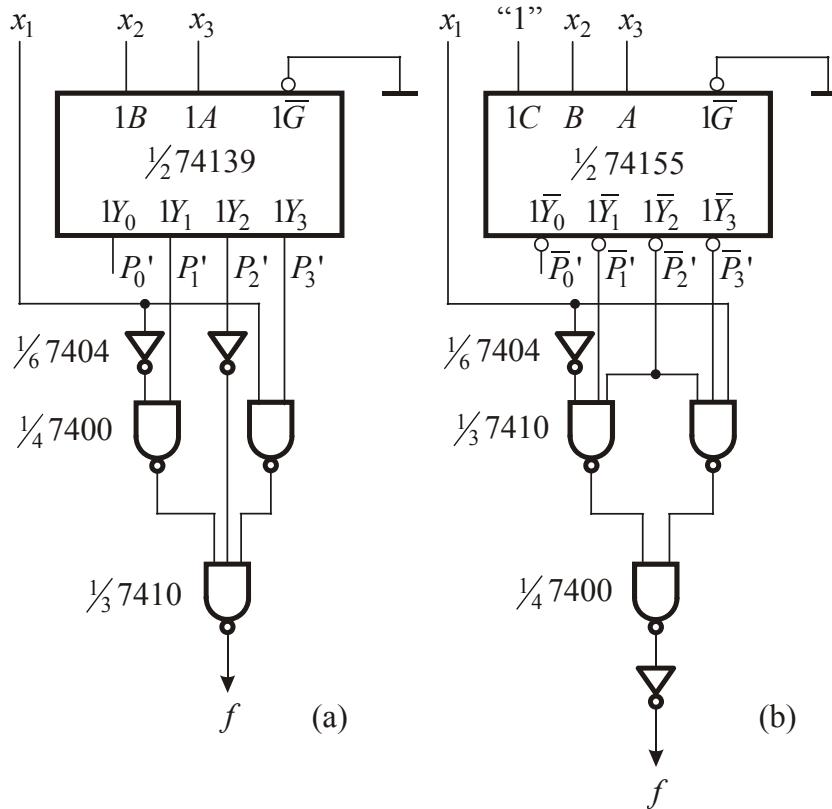
pentru cazul în care ieșirile sunt active în 0 logic.

În aplicații apare frecvent necesitatea extinderii capacitatii DCD/DMUX-urilor standard. Utilizând structuri standard conectate în paralel sau în cascadă se poate mări numărul liniilor de ieșire.

Exemplu. Să se realizeze un DCD cu 16 căi utilizându-se două DMUX-uri de 8 căi.

Schema logică care permite extinderea decodificării la 16 căi, prin utilizarea a două DMUX-uri de 8 căi, este prezentată în figura 2.46. Soluția este evidentă dacă se are în vedere tabelul de adevăr al unui DCD de 16 căi (v. tab. 2.22). Astfel, se observă că pentru primele 8 combinații ale valorilor variabilelor de intrare (pentru care bitul cel mai semnificativ $D = 0$), DCD-ul cu 16 căi poate fi obținut cu un DMUX de 8 căi funcționând normal și cu un DMUX de 8 căi cu funcționarea inhibată. Pentru următoarele 8 combinații ale valorilor variabilelor de intrare ($D = 1$) situația se inversează.

În concluzie se poate aprecia că se pot folosi două DMUX-uri de 8 căi lucrând alternativ după cum intrarea D este zero sau unu. A patra intrare (având ponderea cea mai mare) a DCD-ului cu 16 căi se obține prin utilizarea intrărilor $1\bar{G}$ și $2\bar{G}$ ale celor două demultiplexoare.


 Fig.2.45 Implementarea cu DMUX 1:2² a functiei din exemplu.

Tab.2.22 Tabelul de adevar pentru un DCD cu 4 intrari/16 ieşiri.

	Intrări				Ieșiri															
	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Exemplu. Să se realizeze extinderea domeniului de demultiplexare la 64 de căi.

Figura 2.47 ilustrează posibilitatea extinderii capacitatei la 64 linii de ieșire. Au fost prevăzute în acest scop patru circuite DMUX 1:2⁴ (74154) și un circuit DMUX 1:2²

(1/2 74155) pentru autorizarea funcționării circuitelor DMUX de 16 căi, în urma decodificării variabilelor de selecție. Schema a fost reprezentată în ipoteza utilizării ca DMUX (se transmit datele, 0 sau 1, de la intrarea \bar{G} pe oricare din cele 64 de căi selectate prin adresă). În situația când ea funcționează în regim de DCD, se decodifică cele 64 de stări de intrare atunci când \bar{G} este pe 0 logic.

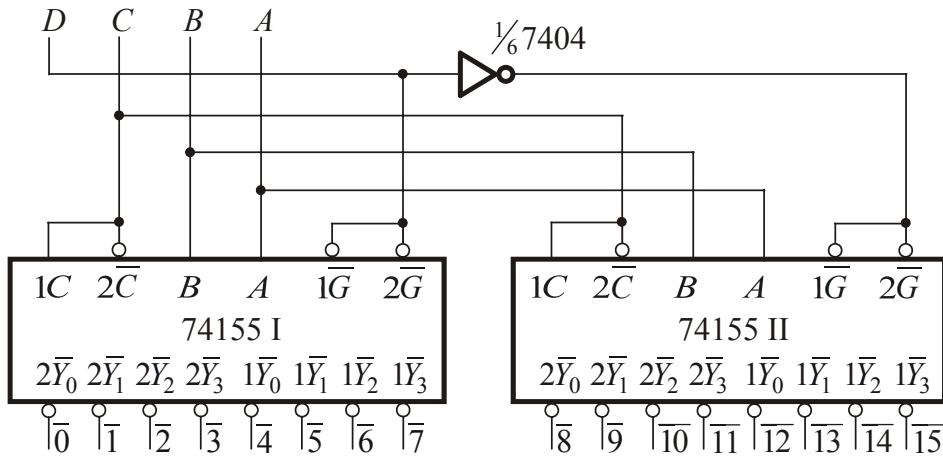


Fig.2.46 Realizarea unui DCD cu 16 căi prin conectarea în paralel a două DMUX-uri 1 : 2³.

2.3.2.2.3 Implementarea cu circuite integrate LSI

Implementarea cu circuite integrate pe scară mare, *LSI* (Large Scale Integration – acest termen este folosit pentru circuitele integrate logice cu 100 până la 1000 de porți/capsulă, respectiv cu până la 50.000 de tranzistoare/cip) utilizează la rândul ei două metode specifice, diferențiate după tipul circuitelor utilizate⁹:

- implementarea cu memorii ROM;
- implementarea cu circuite/dispozitive logice programabile (PLD).

Trebuie remarcat faptul că, deoarece memoria ROM poate fi privită ca un circuit universal de implementare cu DCD și porți logice, nu mai este necesară minimizarea matematică a funcțiilor logice. Minimizarea implementării cu memorii ROM este independentă de aceasta și se realizează după următoarele criterii:

- ocuparea cât mai completă a memoriei;
- reducerea capacitatei memoriei ROM prin codificare la intrare sau decodificare la ieșire. Este evident că reducerea ROM-ului operând în spațiul de intrare este considerabil mai mare, făcându-se cu puteri ale lui 2.

Implementarea cu structuri PLD are particularitățile sale. În acest caz se revine la minimizarea matematică a funcțiilor logice, implementarea fiind similară cu aceea a sistemelor de funcții combinaționale cu porți logice. Circuitele logice combinaționale PLD sunt de fapt structuri universale, extinse, de implementare, organizate ca arii de porți **ȘI** și **SAU** care implementează expresii sumă-de-produse.

În concluzie, trebuie subliniat că cele două metode de implementare cu circuite integrate SSI și MSI-DCD au fost generalizate, ajungându-se la structurile LSI corespondente, PLD-ul și respectiv ROM-ul.

⁹ S-a făcut această demarcăție deoarece, pentru o serie de autori (v., de exemplu, [33]), memoriile ROM nu sunt strict circuite PLD.

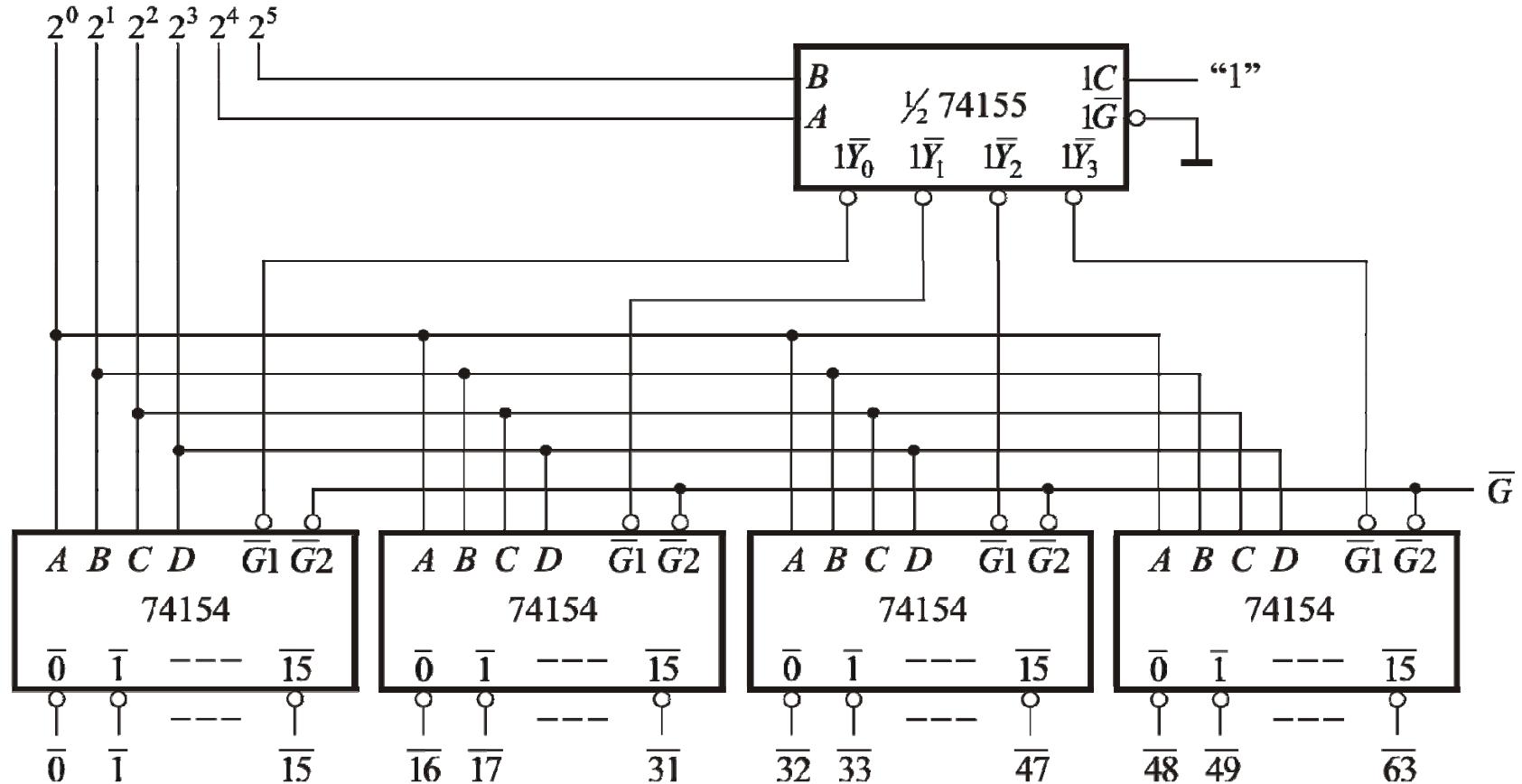


Fig.2.47 Extinderea domeniului de demultiplexare la 64 de căi.

• Memorii ROM

Se definește funcția de memorare ca fiind posibilitatea de regăsire a unor informații reprezentate sub formă binară care au fost anterior stocate. Un circuit de memorare este un circuit care implementează funcția de memorare. Memoria este deci un echipament sau subsistem aparținând unui sistem de calcul având rolul de a păstra informația. Menționăm că implementarea funcției de memorare se poate realiza în mai multe moduri, depinzând de suportul fizic folosit pentru stocarea datelor. Putem avea spre exemplu, memorii magnetice, memorii optice, memorii semiconductoare. În lucrare se au în vedere doar circuitele de memorie realizate cu dispozitive semiconductoare.

În funcție de modul de utilizare în raport cu un sistem de calcul, a acestor memorii există următoarele tipuri de funcții de memorare:

– funcția de memorare cu citire și scriere de date; în această categorie intră aşa numitele memorii cu acces aleatoriu *RAM* (Random Access Memory) care permit citirea și înscrierea unor noi date de către sistemul care le utilizează, precum și memoriile *E²PROM* respectiv *EEPROM* (Electrically Eraseable Programmable Read Only Memory)¹⁰ sau memoriile *EAROM* (Electrical Alterable ROM), care pot fi atât citite cât și șterse în mod selectiv și reprogramate de către sistemul care le utilizează;

– funcția de memorare numai cu citire de date, din această categorie făcând parte memoriile *ROM* (Read Only Memory); în funcție de modul în care informația poate fi înscrisă (și eventual ștersă) există următoarele tipuri de memorii ROM: *PROM* (Programmable Read Only Memory) respectiv *EPROM* (Erasable Programmable Read Only Memory), care pot fi numai citite de către sistemul care le utilizează; ștergerea, posibilă numai în cazul memoriilor de tip EPROM, nu este efectuată de către sistemul utilizator și nu este selectivă în raport cu informația înscrisă.

Așa cum este ușor de observat, regăsirea unei informații stocate necesită furnizarea unor semnale privind locul unde se găsește această informație. Aceste semnale constituie intrări pentru circuitul de memorie și se numesc *adrese*. Cuvintele (numerele) binare memorate constituie *date* pentru acest circuit și ele sunt semnale de intrare atunci când se înscrive în memorie și semnale de ieșire atunci când se citește din memorie. În final trebuie să se precizeze că accesul la memorie se face la un moment de timp bine determinat, moment necesar a fi comunicat printr-un semnal circuitului de memorie. Ca urmare, un circuit de memorie împreună cu conexiunile sale informaționale poate fi reprezentat ca în figura 2.48.

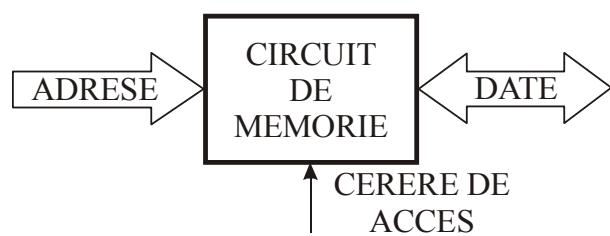


Fig.2.48 Conexiunile informaționale ale unei memorii.

Trebuie precizat că transferul de date este bidirectional (datele intră și ies din cir-

¹⁰ Circuitele E²PROM fac parte din categoria memoriilor ROM, dar funcțional pot fi privite ca memorii RAM.

cuit) în cazul memoriilor RAM și E²PROM și unidirecțional (datele ies din circuit) în cazul memoriilor ROM, PROM și EPROM.

Caracteristicile mai importante ale unei memorii sunt:

- geometria sau modul de organizare a memoriei reprezentat de lungimea unui cuvânt și numărul de cuvinte memorate;
- capacitatea memoriei, reprezentând numărul total de biți ce pot fi memorati;
- timpul de acces al memoriei, reprezentând intervalul de timp din momentul aplicării cuvântului de adresă, pe magistrala de adresare, până în momentul când cuvântul (de la acea locație) se află pe magistrala de date;
- puterea consumată; pentru caracterizarea din acest punct de vedere a unei memorii se folosește puterea consumată raportată la un bit de informație, respectiv raportul dintre puterea totală consumată de circuit și capacitatea acestuia;
- volatilitatea; o memorie este volatilă dacă informația înscrișă se pierde în timp; pierderea informației se poate datora fie modului de stocare a acesteia (memorii dinamice) fie datorită dispariției tensiunilor de alimentare ale circuitului.

În cele ce urmează se va face o scurtă prezentare a memoriilor ROM [2,15,17,18,26, 27,30].

Din punctul de vedere al tehnologiei utilizate se deosebesc memoriile ROM realizate în tehnica bipolară sau în tehnica unipolară (MOS). Performanțele circuitelor de memorie integrate poartă amprenta caracteristicilor tehnologiei respective: bipolară – viteză de procesare mare, putere disipată ridicată, densitate de integrare scăzută; MOS – densitate de integrare ridicată, viteză de procesare mai scăzută decât la bipolară, putere disipată relativ scăzută.

Programarea memoriilor ROM este procesul prin care se stabilește conținutul memoriei, în conformitate cu aplicația avută în vedere. Metodele folosite pentru programare depind de tipul de memorie.

Se consideră ca fiind o *celulă de memorie*, circuitul elementar care realizează memorarea unui bit. Aceasta se poate realiza, așa cum s-a menționat, fie cu tranzistoare bipolare fie cu tranzistoare MOS. Circuitul de memorie propriu-zis este aranjat sub forma unei matrice în care nodurile se găsesc celulele de memorie.

Memoriile ROM sunt circuite de memorie al căror conținut este programat la fabricare și nu poate fi schimbat de utilizator. Un exemplu de celulă de bază pentru o astfel de memorie este dat în figura 2.49. Ea este constituită dintr-un tranzistor cu efect de câmp a cărui tensiune de intrare diferă (fiind mai mare sau mai mică) față de cea de prag, în funcție de conținutul informațional al locației respective. Dacă la aplicarea unui impuls pozitiv pe grila tranzistorului acesta conduce (cazul unui tranzistor MOS cu canal de tip

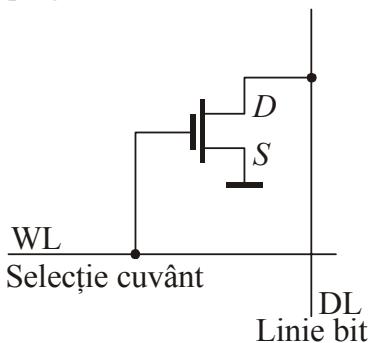


Fig.2.49 Celulă de memorie ROM cu tranzistor MOS.

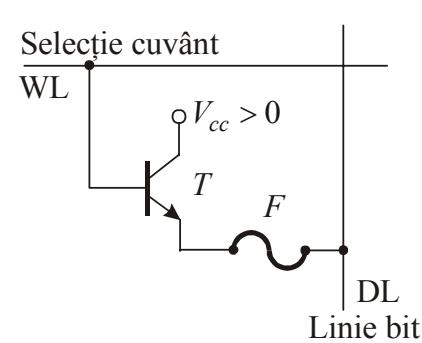


Fig.2.50 Celulă de memorie PROM.

n), atunci canalul devenit conductor al tranzistorului se comportă ca un scurtcircuit între drenă (*D*) și sursă (*S*). În acest caz informația înscrisă este 0 logic, respectiv, dacă tranzistorul este blocat aceasta este 1 logic.

Memoriile PROM sunt circuite de memorie al căror conținut este programat o singură dată de utilizator. După înscriere, informația nu mai poate fi ștearsă. Celula de memorie a unor astfel de circuite are la bază un fuzibil care este ars la programare. Celula de bază a unei memorii PROM realizată cu tranzistoare bipolare este dată în figura 2.50.

Înțial toate fuzibilele memoriei sunt scurtcircuitate. Programarea unei celule înseamnă arderea fuzibilului din nodul respectiv. Pentru programare se aplică un impuls pozitiv pe bază, iar linia de bit DL se menține la potențial coborât. Currentul de emitor al tranzistorului, suficient de mare, produce arderea fuzibilului *F*. Programarea se face succesiv pe fiecare celulă, selecția unei celule făcându-se pe liniile WL și DL.

Memoriile EPROM folosesc pentru realizarea celulei de memorie un tranzistor cu efect de câmp cu dublă poartă (grilă), una comandanță și una izolată ca în figura 2.51.

Tranzistorul MOS canal *n*, se formează între două regiuni de tip *n*. Poarta izolată se găsește în imediata apropiere a substratului *p* și a regiunilor *n⁺*. A doua poartă, situată deasupra primei, constituie în același timp și selecția pe linia respectivă, iar drena este legată pe linia de bit corespunzătoare coloanei. Dacă pe poarta izolată este acumulată sarcină electrică negativă atunci aplicarea unor tensiuni pozitive pe grila a două nu poate aduce tranzistorul în starea de conducție. Dacă pe poarta izolată nu este acumulată o sarcină, atunci aplicarea tensiunii pozitive pe grila a două ceează un câmp care duce la formarea canalului *n* și la conducția tranzistorului. Nivelul logic pe linia de bit este 1 când tranzistorul este blocat și 0 când acesta conduce. Pentru ștergerea informației din celulă și revenirea în starea neprogramată (tranzistor blocat) se expune circuitul la acțiunea radiației ultraviolete. Electronii din grilă preiau energie de la radiație și trec înapoi în substrat prin stratul izolator.

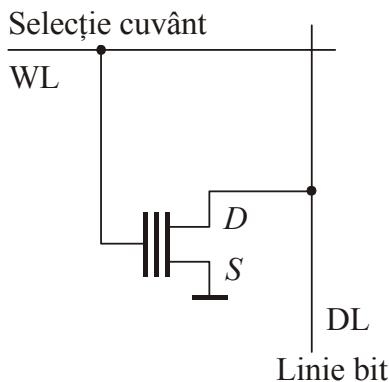


Fig.2.51 Celulă de memorie EPROM.

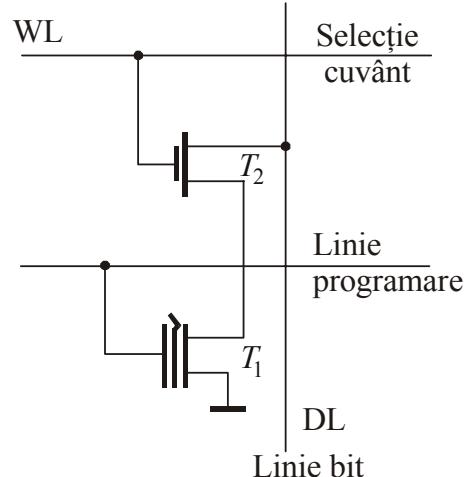


Fig.2.52 Celulă de memorie E²PROM.

Memoriile E²PROM folosesc un principiu asemănător numai că pentru trecerea electronilor prin stratul izolator utilizează efectul de tunel. Structura unei celule de memorie este prezentată în figura 2.52. Celula de memorie pentru acest tip de circuit este formată din două tranzistoare, un TEC obișnuit și un TEC-MOS bigrilă, conectate ca în figură. Într-o celulă de memorie ștearsă, grila izolată (flotantă) a tranzistorului *T*₁ este încărcată cu sarcină negativă și astfel tranzistorul *T*₁ este blocat. Ștergerea informației din

celulă se face astfel: se aplică tensiune pozitivă pe linia de selecție cuvânt, punând în conductie tranzistorul T_2 .

Înscrierea informației în celulă se face aplicând tensiuni pozitive pe linia de selecție cuvânt și pe drena tranzistorului T_2 , în timp ce linia de programare este la potențial zero. Câmpul electric format între grilă și substratul p ("+" substrat, "-" grilă) smulge electronii din grila a două, aceasta acumulează sarcină pozitivă și tranzistorul T_1 intră în conductie prin formarea canalului n între drenă și sursă.

Structura internă a unei memorii ROM (PROM, EPROM) este prezentată în figura 2.53 [2,26,30]. Structural, un circuit ROM se prezintă sub forma unei matrice, în fiecare nod al acesteia (intersecția unei linii cu o coloană) fiind memorat un bit. Liniile matricei, $0 \div (2^n - 1)$ sunt activate de către cele 2^n ieșiri ale unui decodificator, iar intrările acestuia sunt conectate la cei n biți ai magistralei de adresare, A_{n-1}, \dots, A_1, A_0 . Coloanele matricei sunt conectate la magistrala de date prin intermediul unor amplificatoare inversoare (buffere) tip TSL¹¹ a căror comandă se obține de la magistrala de control a sistemului. Circuitul de ieșire trebuie să asigure și posibilitatea programării memoriei, în cazul circuitelor PROM și EPROM.

Din punct de vedere logic memoria ROM poate fi privită ca o structură formată din două nivele de porți: řI-uri (decodificatorul) și SAU-NU-uri (matricea de memorie). DCD-ul primește codurile de intrare în binar, (fie n numărul intrărilor) și activează pentru fiecare cod câte o ieșire din cele 2^n . Această ieșire se conectează sau nu la circuitul SAU-NU, realizându-se astfel memorarea lui 0, respectiv 1. De exemplu, când se activează ieșirea 0 a DCD-ului va apărea la $\overline{O}_0 \div \overline{O}_{m-1}$, vectorul 00...10.

În concluzie, se poate aprecia că cele n linii de adresă sunt decodificate în 2^n linii de cuvânt iar matricea de memorie furnizează la ieșire 2^n cuvinte de câte n biți fiecare.

Fie, de exemplu:

n – numărul de biți ai vectorului de intrare (adresa);

c – numărul de cuvinte memorate în ROM;

m – numărul de biți din fiecare cuvânt.

În general, informația este stocată sub forma unui cuvânt cu lungimea de m biți, astfel ocupând o linie întreagă a matricei. Numărul de adrese, deci și numărul de cuvinte memorate, deoarece liniile matricei sunt egale cu ieșirile DCD-ului, este întotdeauna o putere a lui 2 ($c = 2^n$). Modul de organizare al unei memorii ROM este specificat prin produsul $c \times m$ (de exemplu, 256×4 , 512×8 , 1024×8 , etc.).

Capacitatea memorie ROM se exprimă, așa cum s-a precizat, prin numărul total de biți memorati. Unitatea de măsură pentru aceștia este kilobitul ($1K = 1024$ biți). De exemplu: $1K \times 1$ (are capacitatea $= 1024$ biți); $4K \times 1$ ($= 4096$ biți); $1K \times 8$ notat și $1KB$ ($= 8192$ biți, un kilobait), etc.

Pentru ca circuitul de memorie să poată fi selectat, în cazul în care modulul de memorie este format din mai multe astfel de circuite, este necesară și o bornă de selectare circuit, \overline{CS} (Chip Select). Semnalul de control \overline{CS} face selecția circuitului controlând și

¹¹ Pentru a se putea conecta mai simplu la magistrale, majoritatea circuitelor logice integrate au trei stări la ieșire (TSL – Three State Logic). De fapt, circuitele TSL prezintă tot două stări logice 1 și 0, dar, în plus, există și o stare de înaltă impedanță, HZ. Pentru autorizarea/validarea stării HZ se folosește un semnal aplicat pe o bornă notată prin simbolul OE (Output Enable), adică ieșire autorizată/validată (v. Anexa B).

starea ieșirilor (HZ sau conectate la magistrala de date) în timp ce semnalul \overline{OE} controlă numai starea circuitelor de ieșire. Pentru $\overline{OE} = 1$, ieșirea este în starea HZ . Același lucru se întâmplă și dacă $\overline{CS} = 1$, indiferent de valoarea lui \overline{OE} . Aceste două funcțiuni, \overline{CS} și \overline{OE} sunt luate împreună, pentru a se micșora numărul de pini la cip și scoase la un singur pin cu simbolul $\overline{CS}/\overline{OE}$ (v. fig. 2.53,b). În acest caz, însă, acest semnal unic, pentru $\overline{CS}/\overline{OE}$ trebuie să se obțină printr-o logică exterioară cipului.

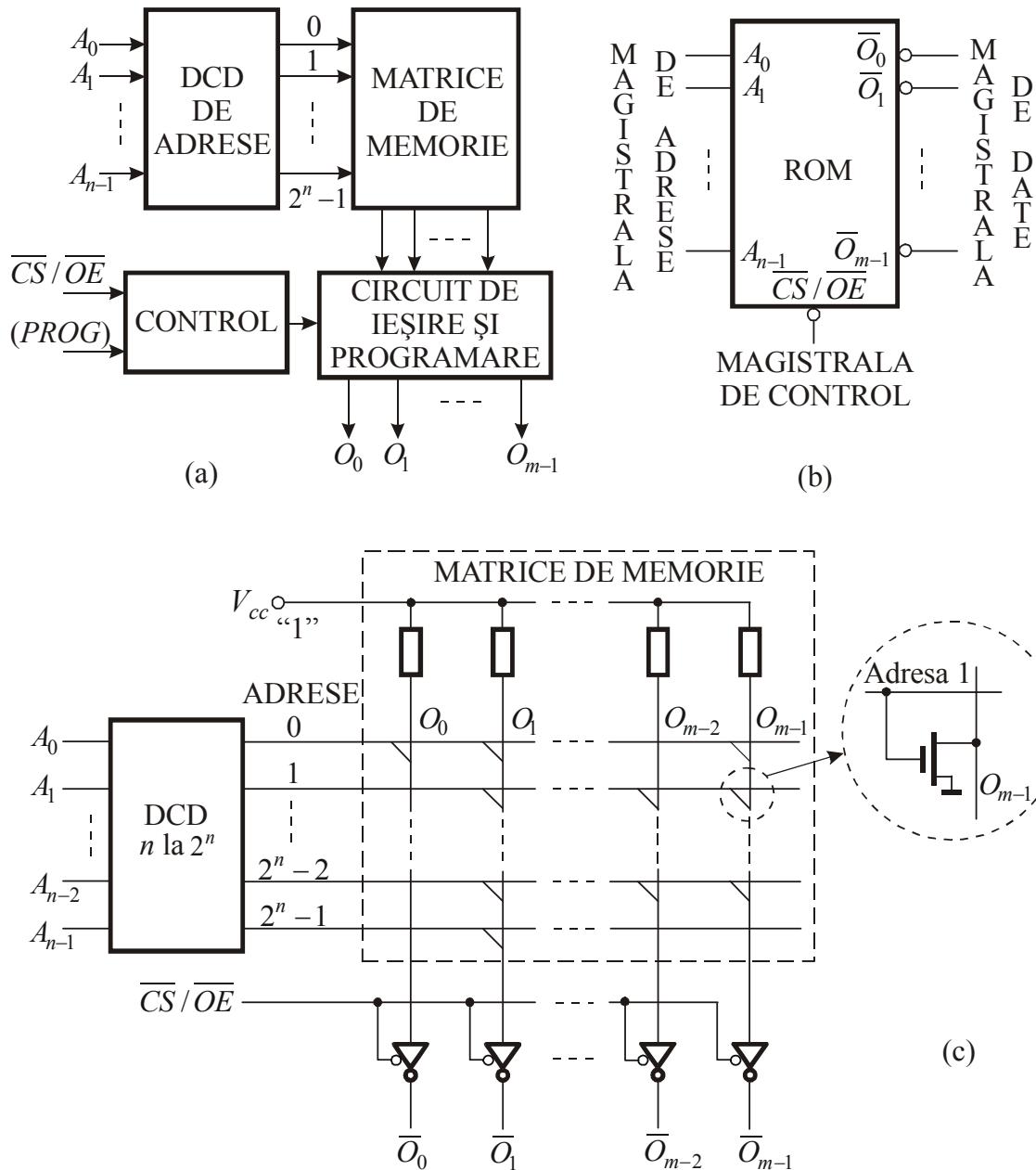


Fig.2.53 Memorie ROM: (a), (b) scheme-bloc; (c) schema de principiu.

Pentru funcționarea corectă a circuitelor de memorie semnalele aplicate la bornele trebuie să satisfacă anumite restricții de timp. În figura 2.54 sunt prezentate diagramele de

semnal pentru un ciclu de citire tip al unei memorii ROM. Semnificația notațiilor este următoarea:

- t_{RC} – durata ciclului de citire;
- t_A – timpul de acces la memorie;
- t_{CS} – intervalul de timp în raport cu \overline{CS} după care apar date stabile la ieșire;
- t_{OE} – intervalul de timp după care apar date stabile la ieșire în raport cu impulsul \overline{OE} ;
- t_{OH} – timpul de menținere a datelor la ieșire după dispariția impulsului \overline{OE} .

Forma impulsurilor și duratele lor pentru programarea unei memorii (PROM, EPROM) sunt prezentate în fișele de catalog ale circuitelor respective.

• Implementarea funcțiilor booleene cu memorii ROM

Memoriile ROM se pot utiliza la implementarea CLC-urilor cu un număr mare de intrări și ieșiri, care altfel s-ar realiza cu foarte multe circuite integrate SSI sau MSI [15,27,30].

Această aplicație este posibilă observând că structura internă a memoriei ROM constituie o generalizare a formei de implementare cu DCD-uri și porți logice: se decodifică pe nivelul de SI-uri toți termenii canonici (fiecare linie de cuvânt de la intrarea matricei reprezintă de fapt un termen canonic format din variabilele de intrare) iar cei care intervin în expresia fiecărei funcții, $\overline{O}_0 \div \overline{O}_{m-1}$, sunt adunați prin nivelul de SAU-NU-uri (nivelul SAU se obține deci, prin programarea nodurilor de pe o coloană a matricei). Deci lista de cuvinte a memoriei ROM reprezintă chiar tabelul de adevăr al CLC-ului. În concluzie, fiecare coloană a memoriei ROM generează valorile unei funcții de variabile de intrare (adresare) pentru acei termeni canonici care au fost selectați prin programarea nivelului SAU.

La implementarea cu ROM-uri nu este necesară minimizarea, deoarece sunt memorați în DCD toți termenii canonici (mintermi) care se pot forma cu variabilele aplicate pe intrările de adresă și sunt incluse toate posibilitățile de apariție a acestora în funcțiile de ieșire, prin programarea matricei de memorie.

Programarea nodurilor din matricea de memorie se face în urma inspecției tabelului de adevăr al funcției; se programează acele noduri care introduc în compunerea funcției termenii canonici ce au valoarea 1 logic. S-a considerat că un nod neprogramat este zero în matricea SAU, adică nu introduce termenul canonic corespunzător liniei la intersecția căreia se află.

Datorită faptului că pot memora tabele de adevăr foarte mari, ROM-urile se utilizează în micropogramare, conversii de cod, generarea de caractere, implementarea CLC-urilor cu un număr mare de variabile de intrare și cu mai multe ieșiri, etc.

Implementarea cu memorii ROM a CLC-urilor, presupune parcurgerea următoarelor etape:

- stabilirea dimensiunii memoriei necesare pentru aplicația respectivă;
- alegerea tipurilor de circuite ROM integrate cele mai potrivite: cu dimensiuni iden-

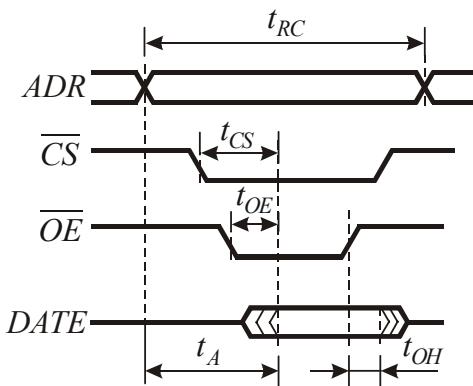


Fig.2.54 Diagramă de semnal pentru citirea ROM-ului.

tice sau cât mai apropiate de cele stabilite anterior;

– dacă nu există memorii ROM cu dimensiuni identice cu cele dorite, se fac transformări de dimensiuni cu diverse metode;

– stabilirea tabelului de adevăr al memoriei ROM;

– reducerea dimensiunii memoriei ROM, atunci când este posibil, utilizând codificarea la intrare sau decodificarea la ieșire.

Implementarea cu memorii ROM se face având în vedere următoarele două aspecte:

– utilizarea unui număr minim de circuite integrate;

– folosirea integrală a capacitatei memoriei.

Uneori aceste deziderate pot fi atinse simultan, alteori trebuie realizat un compromis în favoarea uneia dintre ele.

Exemplu. Să se implementeze cu memorie ROM un CLC care realizează funcțiile:

$$\begin{aligned}
 f_1^{\text{FCD}} &= P_0 + P_2 + P_4 + P_6 + P_{20} + P_{22} + P_{26}, \\
 f_2^{\text{FCD}} &= P_{13} + P_{14} + P_{15} + P_{30}, \\
 f_3^{\text{FCD}} &= P_1 + P_5 + P_{11} + P_{19} + P_{20}, \\
 f_4^{\text{FCD}} &= P_2 + P_7 + P_9 + P_{17} + P_{23} + P_{28}, \\
 f_5^{\text{FCD}} &= P_{25} + P_{26} + P_{27} + P_{28} + P_{29}, \\
 f_6^{\text{FCD}} &= P_0 + P_5 + P_{10} + P_{15} + P_{20} + P_{25} + P_{28}, \\
 f_7^{\text{FCD}} &= P_0 + P_3 + P_6 + P_9 + P_{12} + P_{15} + P_{18} + P_{21} + P_{24} + P_{27} + P_{29}, \\
 f_8^{\text{FCD}} &= P_4 + P_8 + P_{12} + P_{16} + P_{20} + P_{24} + P_{26} + P_{29}.
 \end{aligned} \tag{2.133}$$

Se observă că termenul canonic de rang cel mai mare este P_{30} , deci se poate presupune că funcțiile sunt de 5 variabile. În aceste condiții sunt necesare 5 intrări în memoria ROM, pentru a se obține din DCD 32 de termeni canonici. Trebuie implementate 8 funcții; înseamnă că memoria ROM necesară trebuie să aibă organizarea internă 32×8 biți. Există astfel de circuite integrate și este necesar unul singur pentru a rezolva problema (de exemplu, TBP18S030 [34]).

Tabelul de adevăr al memoriei ROM se stabilește pornind de la formele canonice disjunctive ale funcțiilor negate (v. tab. 2.23).

Tab.2.23 Tabelul de adevăr al memoriei ROM din exemplu.

	x_1	x_2	x_3	x_4	x_5	\bar{f}_1	\bar{f}_2	\bar{f}_3	\bar{f}_4	\bar{f}_5	\bar{f}_6	\bar{f}_7	\bar{f}_8
0	0	0	0	0	0	0	1	1	1	1	0	0	1
1	0	0	0	0	1	1	1	0	1	1	1	1	1
2	0	0	0	1	0	0	1	1	0	1	1	1	1
3	0	0	0	1	1	1	1	1	1	1	1	0	1
...
30	1	1	1	1	0	1	0	1	1	1	1	1	1
31	1	1	1	1	1	1	1	1	1	1	1	1	1

Implementarea CLC-ului este prezentată în figura 2.55, în care memoria ROM are

programată în interior acest tabel de adevăr. Deoarece ieșirile memoriei sunt active pe nivel 0 logic, se obțin în exterior funcțiile:

$$\bar{f}_k = f_k, \quad \text{unde } k=1 \div 8. \quad (2.134)$$

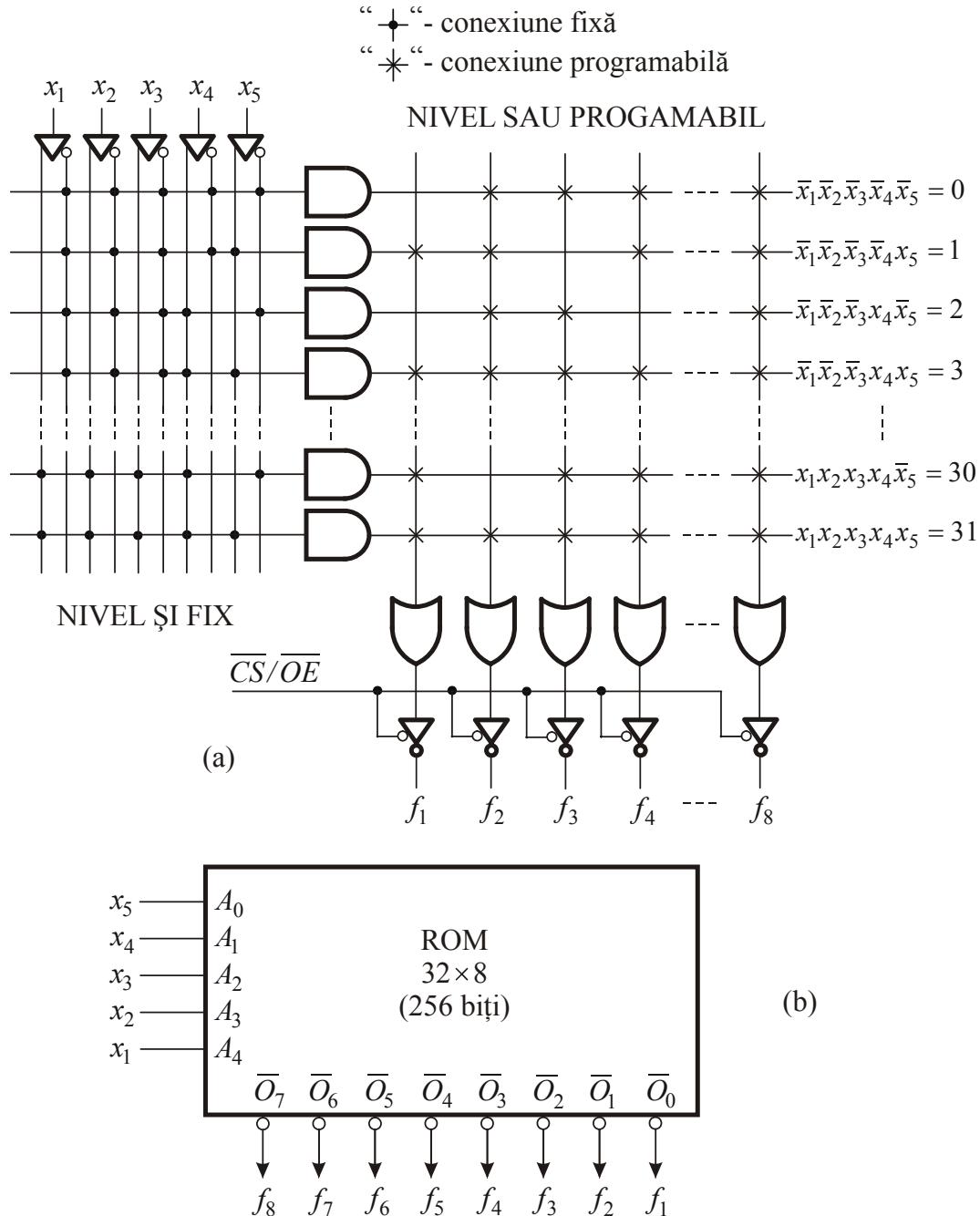


Fig.2.55 Memoria ROM utilizată pentru realizarea CLC-ului din exemplu.

Modificarea geometriei unei memorii presupune realizarea de circuite de memorie având un număr de cuvinte sau/și un număr de biți pe cuvânt diferit de cel al circuitelor disponibile.

Dată fiind capacitatea unei capsule ROM, aşa cum s-a mai menționat, prin produsul

dintre numărul de cuvinte și numărul de biți ai cuvântului de ieșire, există posibilitatea de a realiza o memorie de capacitate mai mare prin utilizarea unui număr sporit de capsule. Extinderea capacitatii se poate obține acționând asupra numărului de cuvinte, lăsând numărul de biți pe cuvant neschimbat (extindere la intrare sau de adresă), modificând numărul biților de ieșire la un număr neschimbat de cuvinte (extindere la ieșire), respectiv, atât prin modificarea numărului de cuvinte cât și a numărului de biți ai cuvântului de ieșire (extindere mixtă).

Exemplu. Să se realizeze o memorie ROM cu organizarea internă de 256×4 biți, la care matricea de memorie este de 32×32 biți.

În figura 2.56 se prezintă organizarea tipică a unei memorii ROM având capacitatea de 1K (v., de exemplu, TBP24S10 [34]). Se constată că selecția unui cuvant de ieșire de 4 biți ($\bar{O}_0, \dots, \bar{O}_3$), din cele 256, nu se face cu ajutorul unui DCD cu 256 de linii de ieșire ci sunt utilizate: un DCD cu 32 linii și 4 MUX-uri de 8 căi. La intrările DCD-ului se vor aplica primii 5 cei mai semnificativi biți ai cuvântului de adresă, în timp ce pe intrările de selecție ale MUX-urilor, ceilalți 3 biți mai puțin semnificativi.

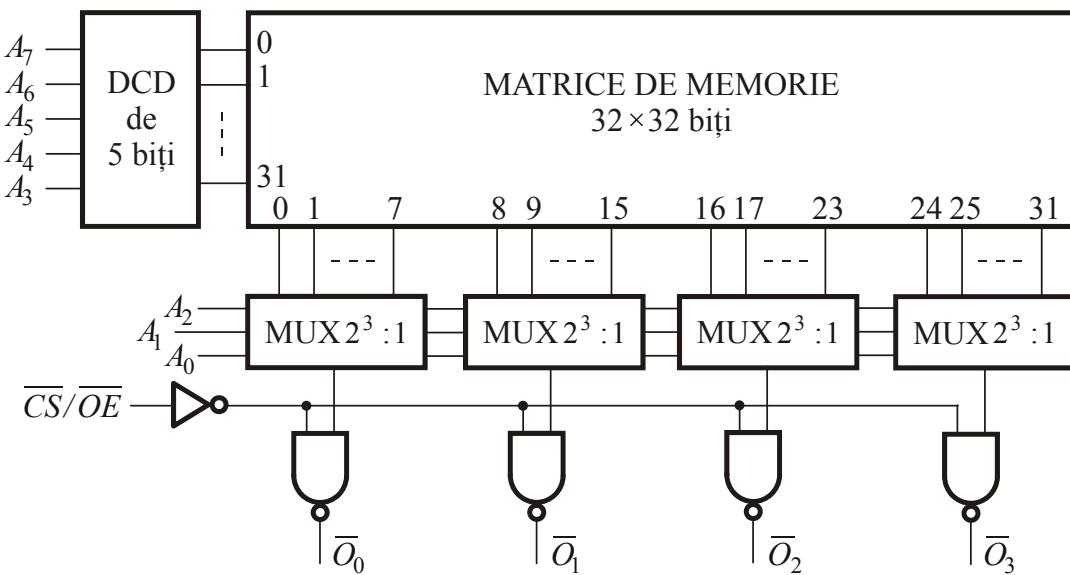


Fig.2.56 Realizarea unei memorii ROM de 256×4 (1K).

Exemplu. Cu ajutorul unor circuite integrate ROM de 8 Kbiți, cu organizarea internă de 1024×8 biți, să se realizeze o memorie de $4K \times 8$ biți (extinderea numărului de cuvinte).

În general, pentru extinderea numărului de cuvinte al unei memorii ROM se procedă în felul următor:

- se calculează numărul circuitelor integrate necesare;
- se conectează în paralel adresele și ieșirile circuitelor;
- biții suplimentari de adresă se decodifică și selectează circuitele pe intrările \overline{CS} .

Memoria de $4K \times 8$ biți are de 4 ori mai multe cuvinte decât circuitul integrat. Deci vor fi necesare 4 circuite integrate pentru a o implementa (se pot utiliza, de exemplu, circuitele TBP28S86A [34]).

Extinderea numărului de cuvinte ale memoriei se face conectând cele 4 circuite integrate ca în figura 2.57. Fiecare circuit integrat se adresează cu 10 biți ($A_9 \div A_0$),

conectați în paralel la toate ROM-urile. Pentru adresarea memoriei de $4K \times 8$ biți mai sunt necesari încă 2 biți, A_{11} și A_{10} , care, decodificați (se poate folosi 1/2 74139 [34]), activează doar câte unul dintre circuitele integrate ROM. Astfel, la ieșirile cuplate în paralel se citește pentru $A_{11}A_{10} = 00$, conținutul ROM₁, pentru $A_{11}A_{10} = 01$, conținutul ROM₂, etc., celelalte trei circuite integrate fiind în acest timp inhibate.

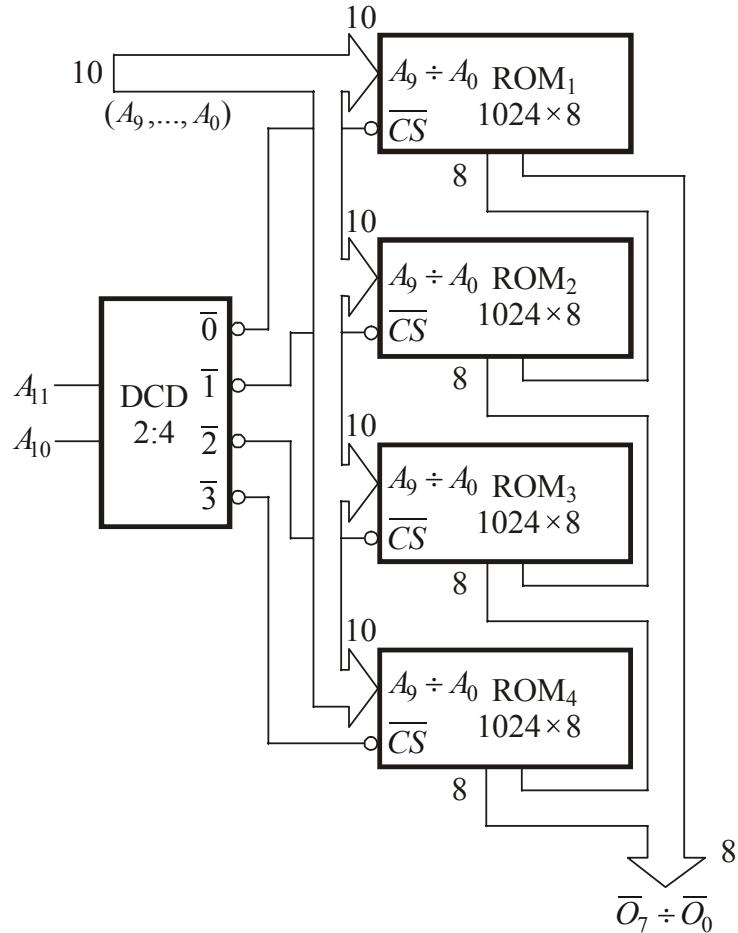


Fig.2.57 Extinderea numărului de cuvinte.

Exemplu. Cu ajutorul unor circuite integrate de memorie ROM de 8 Kbiți (1024 cuvinte $\times 8$ biți, de exemplu, TBP28S86A [34]) să se realizeze o extindere la $2K \times 16$ biți (extinderea numărului de cuvinte și a numărului de biți pe cuvânt).

Extinderea numărului de cuvinte se face cu ajutorul intrării \overline{CS} , activând pentru $A_{10} = 0$, ROM₁ și ROM₂ și pentru $A_{10} = 1$ celelalte două memorii (v. fig. 2.58). În acest caz, DCD-ul (v. fig. 2.57) s-a redus la un inversor.

Extinderea lungimii cuvântului se face prin adăugarea ROM₃ și ROM₄ conectate în paralel cu ROM₁, respectiv ROM₂, pe adrese și \overline{CS} . Ieșirile ROM₃ și ROM₄ sunt conectate în paralel cu cele ale ROM₁ respectiv ROM₂ pentru a se obține lungimea cuvântului de 16 biți.

Exemplu. Să se realizeze o memorie cu organizarea internă de 64 cuvinte $\times 4$ biți cu circuite ROM de 32×8 biți (extinderea numărului de cuvinte și reducerea numărului de biți pe cuvânt, capacitatea de memorare rămânând aceeași).

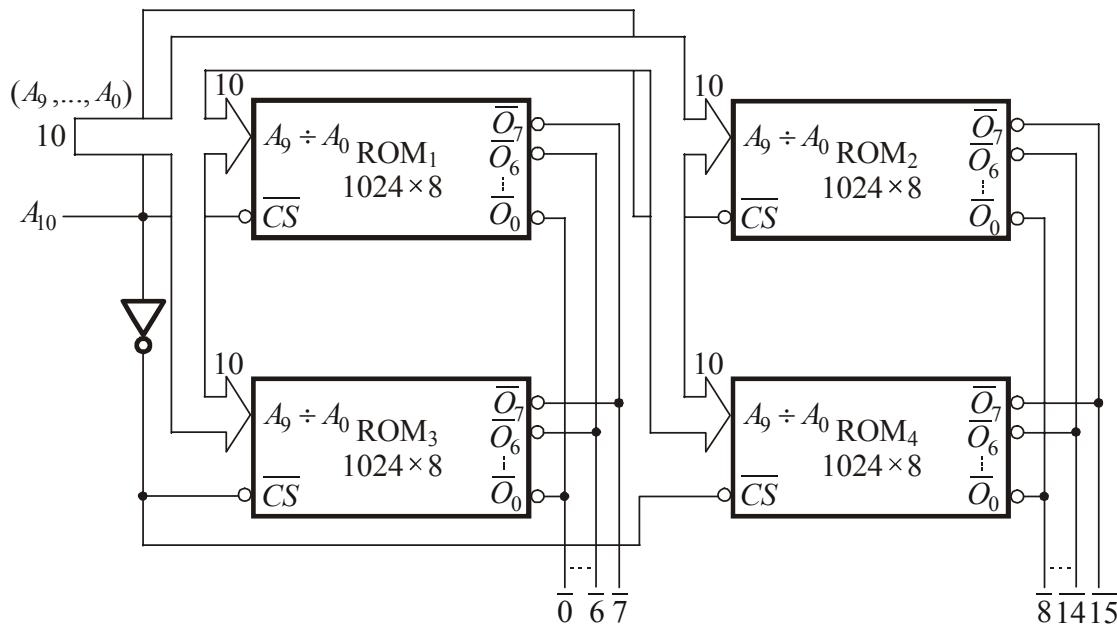


Fig.2.58 Extinderea numărului de cuvinte și a numărului de biți pe cuvânt.

Se poate reduce lățimea cuvântului concomitent cu mărirea numărului de adrese prin diverse procedee dintre care unul este prezentat în figura 2.59. Se folosește circuitul integrat 74157 [34] care conține 4 MUX-uri neinversoare de câte 2 căi ($4 \times \text{MUX } 2:1$), conectat, prin intermediul intrărilor de date ale celor 4 MUX-uri, la ieșirile circuitului ROM (32×8) TBP18S030 [34]. Tabelul de adevăr al unui MUX 2:1 este prezentat în tabelul 2.24. Cu ajutorul circuitului integrat 74157, pentru $A_5 = 0$ se transferă la ieșire $\overline{O}_0 \div \overline{O}_3$, iar pentru $A_5 = 1$, $\overline{O}'_0 \div \overline{O}'_3$.

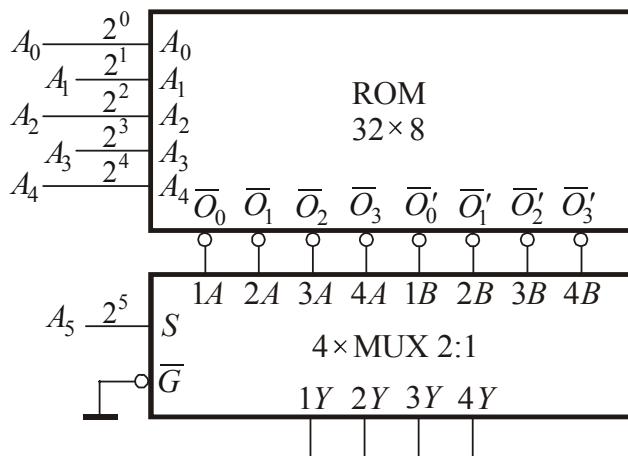


Fig.2.59 Extinderea numărului de cuvinte și reducerea numărului de biți pe cuvânt.

Tab.2.24 Tabelul de adevăr pentru un MUX 2:1 (1/4 74157).

\overline{G}	S	A	B	Y
1	*	*	*	0
0	0	0	*	0
0	0	1	*	1
0	1	*	0	0
0	1	*	1	1

Pentru reducerea numărului de intrări într-o memorie ROM se utilizează adeseori multiplexoare. Dacă, spre exemplu, există mai multe moduri de operare și fiecărui mod îi corespunde o anumită configurație de

intrare, se pot utiliza circuite de selecție (multiplexoare) pentru biții corespunzători configurației de interes în funcție de mod. Dacă modurile de operare nu sunt codificate într-o modalitate direct utilizabilă se poate folosi câteodată, pentru generarea adreselor multiplexorului, chiar o memorie ROM suplimentară. Uneori se pot folosi chiar ieșirile memoriei ROM pentru selecția modului, dar numai într-o schemă secvențială.

Exemplu. Un CLC primește pe intrări numere de 8 biți, unele în cod Gray, altele în BCD Exces 3 și trebuie să furnizeze la ieșire numai numere în cod binar. Să se implementeze cu memorie ROM acest CLC, știind că nu pot apărea simultan mai multe numere pe intrări.

Organizarea memoriei ROM este prezentată în figura 2.60,a. Pentru implementare sunt necesare, de exemplu, 16 circuite ROM de $8K \times 8$ biți și un DCD.

Deoarece nu pot apărea simultan pe intrări numere din cele 2 grupe, se poate reduce dimensiunea memoriei ROM cu schema din figura 2.60,b. Bitul cuplat pe A_8 , care indică în ce cod este numărul sosit pe intrări, comandă trecerea acestuia prin MUX către memorie. În acest fel, prin adăugarea doar a 2 MUX-uri (de fapt se folosesc 2×74157), s-a redus foarte mult capacitatea memoriei ROM necesare: de $2^8 = 256$ ori (se poate utiliza circuitul TBP28S42 (512×8), [34]).

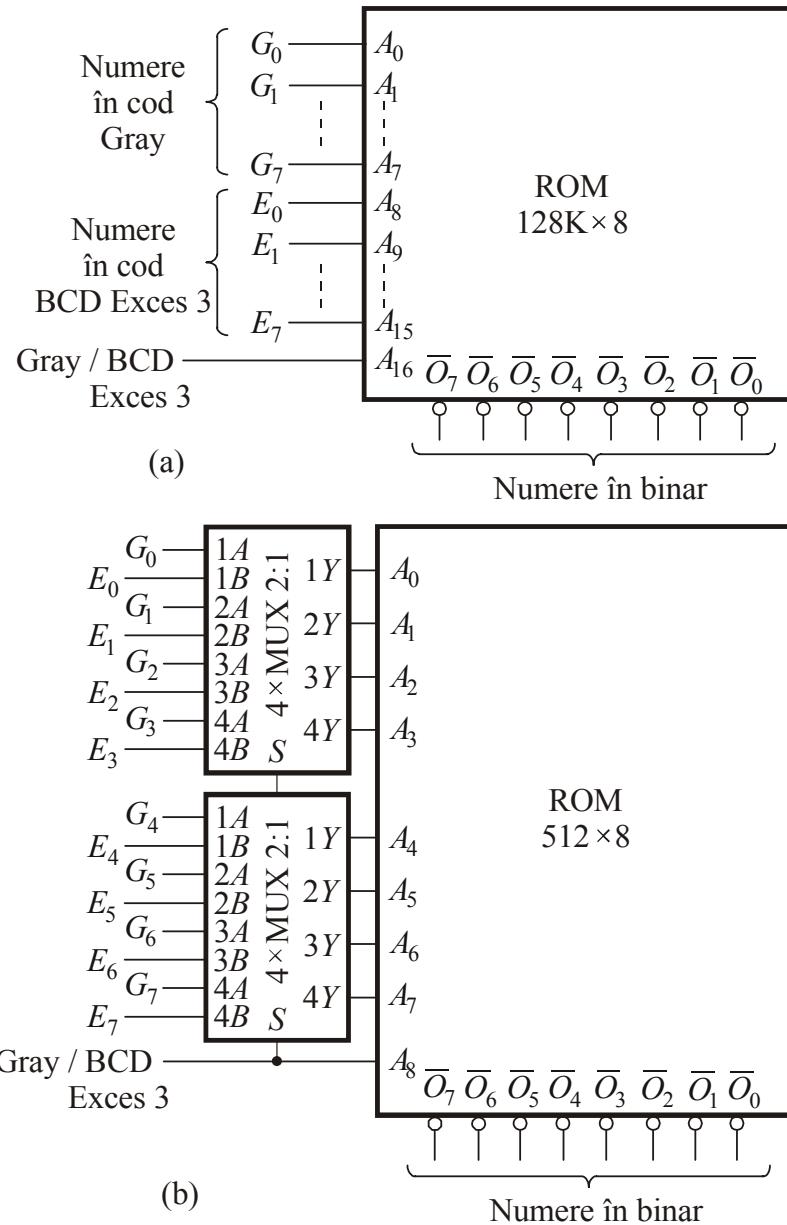


Fig.2.60 Reducerea dimensiunii memoriei ROM prin micșorarea numărului de intrări.

Exemplu. Utilizându-se circuite MUX să se reducă dimensiunea unei memorii ROM de la $4K \times 8$ biți la 64×8 biți.

Schema de principiu este prezentată în figura 2.61. Adresele $A_2 \div A_5$ codificate comandă multiplexarea căilor $I_0 \div I_7$ pe intrările de adresă A_0 și A_1 (se poate folosi circuitul integrat 74153 [34]). Montajul permite reducerea dimensiunii ROM-ului prin cuplarea MUX-ului la intrare, de 64 de ori. Memoria ROM 64×8 se poate realiza prin cuplarea în paralel (v. fig 2.57) a două circuite ROM 32×8 (de exemplu, TBP18S030[34]).

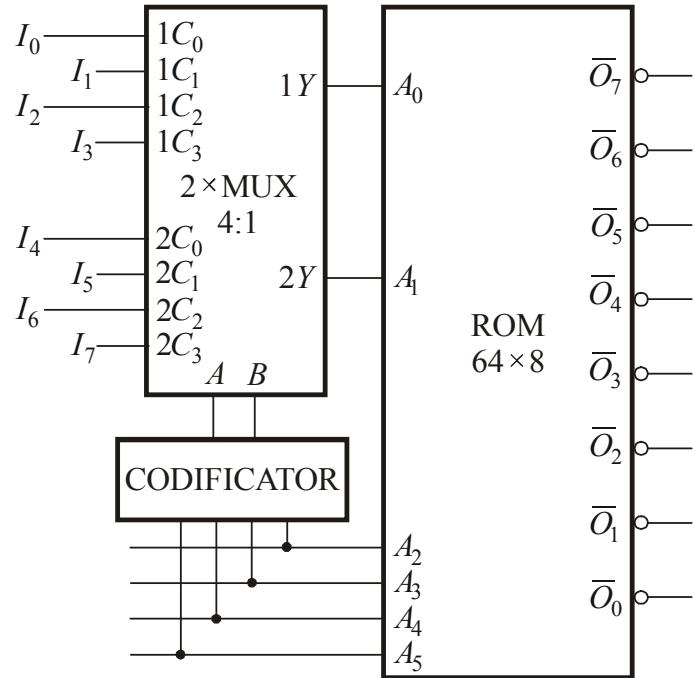


Fig.2.61 Reducerea dimensiunii memoriei ROM prin micșorarea numărului de intrări.

Reducerea dimensiunilor memoriei ROM se poate realiza, aşa cum s-a mai menționat, și prin decodificarea ieșirilor.

Exemplu. Să se implementeze cu memorie ROM un convertor de cod din binar (8 biți) în zecimal.

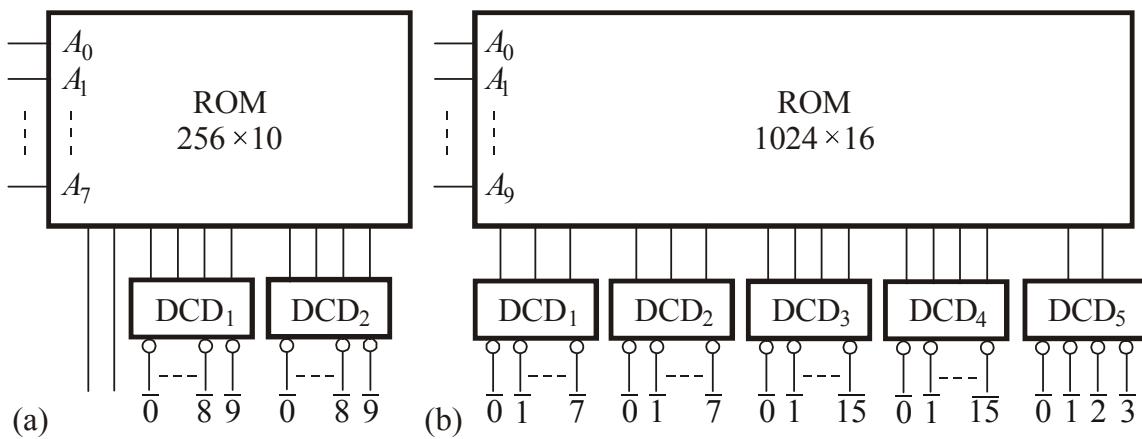


Fig.2.62 Reducerea dimensiunii memoriei ROM prin decodificarea la ieșire.

ROM-ul are 8 intrări și 22 de ieșiri (10 pentru unități, 10 pentru zeci și 2 pentru sute) deci

256×22 biți și se poate implementa cu 3 ROM-uri de 256×8 biți.

Cu schema din figura 2.62,a se realizează cu ROM conversia din binar în BCD și apoi cu ajutorul DCD-urilor 7442 [34], conversia BCD-zecimal. Se reduce astfel capacitatea ROM-ului la 256×10 biți, deci de 2,2 ori. Implementarea se poate face cu 3 ROM-uri de 256×4 biți sau 2 ROM-uri de 256×8 biți (se pot utiliza, de exemplu, circuitele integrate TBP24S10, respectiv TBP18S22, [34]).

Exemplu. Cu cât s-a redus dimensiunea ROM-ului în schema din figura 2.62,b, datorită câmpurilor codificate ce apar la ieșire?

Reducerea este de $52/16 = 3,25$ ori.

• Dispozitive logice programabile

Utilizarea tot mai largă a microprogramării în tehnica digitală a fost susținută de disponibilitatea crescândă a unor memorii fixe programabile la utilizator (PROM, EPROM). Există însă o serie de aplicații unde se operează cu un număr mare de variabile de intrare la viteze de funcționare ridicate, în care rigiditatea posibilităților de adresare ale acestor memorii le face improprii și neconomice¹². Pentru acoperirea acestor domenii de utilizare a fost realizată o componentă logică flexibilă numită *dispozitiv logic programabil, PLD (Programmable Logic Device)*. Un PLD are o structură internă care poate fi configurată cu diferite circuite combinatoriale și adesea cu circuite secvențiale. PLD-urile sunt folosite și pentru a reduce substanțial numărul componentelor unui sistem logic complex.

Dispozitivele logice programabile conțin, aşa cum s-a precizat, porți și, în unele cazuri, circuite bistabile, aranjate în aşa fel încât interconectările între componente să poată fi modificate pentru a se implementa diverse funcții logice. Circuitele logice combinatoriale PLD, cele care conțin numai porți logice de bază, sunt, de obicei, organizate ca arii de porți SI și SAU, care implementează expresii sumă-de-produse. PLD-urile secvențiale adaugă bistabili la ieșiri.

Oricare ar fi aranjamentul logic, o cerință generală a PLD-urilor este de a avea o modalitate de schimbare a interconectărilor pentru a forma o configurație logică diferită. Calea originală, definitivă, dar selectabilă de a face acest lucru a fost fabricarea dispozitivelor cu semiconductoare fuzibile (siguranțe fuzibile integrate). Inițial, aceste siguranțe sunt intacte în dispozitiv, ceea ce oferă interconectări extinse. Ulterior siguranțele sunt "arse" selectiv de către utilizator pentru a se obține interconectări dorite, folosind o unitate specială de *programare pentru PLD*. Uzual, programatorul de PLD este atașat unui mic calculator și se oferă software-ul pentru a se traduce funcția logică dorită în combinația internă de interconectări corespunzătoare. Desigur, o dată ce o siguranță este arsă, conexiunea nu se mai poate reface. PLD-urile pe care le poate programa utilizatorul, ca acestea cu semiconductoare fuzibile, se numesc "*arii programabile*".

PLD-urile cu semiconductoare fuzibile au fost dezvoltate în anii '70 iar acum există versiuni de PLD-uri arii programabile, care folosesc tehnologia memorilor semiconductoare *erasable read-only*. Aici, conexiunile depind de informațiile binare stocate. O celulă de memorie există la fiecare punct de conectare pentru a se stoca (să zicem) un 0, pentru menținerea conexiunii sau un 1, pentru anularea ei. PLD-urile bazate pe tehnologia memorilor oferă capacitatea de a modifica rapid interconectările și de mai multe ori.

¹² Așa cum s-a mai precizat, dezavantajul implementării pe un circuit ROM decurge din existența nivelului SI neprogramabil, deci generarea tuturor termenilor canonici produs de n variabile, indiferent dacă aceștia sunt necesari sau nu.

De asemenea, producătorii realizează versiuni de PLD-uri cu conexiuni permanente în funcție de specificațiile clientilor. Pentru a se realiza conexiunile fixe, trebuie realizate măști specifice circuitelor integrate la fabricarea dispozitivului. Acest proces este costisitor pentru fabricarea unui dispozitiv, dar costul scade semnificativ când este vorba de fabricarea unui număr mare de dispozitive identice.

Toate cele trei tipuri de PLD-uri (folosind siguranțe semiconductoare, tehnologia memoriilor sau conexiunile din fabricație) au produs capsule cu pini compatibili, astfel încât, într-un sistem, se poate înlocui un tip cu un altul. Un tip reprogramabil se poate folosi inițial pentru experimentări, deși acestea sunt cele mai lente în operațiile logice și cele mai scumpe; apoi, poate, se va folosi un număr mic de dispozitive (matrice) programabile, pentru o producție mică, pilot; și în sfârșit, fabricarea de PLD-uri cu conexiuni specifice poate fi folosită pentru producții mari.

În cele ce urmează, se va discuta despre PLD-uri în termeni de “conexiuni programabile”, care se aplică tuturor PLD-urilor, indiferent dacă folosesc siguranțe semiconductoare, tehnologia memoriilor sau conexiuni din fabricație. Structurile logice ale diferitelor implementări sunt similare. În acest paragraf se vor analiza doar circuitele combinatoriale PLD.

Principalele familii de circuite combinatoriale PLD sunt:

- matricele logice programabile, *PLA*-urile (*Programmable Logic Array*);
- matricele programabile logic, *PAL*-urile (*Programmable Array Logic*);
- matricele de porți programabile, *PGA*-urile (*Programmable Gates Array*).

Litera *F* (de la “field” sau “fuse”) plasată în fața denumirilor abreviate ale acestor circuite (devenind astfel *FPLA*, *FPAL*, *FPGA*) indică faptul că acestea sunt programate de către utilizator. În continuare, se va face o scurtă prezentare a acestor tipuri de circuite logice programabile.

PLA-ul este un CLC cu *două nivele de logică programabilă*: o matrice de porți *SI* și o matrice de porți *SAU*, precum și cu amplificatoare de ieșire programabile [2,18,23,27, 30,33]. Matricea *SI* constituie un decodificator condiționat, de dimensiuni reduse față de numărul mare de intrări ale circuitului, iar matricea *SAU* are rol de codificator.

Spre deosebire de memoriile ROM, la care matricea de *SI*-uri are o configurație fixă (este un decodificator al tuturor cuvintelor de intrare), fiind programabil doar nivelul de *SAU*-uri, la *PLA*-uri este programabil și primul nivel logic. Într-o altă interpretare se poate spune că nivelul de *SI*-uri la ROM furnizează toți termenii canonici posibili de construit cu variabilele de intrare, iar prin nivelul de *SAU*-uri programabil, se selectează acei termeni care intervin în expresia fiecărei funcții. La *PLA*, fiind programabil și nivelul de *SI*-uri, este deci posibilă și necesară o implementare pornind nu de la termenii canonici, ci de la cei normali, constituenți ai formei minimize.

O structură tipică de *FPLA* este prezentată în figura 2.63 (este vorba de schema de principiu a circuitului *FPLA PLS153* ($18 \times 32 \times 10$), produs de firma Philips [36]). Schema logică are 18 intrări (A_0, \dots, A_{17}), matricea *SI* conține 32 de porți cu câte 36 de intrări iar matricea *SAU* este prevăzută cu 10 porți cu câte 32 de intrări. Numărul de ieșiri este egal cu 10 (O_0, \dots, O_9). De asemenea, în figura 2.63 sunt prezentate și posibilitățile de modificare ale conexiunii dintr-un nod al matricei *SI*. Există deci patru situații pentru fiecare variabilă de intrare, așa cum se arată în figura menționată:

- starea inițială neprogramată, cu toate conexiunile intacte, cu toate variabilele și inversele lor conectate;

- pot fi selectate fie variabilele, fie negatele lor, prin eliminarea câte unei conexiuni;
- dacă sunt eliminate ambele conexiuni, variabila nu este selectată, cu alte cuvinte, variabila nu apare în termenul produs.

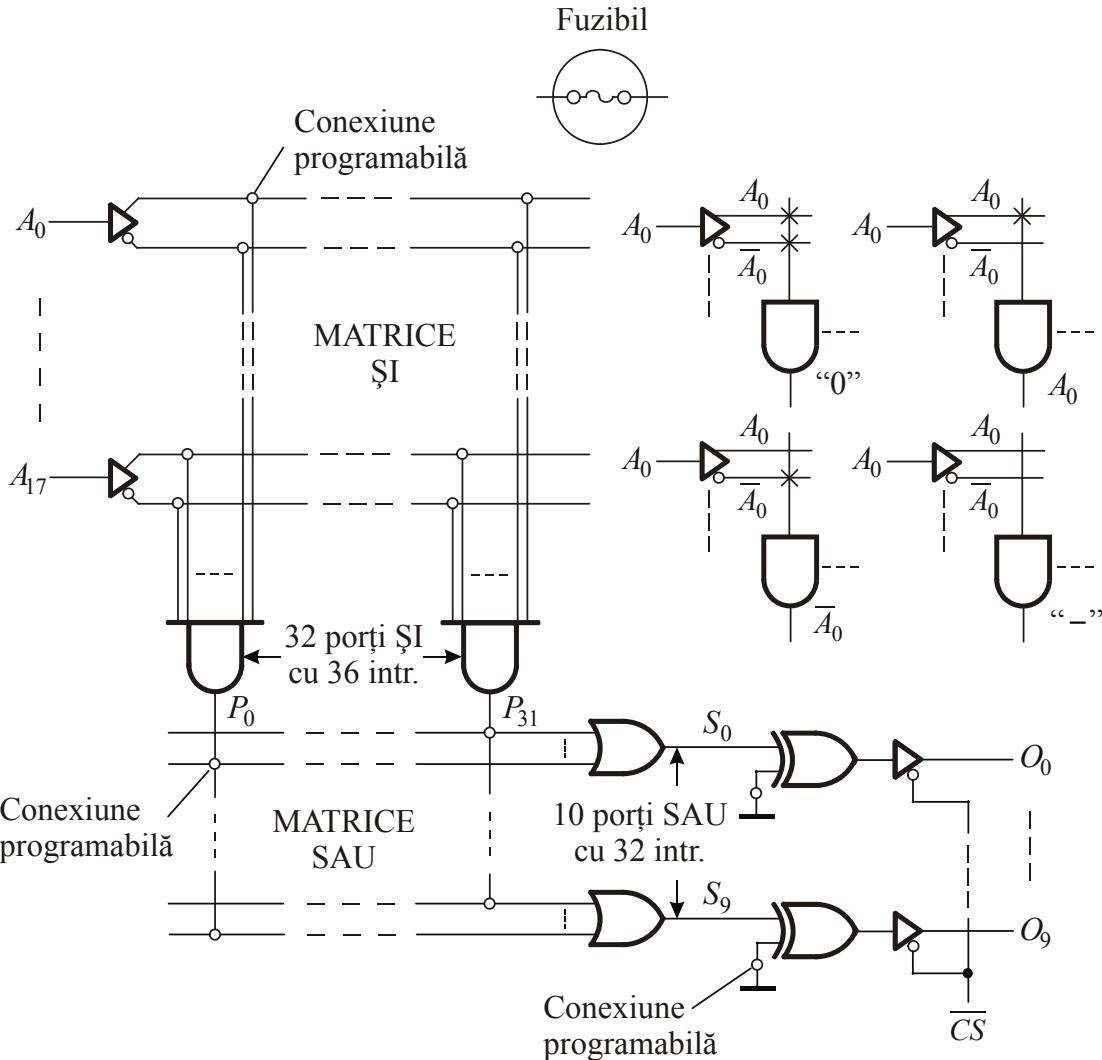


Fig.2.63 Structură tipică de FPLA.

La ieșirile portilor SI se obțin termenii produs (P_0, \dots, P_{31}), care pot fi exprimați prin:

$$P_k = \prod_0^{17} (i_n A_n + j_n \bar{A}_n), \text{ unde } k = 0, \dots, 31, \quad (2.135)$$

în care, dacă:

- $i_n = j_n = 0$ – intrare neprogramată,
- $i_n = \bar{j}_n$ – intrare programată,
- $i_n = j_n = 1$ – intrare redundantă.

Toate ieșirile portilor SI sunt conectate, în stare neprogramată, la intrările fiecărei porturi SAU.

SAU. La ieşirile portilor SAU se obțin semnale (S_0, \dots, S_9) , pentru care se poate scrie:

$$S_r = \sum_0^{31} t_k P_k, \text{ unde } r = 0, \dots, 9, \quad (2.136)$$

în care:

- $t_k = 0$, termen P inactiv (programat),
- $t_k = 1$, termen P activ (neprogramat).

Se pot forma deci zece sume logice, cu până la 32 de termeni, în fiecare sumă logică putând să intre oricare dintre cei 32 de termeni produs, P_0, \dots, P_{31} .

Circuitul de ieșire, realizat cu 10 porti SAU-EXCLUSIV (cu rolul de inversoare programabile: $S_i \oplus 0 = S_i$ (pentru fuzibil intact), respectiv $S_i \oplus 1 = \bar{S}_i$ (pentru fuzibil ars), unde $i = 0, \dots, 9$), permite obținerea la o ieșire O_i ($i = 0, \dots, 9$) a circuitului, fie a funcției directe ($O_i = S_i$), fie a funcției negate ($O_i = \bar{S}_i$). Posibilitatea interconectării ieșirilor cu circuite similare sau magistrale este asigurată prin amplificatoare de ieșire (buffere) cu trei stări (TSL) sau cu colector deschis (OC), controlate de o bornă comună \bar{CS} . Toate conexiunile, marcate prin cerculete în figură, pot fi întrerupte prin programare.

Avantajele pe care le oferă PLA/FPLA-ul se evidențiază prin comparația cu un ROM având același număr de intrări (18) și ieșiri (10).

PLA-ul are o capacitate mult mai mică (32 cuvinte \times 10 biți) față de ROM-ul echivalent (2^{18} cuvinte \times 10 biți), deci dacă corespunde aplicației avute în vedere, constituie o soluție mai economică.

Pentru memoria ROM, fiecărei combinații a intrărilor de adresă îi corespunde un cuvânt de ieșire, în schimb în cazul PLA-ului, o combinație a intrărilor poate selecta un cuvânt sau niciunul, respectiv același cuvânt poate fi selectat prin mai multe combinații ale intrărilor (adrese). Această ultimă proprietate rezultă din posibilitatea de a programa unele intrări redundante ("don't care"), fapt care duce la importante reduceri ale matricei decodificatoare.

Față de memoria ROM, ieșirile pot fi programate individual ca active pe 0 logic sau pe 1 logic. O astfel de facilitate asigură, în unele cazuri, minimizarea numărului termenilor produs (P) utilizati pentru implementarea unei funcții f . Un exemplu în această privință poate fi funcția cu patru termeni produs:

$$f = x_1 \bar{x}_2 + \bar{x}_1 x_2 + x_2 x_3 + \bar{x}_2 \bar{x}_4, \quad (2.137)$$

care se poate implementa și prin varianta ei cu 2 termeni produs:

$$\begin{aligned} f &= \overline{x_1 x_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_4} = (\bar{x}_1 + \bar{x}_2 + x_3)(x_1 + x_2 + \bar{x}_4) = \\ &= x_1 \bar{x}_2 + \bar{x}_1 x_2 + x_2 x_3 + \bar{x}_2 \bar{x}_4 + \underbrace{\bar{x}_1 \bar{x}_4 + x_1 x_3 + x_3 \bar{x}_4}_{\text{redundanță algebraic}}. \end{aligned} \quad (2.138)$$

Rezultă din cele de mai sus că avantajele pe care le oferă PLA-ul față de ROM, într-o serie de aplicații care presupun un număr mare de variabile de intrare, rezidă în posibilitatea programării matricei și (decodificatoare) și a complementării variabilelor

de ieșire.

Pe de altă parte, la un *circuit PLA* există *flexibilitatea de a genera simultan mai multe funcții, spre deosebire de un circuit ROM care poate genera doar succesiv mai multe funcții*. Un PLA de capacitate mai mare se poate partaja în mai multe segmente disjuncte, în fiecare segment intrând anumite elemente din lanțul intrări-portă řI-portă SAU- ieșiri; câte un astfel de segment poate fi asignat pentru implementarea uneia din funcții, eventual un segment poate fi lăsat neprogramat pentru modificări ulterioare [30].

Uzual, portile řI din cadrul matricei decodificatoare a FPLA-ului sunt realizate cu diode Schottky conectate în serie cu o peliculă fuzibilă de crom-nichel, iar portile SAU, care formează matricea codificatoare, sunt constituite din tranzistoare în conexiunea colector comun, având pelicula fuzibilă inserată în circuitul de emitor (v. fig. 2.64).

Programarea circuitului în vederea realizării unui sistem de funcții booleene necesită, aşa cum s-a mai precizat, formele minime disjunctive ale acestora. Este de dorit o minimizare corelată a funcțiilor sistemului pentru ca numărul termenilor distincți cu care se realizează funcțiile să fie cât mai mic, în orice caz cel mult 32.

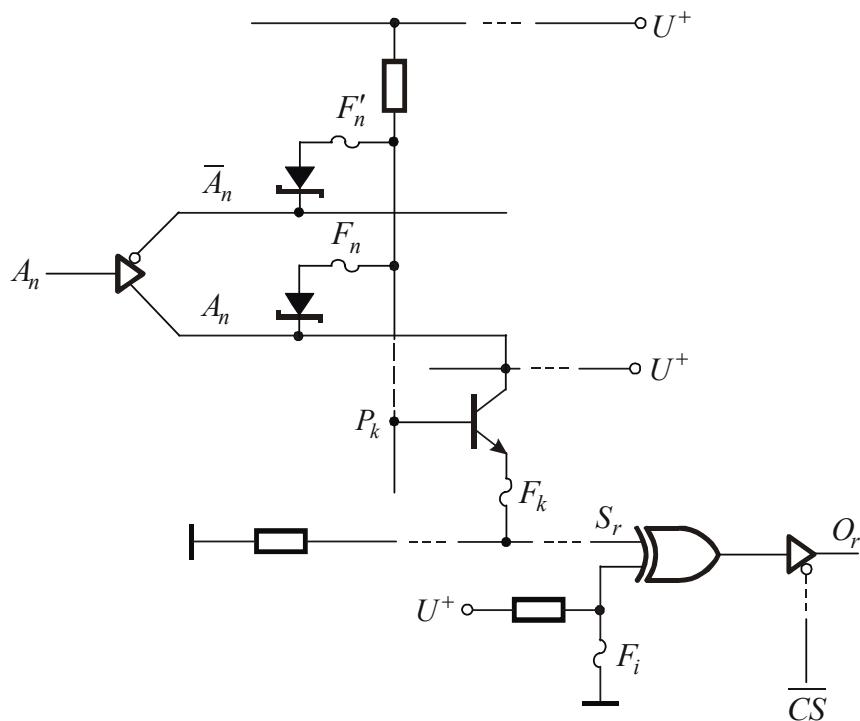


Fig.2.64 Structura electronică a unei matrice logice programabile.

Programarea FPLA-ului se realizează pe baza tabelului întocmit de utilizator, folosindu-se un programator, care permite selecția și arderea prin impulsuri de curent a fusibilelor a cărui întrerupere este necesară. În stare neprogramată, fiecare dintre cei 32 termeni P conține toate variabilele de intrare, precum și negațele acestora, iar fiecare din cei 10 termeni S este constituit din suma tuturor termenilor P . Potrivit celor de mai sus, în stare neprogramată, toți termenii P și toate sumele S sunt pe 0 logic și același lucru se poate spune despre ieșirile O .

La programarea matricei decodificatoare, pentru fiecare termen P_k se întrerupe fuzibilele care conectează coloana P_k la fiecare variabilă de intrare astfel încât P_k să conțină toate variabilele de intrare cu starea logică dorită (1, 0, *). Dacă P_k conține variabila A_n se

întrerupe fuzibilul F'_n și invers, dacă conține \bar{A}_n se întrerupe F_n , iar dacă se dorește excluderea ambelor variabile (A_n și \bar{A}_n) se întrerup atât F_n cât și F'_n (starea logică nedeterminată, marcată cu “*”). Dacă se utilizează mai puțin de 18 variabile de intrare, variabilele nefolosite intervin ca redundante în toți termenii P_k folosiți.

Prezența sau absența unui termen P în funcția de ieșire O este programată în matricea SAU (codificatoare). Absența termenului P_k în funcția S_r (v. fig. 2.64) se programază prin întreruperea fuzibilului F_k . Nu este necesară programarea matricei SAU pentru termenii P nefolosiți (neprogramați), aceștia fiind oricum pe 0 logic.

Programarea ieșirii se face în felul următor: dacă se cere realizarea negatei unei funcții O_j , ai cărei termeni sunt reuniți, în urma programării matricei řI, respectiv a matricei SAU, pe linia S_j se arde fuzibilul de la intrarea portii SAU-EXCLUSIV corespondente. În acest fel, intrarea respectivă ajunge în starea logică 1 și generează la ieșirea portii SAU-EXCLUSIV pe \bar{S}_j .

Exemplu. În figura 2.65 se exemplifică programarea funcției:

$$f = \overline{x_1 + \bar{x}_2 \bar{x}_3} . \quad (2.139)$$

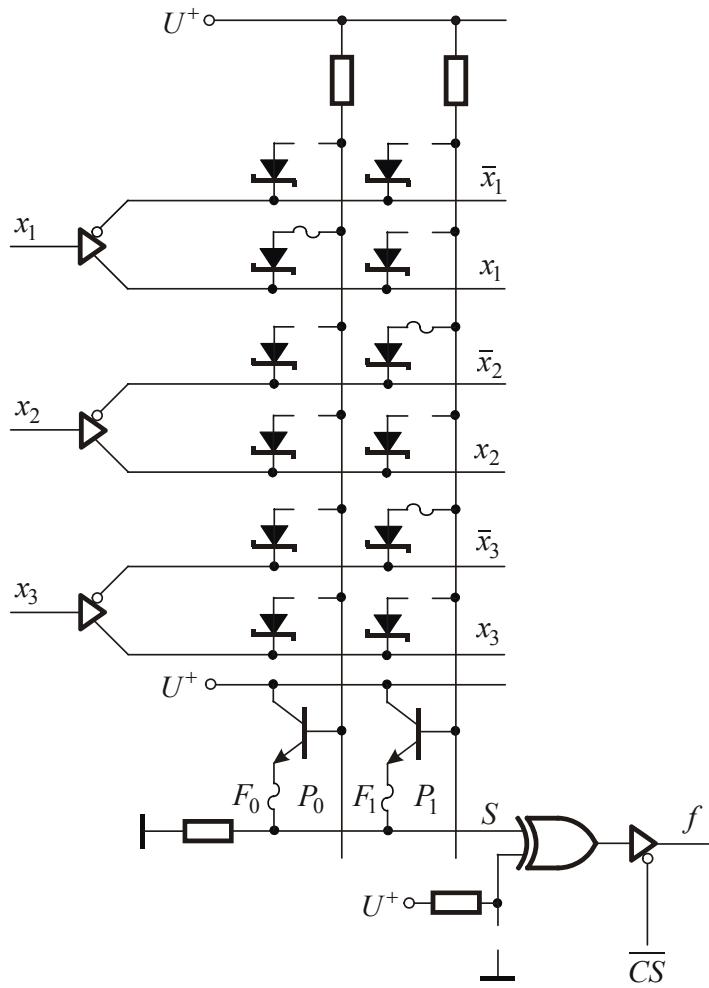


Fig.2.65 Programarea funcției din exemplu.

Este de menționat că, în acest caz, “coloanele fuzibile” care conectează termenii P_2, \dots, P_{31} la linia S se întrerup numai dacă termenii respectivi sunt programati și intervin în restul de 9 funcții de ieșire.

Principalele aplicații ale FPLA-urilor sunt în microprogramare, generarea de caractere, conversia de cod, realizarea tabelelor de funcții și implementarea automatelor secvențiale.

Dispozitivele PLA descrise mai sus au atât conexiuni programabile la intrările porților SI, cât și conexiuni programabile la intrările porților SAU. Firma Advanced Micro Devices, Inc. fabrică o familie de dispozitive PLD având *conexiuni programabile numai la intrările porților SI*, sau, altfel spus, aceste circuite dispun numai de *un nivel SI programabil*, iar termenii produs generați sunt aplicații unei *rețele SAU fixe (neprogramabile)*. Producătorii numesc acest dispozitiv *matrice programabilă logic* (PAL), termen care este acceptat unanim pentru aceste tipuri de dispozitive [33,37]. Într-un PAL, fiecare poartă SI este conectată direct la o intrare a unei porți SAU. În mod tipic, opt porți SI vor fi conectate direct la intrările fiecărei porți SAU (de opt intrări). Un motiv pentru această restricție este creșterea vitezei operațiilor, deoarece o conexiune programabilă implică o întârziere mai mare decât o legătură directă.

PAL-urile originale introduse în anii '70, au fost proiectate pentru tehnologia bipolară cu fuzibile semiconductoare bipolare, dar acum sunt disponibile versiuni pin-compatibile CMOS, cu ștergere electronică. După un start destul de lent, PAL-urile au devenit acum foarte larg folosite, înlocuind tradiționalele componente logice SSI/MSI într-o serie de aplicații (de exemplu, interfețele de calculatoare). Aceste dispozitive au în prezent furnizori multiplii. Există, de asemenea, versiuni de circuite logice secvențiale de acest tip (PAL cu registre).

Un exemplu de circuit combinatorial PAL este prezentat în figura 2.66. Schema logică corespunde circuitului integrat PAL CE16V8/4 [37] (litera C vine de la CMOS, E de la *erasable*-ștergere, iar V de la *variable*-variabil pentru ieșire (pune la dispoziție atât ieșire inversată cât și neinversată)). Acest dispozitiv are 16 intrări și 8 ieșiri, care sunt prevăzute cu buffere TSL și poate genera 8 expresii sumă-de-produse, fiecare conținând 8 termeni produs și fiecare termen produs având maximum 16 variabile (adevărate sau negative). Nu reprezintă un avantaj folosirea același termen produs în mai multe expresii sumă-de-produse, deoarece, în PAL, fiecare termen trebuie să fie separat pentru fiecare poartă SAU.

La dispozitivul a cărui schemă simplificată este prezentată în figura 2.66, pot fi programate fie ieșiri active în 0 logic, fie ieșiri active în 1 logic, prin încorporarea unui circuit inversor/non-inversor (poarta SAU-EXCLUSIV). Conexiunea programabilă lăsată intactă va produce o ieșire neinversată (adică ieșirea = intrarea), în timp ce eliminarea conexiunii programabile va crea o ieșire inversată (adică ieșirea = non-intrarea). Inversorul programabil este plasat, în mod tipic, după poarta SAU și înaintea bufferului de ieșire. Aceste PAL-uri sunt uneori numite *matrice/arii logice generice*, *GAL*¹³ (*Generic Logic Array*) și intră în categoria de circuite PAL universale.

O altă variantă de circuit logic programabil o reprezintă, aşa numita, *matrice de porți programabilă* (PGA) [23,30].

Circuitele integrate de tipul PGA sunt mai simple, din punct de vedere constructiv, decât circuitele PLA, respectiv PAL. Un circuit PGA conține *un singur nivel de porți lo-*

¹³ GAL este o marcă înregistrată a firmei Lattice Semiconductor Corp.

gice programabil, dispuse într-o matrice řI (nu mai apare deci matricea SAU) legată direct la circuitele de iešire prin intermediul cărora se programează funcția directă, respectiv funcția negată. Prin programarea matricei řI și a circuitelor de iešire se pot realiza funcțiile řI, řI-NU, SAU și SAU-NU, de un număr mare de variabile.

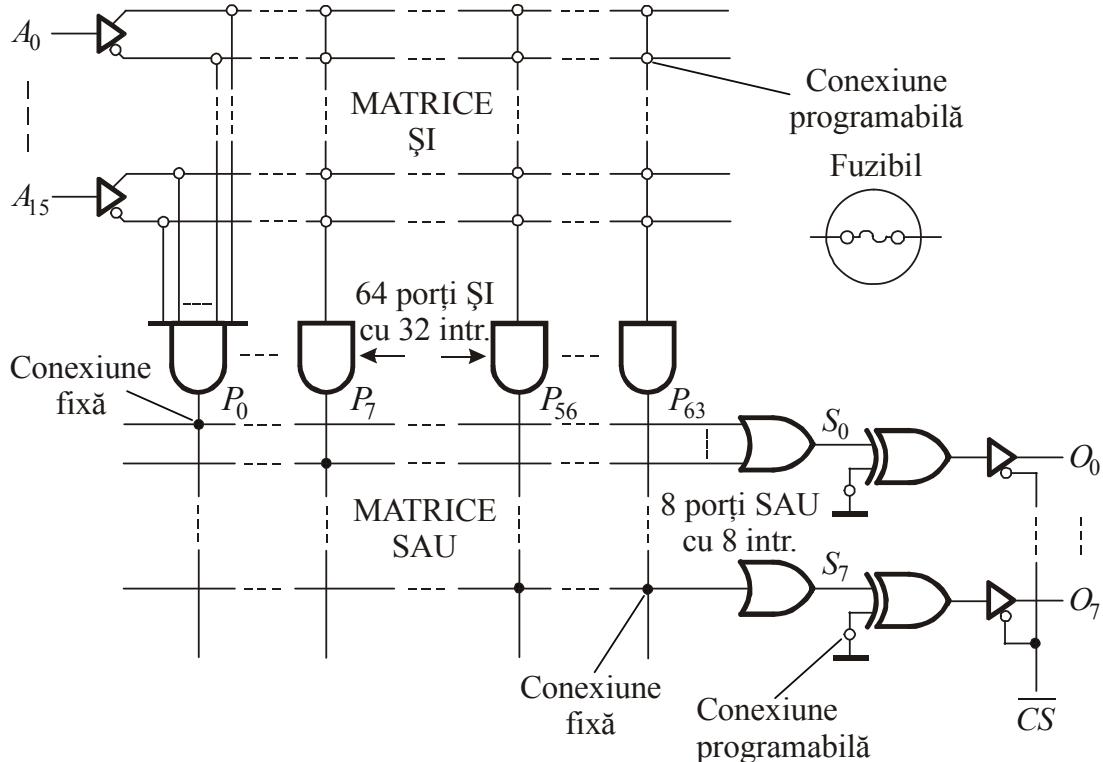


Fig. 2.66 Structură tipică de PAL.

Schema logică a unui circuit PGA, cu 16 intrări și 9 ieširi este prezentată în figura 2.67. Ea corespunde circuitului integrat 82S103 [38] și are ieširile TSL (există circuite integrate PGA și în varianta la care ieširile sunt OC, de exemplu 82S102 [38]).

Circuitele PGA sunt folosite ca elemente logice universale. Cu ajutorul lor se pot realiza CLC-uri pentru diverse aplicații sau circuite specializate.

În ultimii ani s-au impus FPGA-urile secvențiale, firma Xilinx, Inc. fiind cel mai important producător de dispozitive programabile FPGA (seria XC4000 [39]).

Pentru a folosi PLD-urile, trebuie identificate conexiunile care trebuie eliminate. Pentru aranjamente simple, se poate lucra pe schema de conexiuni din diagrama logică a PLD-ului, dar, în mod normal se folosesc programe software pentru a se îndeplini această sarcină. Un astfel de program va lua, ca intrare, o specificație a funcției logice care trebuie realizată. Specificația poate fi sub forma unei expresii booleene descrise printr-un tabel de adevăr, sau (pentru circuitele secvențiale) o descriere printr-o diagramă de stări. Ieširea programului va include un fișier care conține conexiunile necesare (*harta de programare*). Fișierul de iešire este apoi folosit pentru a comanda direct unitatea de programare PLD. Unele programe includ, de asemenea, și un simulator logic astfel încât proiectul logic poate fi verificat înaintea programării PLD-ului. Printre programele populare se pot aminti *PALASM2* (*PAL assembler*) de la firma AMD, Inc. și *ABEL* (*Advanced Boolean Expression Language*), marcă înregistrată a firmei Data I/O Corporation [33].

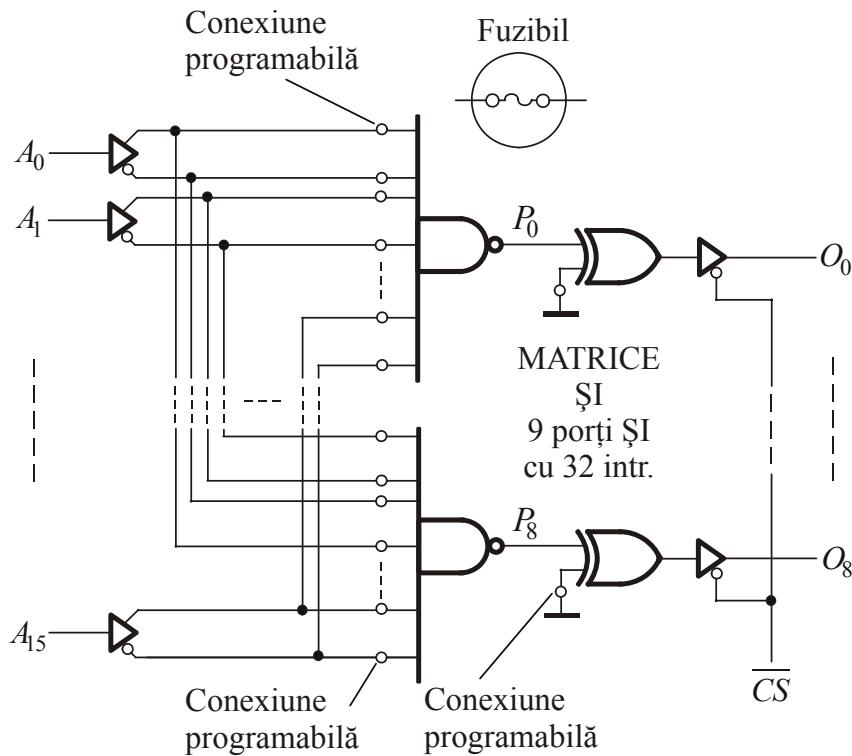


Fig.2.67 Schema logică a unei matrice de porturi programabilă (PGA).

• Implementarea funcțiilor booleene cu dispozitive logice programabile

Una dintre principalele aplicații ale dispozitivelor logice programabile o reprezintă implementarea sistemelor de funcții booleene. Utilizarea în acest scop a PLD-urilor oferă, față de alte metode de implementare, o serie de avantaje dintre care sunt de menționat: flexibilitatea programării, reducerea spațiului ocupat prin reducerea substanțială a numărului componentelor, puterea consumată mai mică și fiabilitatea mult îmbunătățită.

PLA/FPLA-ul este o structură extrem de flexibilă care se poate utiliza eficient pentru implementarea sistemelor de funcții cu mai mult de 8 variabile de intrare. Dacă se aleg în mod convenabil intrările (A_k sau \bar{A}_k) și se folosește posibilitatea de inversare a funcțiilor de la ieșire se poate realiza un CLC în diverse variante.

Așa cum a rezultat din descrierea circuitelor PLA/FPLA, utilizarea eficientă a capacitatei acestui circuit, în cazul implementării unor sisteme logice combinatoriale, necesită operații de minimizare a funcțiilor booleene, similare cu cele întâlnite la implementarea cu porți logice.

PLA/FPLA-urile tind să acopere, la ora actuală, toate acele structuri de circuite logice combinatoriale de complexitate mică și medie, pentru care o implementare cu circuite standard (utilizate ca circuite logice universale) MUX, DMUX/DCD, ROM ar genera un număr foarte mare de termeni produs.

Exemplu. Să se implementeze cu FPLA circuitul logic combinatorial definit de următorul sistem de funcții:

$$f_1^{\text{FMD}} = x_1 \bar{x}_2 x_{17} + \bar{x}_3 \bar{x}_{16} \bar{x}_{17} x_{18}, f_2^{\text{FMD}} = x_2 x_3 + \bar{x}_2 \bar{x}_{17} + x_1 \bar{x}_{18}, \dots, f_{10}^{\text{FMD}} = x_1 \bar{x}_2 x_{17} + x_{17} \bar{x}_{18}. \quad (2.140)$$

Se observă că funcțiile sunt date în forma minimă disjunctivă. Este necesar un FPLA cu 18 intrări și 10 ieșiri și care generează 32 de termeni produs (termeni interni). Pentru implementarea CLC-ului se folosește circuitul PLS153 ($18 \times 32 \times 10$) [36] (v. fig. 2.68). X-urile de pe primul nivel logic arată unde trebuie lăsate conexiunile intacte, sau indică variabilele x_k , respectiv \bar{x}_k care se cuplăză la ȘI-uri. Aceste ȘI-uri implementeză de fapt termenii produs care intervin în expresiile funcțiilor. X-urile de pe al doilea nivel logic arată care dintre termenii produs se cuplăză la SAU-ul respectiv. Se observă, de asemenea, că termenul $x_1 \bar{x}_2 x_{17}$ intervine și în f_1 și în f_{10} , deci acesta se poate forma doar o singură dată, la ȘI-ul P_0 .

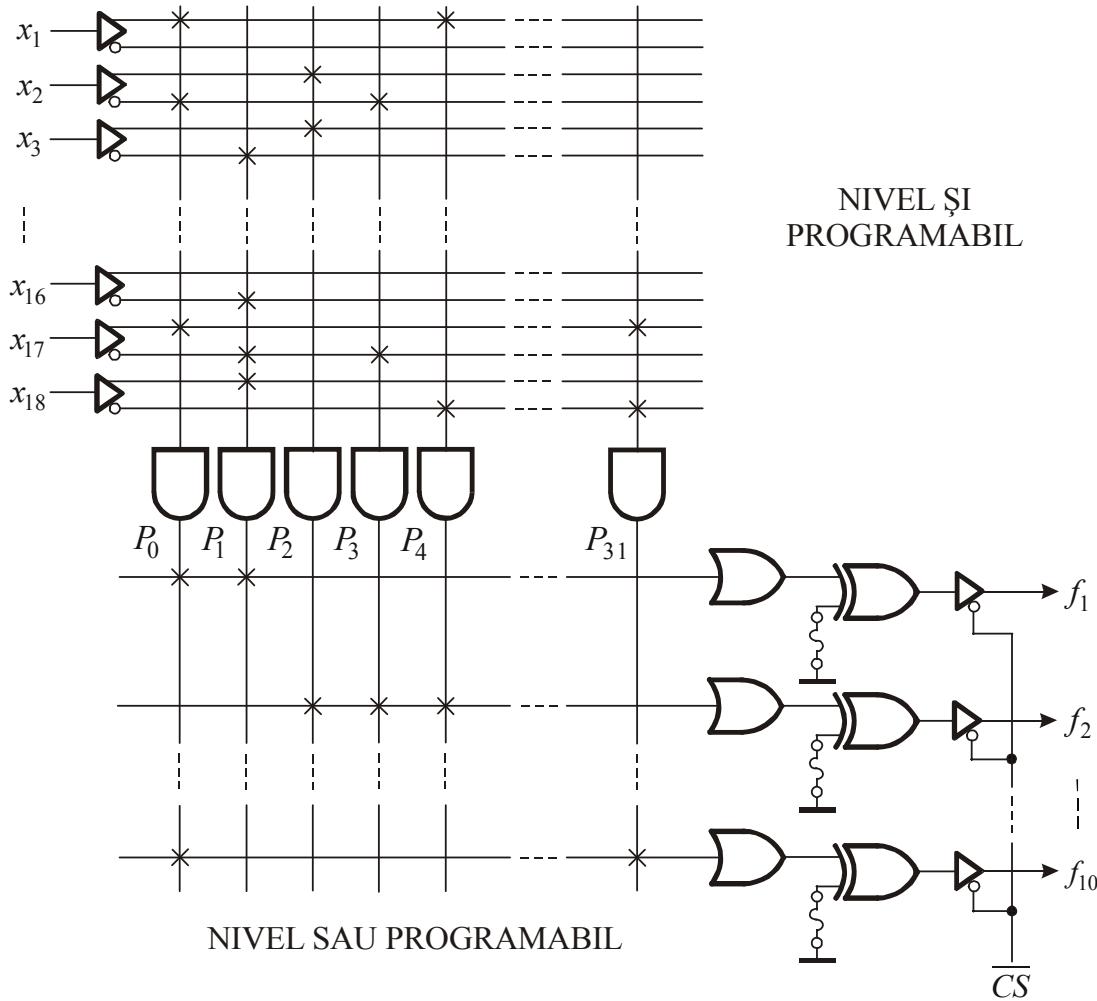


Fig.2.68 Realizarea cu FPLA a sistemului de funcții din exemplu.

Observație. La implementarea cu PLA-uri există o restricție datorată limitării numărului de termeni P distincți ai grupului de funcții. Minimizarea, în acest caz, urmărește obținerea a cât mai multor termeni comuni per ansamblu și nu este întotdeauna identică cu minimizarea matematică.

Extinderea capacitatei PLA/FPLA-urilor este o problemă care se poate pune în mai multe moduri, dintre care următoarele două sunt cele mai frecvent întâlnite în practică:

Exemplu. Utilizându-se circuite PLA cu N intrări, P termeni interni și M ieșiri, să se realizeze scheme pentru:

a) extinderea numărului de termeni interni, păstrând același numărul de intrări și ieșiri;

b) extinderea numărului de termeni interni și ieșiri, N rămânând același.

Creșterea numărului de termeni produs (P), fără modificarea numărului de intrări sau ieșiri, este realizabilă prin cuplarea în paralel a intrărilor și ieșirilor mai multor circuite și selectarea lor succesivă cu ajutorul unui DCD, prin intermediul semnalului \overline{CS} . Cu schema din figura 2.69,a, în care PLA-urile sunt cuplate în paralel (pe intrări și ieșiri) se obțin de n ori mai mulți termeni interni.

Extinderea lungimii cuvântului de ieșire, fără modificarea numărului variabilelor de intrare, este posibilă prin cuplarea în paralel a intrărilor mai multor circuite și conectarea la 0 logic a intrărilor \overline{CS} . În acest caz, toate circuitele sunt adresate simultan, obținându-se $n \times M$ funcții de ieșire la n circuite folosite. Astfel, dacă se decouplează ieșirile din paralel în schema din figura 2.69,a, se obține un PLA cu $n \times M$ ieșiri și $n \times P$ termeni interni (v. fig. 2.69,b).

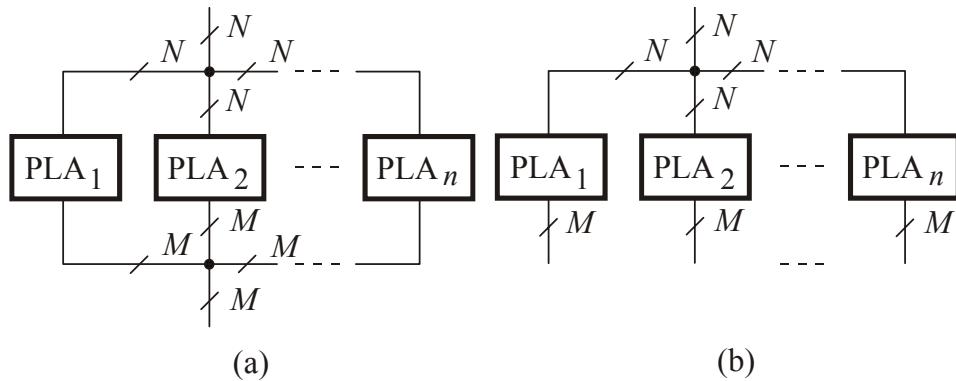


Fig.2.69 Extinderea capacitatei PLA-ului: (a) extinderea numărului de termeni interni; (b) extinderea numărului de ieșiri.

Circuitul PAL se caracterizează prin aceea că numai nivelul \overline{SI} este programabil, iar termenii produs generați sunt aplicații nivelului SAU fix (neprogramabil). Intuitiv, acest circuit poate fi privit ca “o imagine în oglindă” a memoriei ROM (care are nivelul SAU programabil și nivelul \overline{SI} fix). Din acest motiv, la ieșirile matricei \overline{SI} se obțin termeni produs, dar nu neapărat canonici, motiv pentru care implementarea funcțiilor cu astfel de circuite nu trebuie să plece de la formele canonice disjunctive ale funcțiilor. În aplicațiile practice, circuitul PAL apare ca un grup de porți SAU comandate de mai multe porți \overline{SI} .

Exemplu. Să se implementeze cu PAL circuitul logic combinatorial definit de următoarele funcții:

$$\begin{aligned} f_1 &= x_2\bar{x}_3 + x_1x_3x_4 + \bar{x}_1\bar{x}_2x_3, & f_2 &= \bar{x}_1\bar{x}_2x_3 + x_2x_4 + x_1x_2\bar{x}_3, \\ f_3 &= x_1x_3 + \bar{x}_1\bar{x}_3, & f_4 &= x_2\bar{x}_3 + x_2x_4. \end{aligned} \quad (2.141)$$

În figura 2.70 se prezintă soluția de implementare cu PAL. Se folosește o parte din circuitul PAL CE16V8/4 [37]. Remarcați că porțile \overline{SI} nefolosite au toate conexiunile lăsate intacte, ceea ce forțează un 0 pe intrările acestor porți. Ieșirile porților \overline{SI} vor fi permanent 0, neafectând astfel porțile SAU. În mod obișnuit însă, pentru simplitate, conexiunile pentru porțile nefolosite nu se marchează.

Se observă că în expresiile funcțiilor apar termeni comuni, dar acest lucru nu repre-

zintă un avantaj în cazul implementării cu PAL, pentru că, aşa cum s-a mai precizat, într-un PAL, fiecare termen produs trebuie să fie separat pentru fiecare poartă SAU.

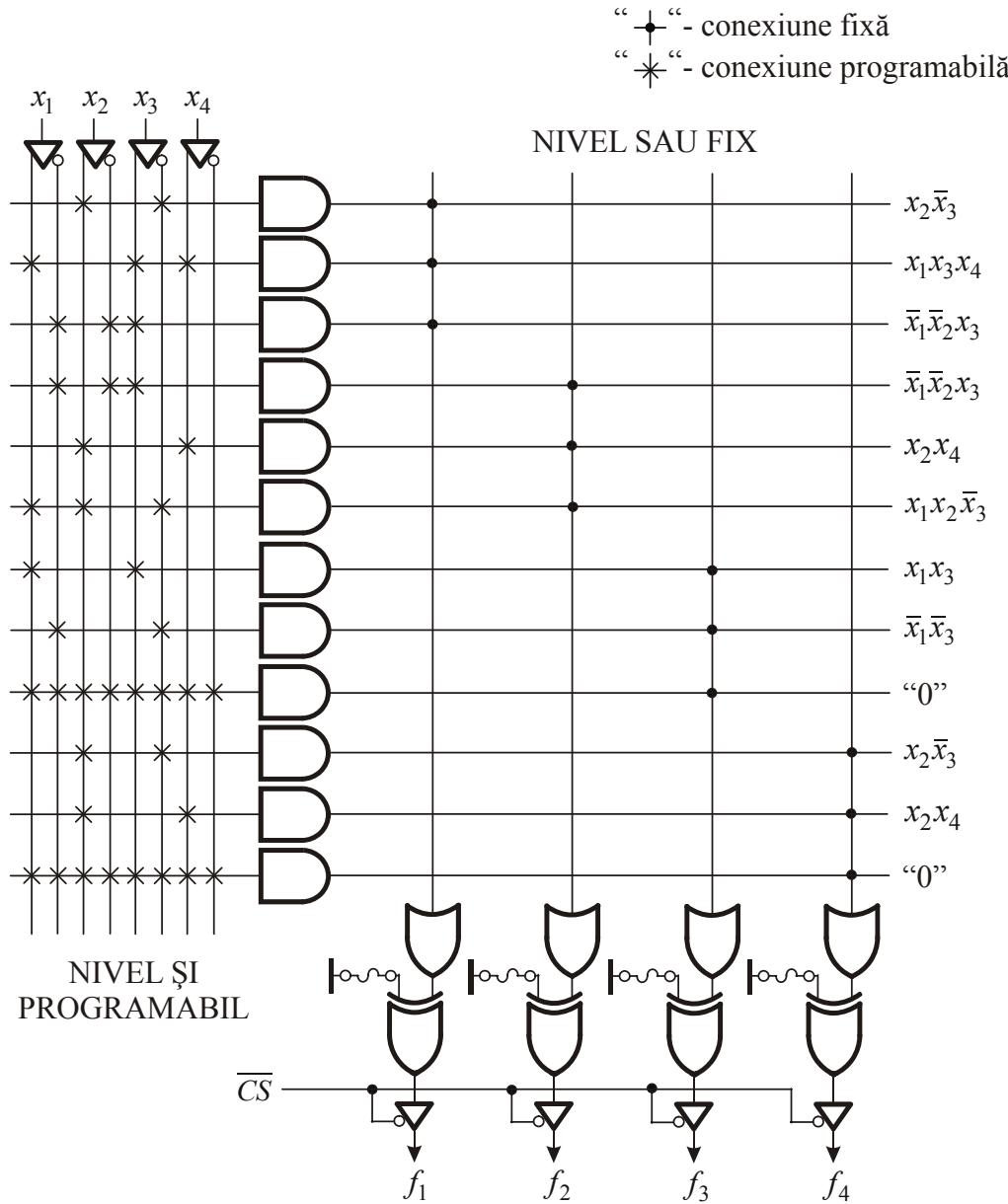


Fig.2.70 Implementarea sistemului de funcții din exemplu, folosindu-se un circuit PAL.

Deoarece circuitul PGA are un singur nivel ȘI/ȘI-NU programabil, domeniul de aplicații în care se poate folosi este destul de limitat. Prinț-o judecțioasă aplicare a teorelor lui De Morgan și prinț-o folosire corespunzătoare a complementării intrărilor și/sau ieșirilor, pot fi implementate cu astfel de circuite funcțiile booleene cu un singur nivel logic. Pe de altă parte, aceste circuite se pot cascada, în sensul că ieșirile mai multor circuite PGA pot fi aplicate ca intrări unui alt circuit, în acest mod obținându-se implementări mai complexe, pentru funcții cu două nivele logice.

Exemplu. Să se implementeze cu PGA circuitul logic combinatorial definit de sistemul de funcții:

$$\begin{aligned}
 f_1 &= \bar{x}_1 + x_2 + x_{14} + \bar{x}_{15} = \overline{\bar{x}_1 \bar{x}_2 \bar{x}_{14} x_{15}} \oplus 0 = \overline{\bar{x}_1 \bar{x}_2 \bar{x}_{14} x_{15}}, \\
 f_2 &= x_2 + x_3 + \bar{x}_{16} = \overline{\bar{x}_2 \bar{x}_3 x_{16}} \oplus 0 = \overline{\bar{x}_2 \bar{x}_3 x_{16}}, \\
 &\dots \\
 f_9 &= x_1 \bar{x}_2 \bar{x}_3 x_{14} \bar{x}_{15} = \overline{x_1 \bar{x}_2 \bar{x}_3 x_{14} \bar{x}_{15}} \oplus 1 = \overline{x_1 \bar{x}_2 \bar{x}_3 x_{14} \bar{x}_{15}}.
 \end{aligned} \tag{2.142}$$

În figura 2.71 este prezentat modul de utilizare al unui PGA pentru implementarea sistemului de funcții din exemplu (se folosește circuitul 82S103 [38]).

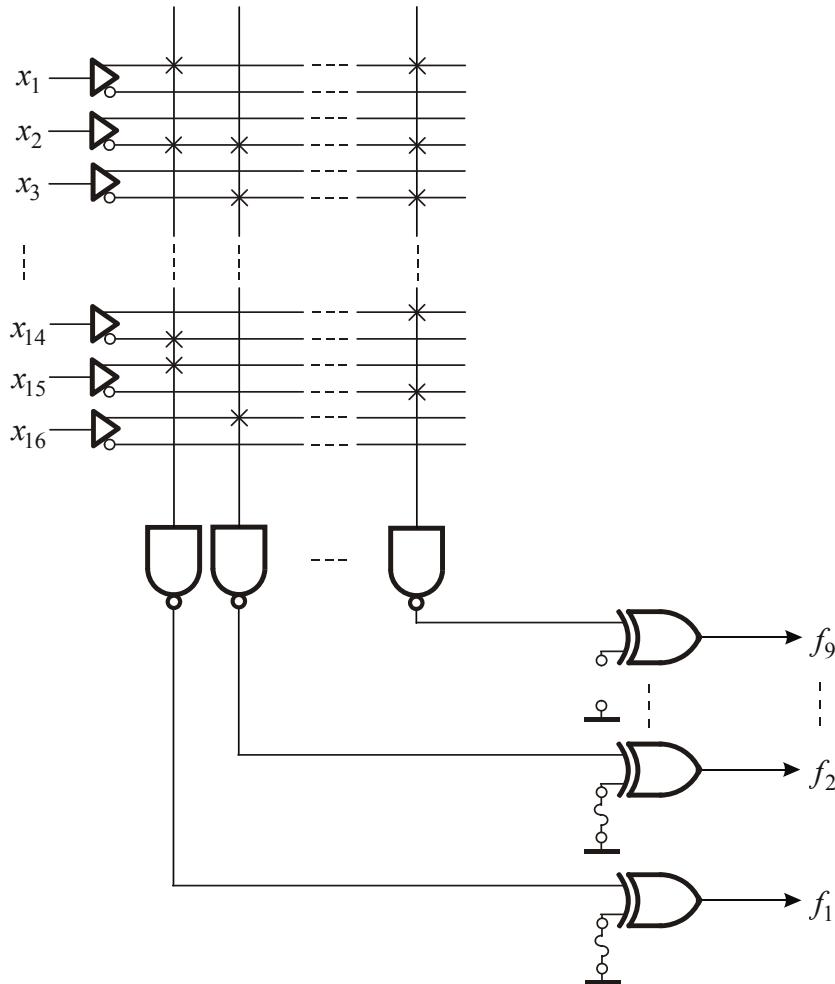


Fig.2.71 Utilizarea unui PGA pentru implementarea unui sistem de funcții cu un singur nivel logic.

În loc de concluzii, în figura 2.72 sunt prezentate, comparativ, variantele de implementare, pentru același set de funcții, utilizându-se cele patru tipuri de circuite integrate LSI studiate.

Exemplu. Să se implementeze circuitul logic combinatorial definit de funcțiile:

$$f_1 = x_1, \quad f_2 = \bar{x}_1 x_2, \quad f_3 = x_1 + \bar{x}_2, \quad f_4 = \bar{x}_1 x_2 + x_1 \bar{x}_2, \tag{2.143}$$

în următoarele variante: a) cu memorie PROM; b) cu FPLA c) cu PAL d) cu PGA.

Soluțiile de implementare obținute sunt prezentate în figura 2.72.

Funcția f_4 nu se poate implementa cu un singur circuit PGA deoarece expresia acesteia conține două nivele logice (v. fig. 2.72,d)

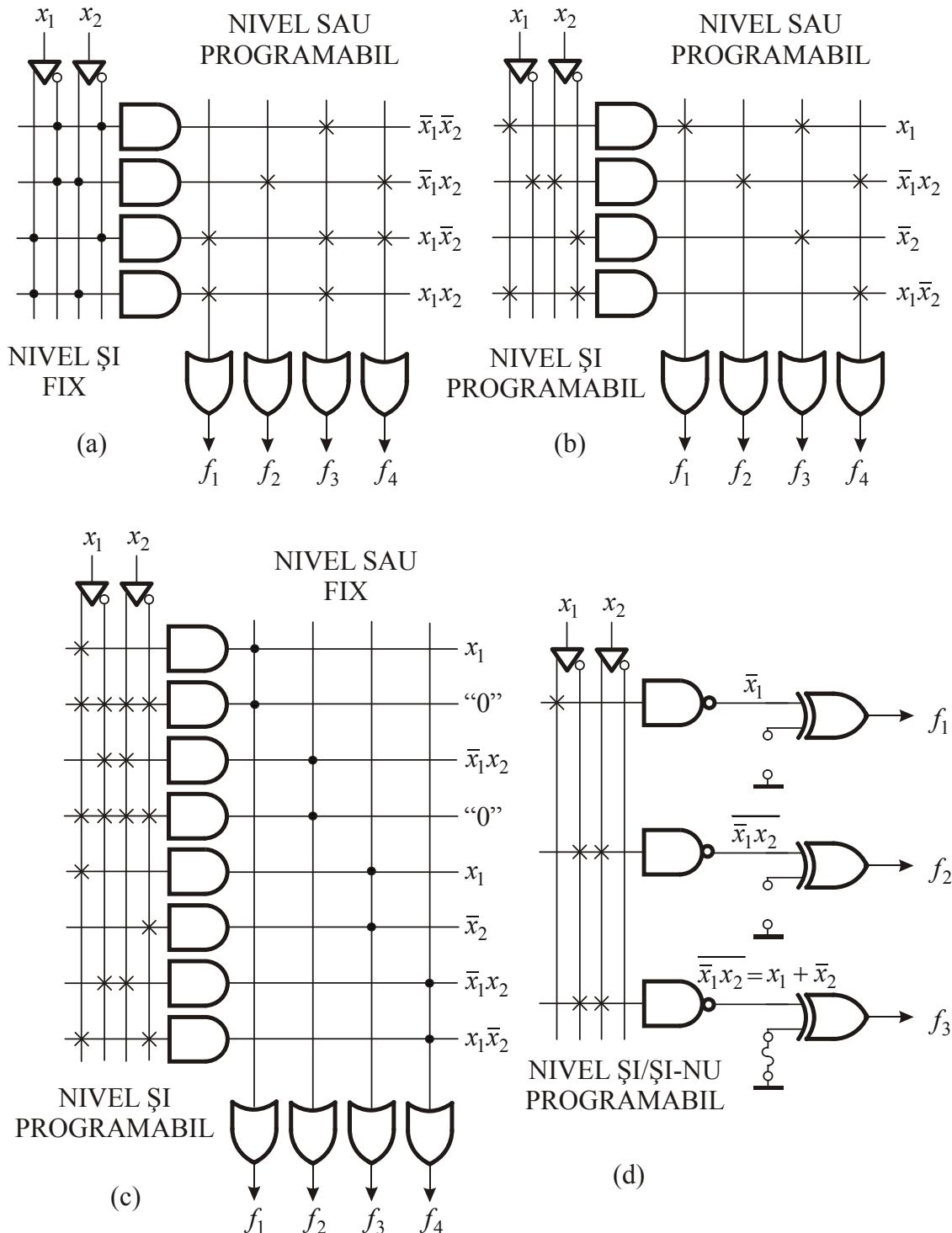


Fig.2.72 Variante de implementare utilizându-se diverse tipuri de circuite integrate LSI: (a) soluția cu memorie PROM; (b) soluția cu FPLA; (c) soluția cu PAL; (d) soluția cu PGA.

2.4 HAZARDUL ÎN CIRCUITELE LOGICE COMBINATORIALE

În studiul circuitelor logice combinatoriale de până acum s-a considerat că elementele logice cu care se materializează schemele sunt identice și comutarea lor este ideală. În realitate, datorită timpului finit de comutare, fiecare modul logic introduce întârzieri înrente, funcționarea în condiții reale determinând apariția unor probleme marginale care pot conduce la erori de funcționare ce pot să apară în timpul funcționării circuitului combinational. Astfel, întârzierile introduse pe „lanțul intrare-ieșire“ fac ca adeseori starea ieșirii circuitului, la un moment dat, să nu fie în concordanță cu starea intrărilor acestuia la momentul considerat. Se întâmplă astfel ca circuitul de comutare să aibă, pentru foarte scurt timp, o comportare greșită, fenomen care se numește *hazard* [2,10,23,24,27,30].

În cazul circuitelor logice combinatoriale există două tipuri distincte de hazard și anume: *hazardul static* și *hazardul dinamic*.

- **Hazardul static.** Atunci când în urma modificării stării intrărilor circuitului se modifică, pentru scurt timp, starea ieșirii, cu toate că nu este justificată din punct de vedere logic, încrucișând noii stări a intrărilor trebuie să îi corespundă aceeași stare a ieșirii, hazardul apărut se numește *static*.

El se caracterizează deci prin aceea că, pe o durată foarte scurtă, una sau mai multe ieșiri, care trebuie să fie 1(0) trec în 0(1), după care revin la starea inițială. În cazul în care ieșirea ar trebui să rămână 1 și trece, pentru foarte scurt timp în 0, avem hazard static în unități, iar dacă ieșirea trece pentru foarte scurt timp în 1 și ar fi trebuit să rămână 0, avem hazard static în zerouri. Pentru exemplificare se consideră câteva circuite logice combinatoriale la care se poate pune în evidență apariția hazardului static.

Exemplul 1. Analizând funcționarea CLC-ului din figura 2.73,a, care realizează următoarea funcție:

$$f^{\text{FMD}}(x_1, x_2, x_3) = \overline{x_1} \overline{x_3} \cdot \overline{x_2} \overline{x_3} = x_1 \overline{x_3} + x_2 x_3, \quad (2.144)$$

se constată că, în condițiile în care elementele componente au întârzieri diferite, la un moment dat, modificarea stării intrărilor x_1 , x_2 și x_3 din 110 în 111, determină apariția hazardului static, dacă $t_{p1} > t_{p2}$, unde t_{p1} este timpul de propagare al portii SI-NU marcată cu 1, iar t_{p2} este timpul de propagare al portii SI-NU marcată cu 2^{14} . Prezența hazardului static la CLC-ul studiat poate fi urmărită pe diagramele de semnal din figura 2.73, b, în care s-a reprezentat modificarea în timp a semnalelor la ieșirile portilor circuitului.

Exemplul 2. Fie CLC-ul din figura 2.74,a, realizând cu porti logice SI-NU expresia logică:

$$f^{\text{FMD}}(x_1, x_2, x_3) = x_1 x_2 + \overline{x}_2 x_3, \quad (2.145)$$

rezultată din diagrama Karnaugh din figura 2.74,c.

Se consideră o stare inițială a intrărilor $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, cu ieșirea $f = 1$ și modificarea $x_2 : 1 \square 0$ (în diagrama Karnaugh, din coloana 11 în coloana 10). Există

¹⁴ În acest caz t_{p1} este timpul de propagare la tranziția “sus-jos” a semnalului de ieșire (t_{pHL}) al portii 1, iar t_{p2} este timpul de propagare la tranziția “jos-sus” a semnalului de ieșire (t_{pLH}) al portii 2. În general, $t_{pHL} > t_{pLH}$.

posibilitatea ca, datorită întârzierii introduse de poarta řI-NU 1, starea intrărilor circuitului să fie $x_1\bar{x}_2\bar{x}_3$ (100), în loc de $x_1\bar{x}_2x_3$ (101), deci pe o durată foarte scurtă a regimului tranzistoriu $f:1 \rightarrow 0$, fără ca acest lucru să fie dorit (funcționare incorectă). După parcurgerea prin poarta řI-NU 1, $f:0 \rightarrow 1$ și circuitul trece în regimul corect de funcționare.

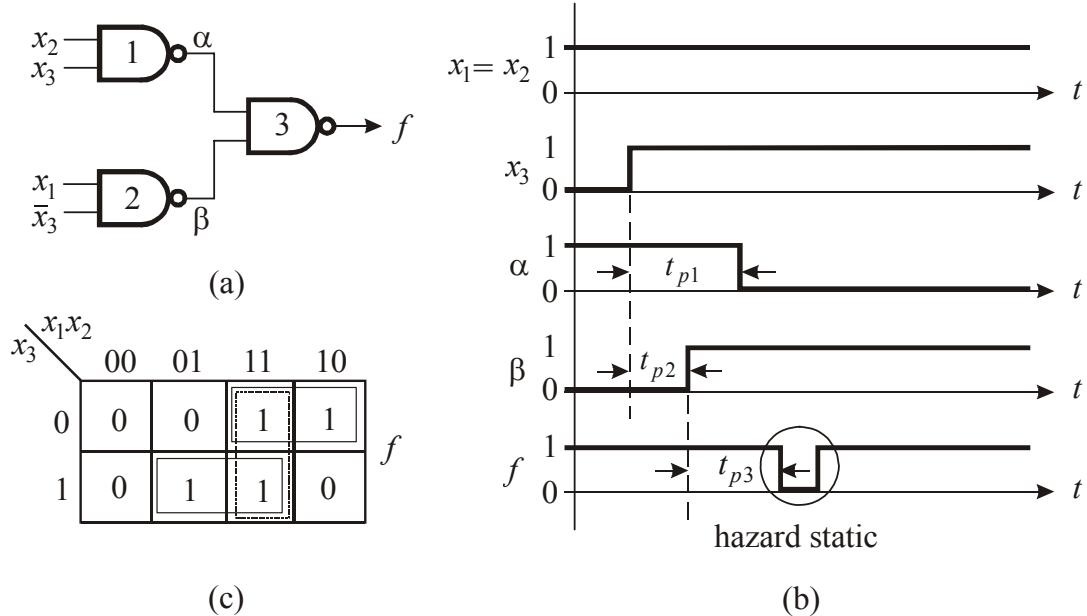


Fig.2.73 Evidențierea hazardului static la CLC-ul din exemplul 1.

Se poate arăta că dacă expresia logică a unui semnal într-un punct al circuitului combinatorial se poate reduce la una din formele $x + \bar{x}$ sau $x \cdot \bar{x}$, la ieșirea porții respective poate exista hazard. În caz contrar, circuitul nu generează semnale parazite.

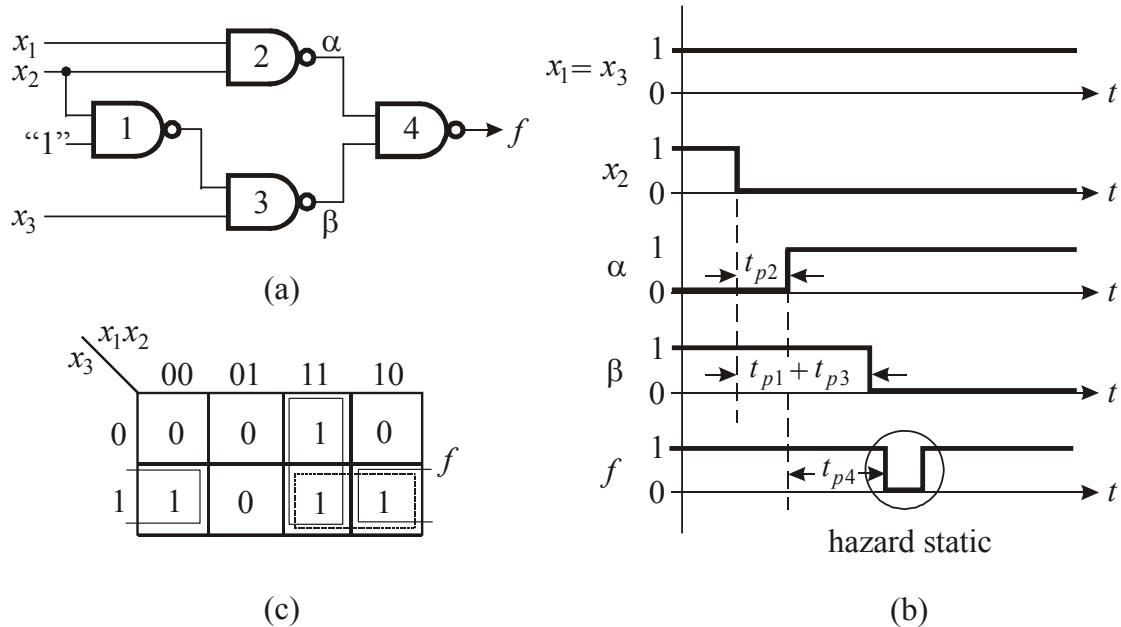


Fig.2.74 Evidențierea hazardului static la CLC-ul din exemplul 2.

Exemplul 3. Se consideră CLC-ul dat în figura 2.75,a, la ieșirea căruia se obține expresia logică:

$$f^{\text{FMD}}(x_1, x_2, x_3) = x_1 x_2 + \bar{x}_1 x_3. \quad (2.146)$$

Se constată că circuitul este afectat de hazard deoarece expresia (2.146) se reduce la $x_1 + \bar{x}_1$, când $x_2 = x_3 = 1$. Situația este ilustrată în figura 2.75,b.

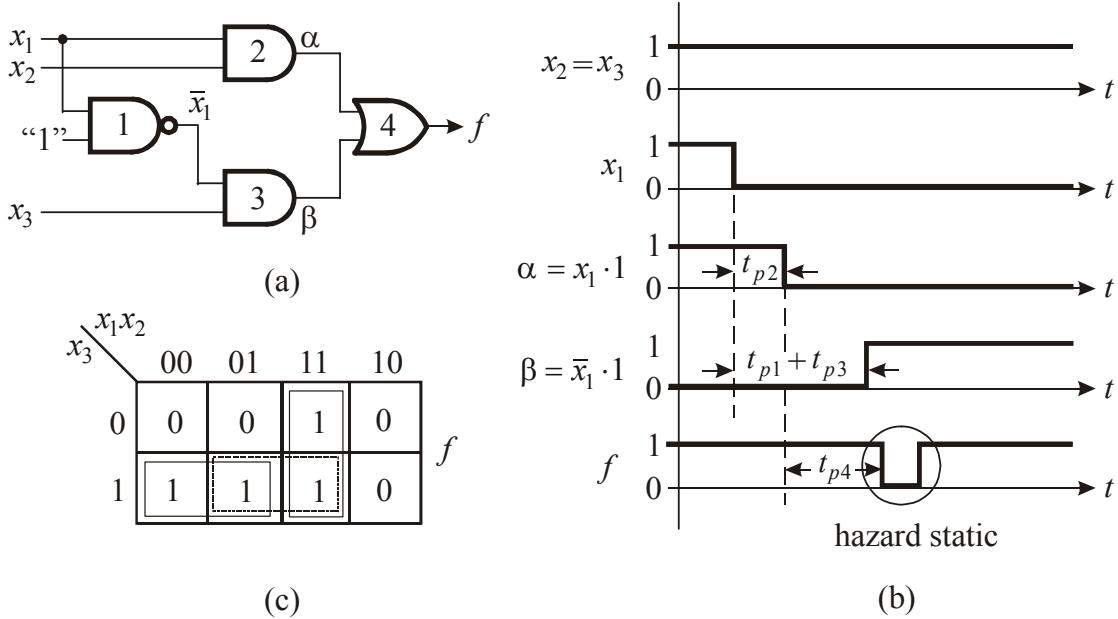


Fig.2.75 Evidențierea hazardului static la CLC-ul din exemplul 3.

Hazardul apare deoarece x_1 trece în 0 înainte ca \bar{x}_1 să treacă în 1 (datorită întârzierii introduse de poarta logică 1) și există un mic interval de timp în care $x_1 + \bar{x}_1 = 0$ (v. fig. 2.75,b) deși, ideal, matematic, $x_1 + \bar{x}_1 = 1$.

- **Hazardul dinamic.** Atunci când în urma modificării stării intrărilor, ieșirea trebuie să-și modifice starea, lucru normal din punct de vedere logic, dar aceasta nu se face direct ci după un număr de oscilații între noua și vechea stare, hazardul apărut se numește *dinamic*.

Hazardul dinamic poate să apară în circuitele de comutare cu mai multe nivele logice, în care există porțiuni de circuit cu hazard static. Se poate demonstra că un circuit care realizează o funcție logică sub formă canonica disjunctivă și care nu are hazard static privind unitățile, nu prezintă nici alte tipuri de hazard static sau dinamic [24].

Exemplul 4. Circuitul logic combinațional a cărui schemă logică este prezentată în figura 2.76,a, realizează funcția:

$$\begin{aligned} f^{\text{FMD}}(x_1, x_2, x_3, x_4, x_5) &= (x_2 \bar{x}_4 + x_3 x_4)(x_1 + x_5) = \\ &= x_1 x_2 \bar{x}_4 + x_1 x_3 x_4 + x_2 \bar{x}_4 x_5 + x_3 x_4 x_5, \end{aligned} \quad (2.147)$$

reprezentată pe diagrama Karnaugh din figura 2.76,c.

Se presupune că, din întâmplare, întârzierile introduse de porțile logice sunt foarte

diferite și $t_{p1} < t_{p2}$, iar $t_{p4} > t_{p1} + t_{p3}$, unde cu t_{pi} s-a notat, aşa cum s-a mai menționat, timpul de propagare a semnalului prin poarta i .

În aceste condiții, la ieșirea circuitului apare hazard dinamic, dacă starea intrărilor x_1, x_2, x_3, x_4, x_5 se modifică, de exemplu, din 01101 în 01110. Acest fapt se poate urmări pe diagrama Karnaugh din figura 2.76,b. La apariția hazardului dinamic la ieșirea circuitului f , a contribuit și hazardul static apărut la ieșirea intermedieră γ .

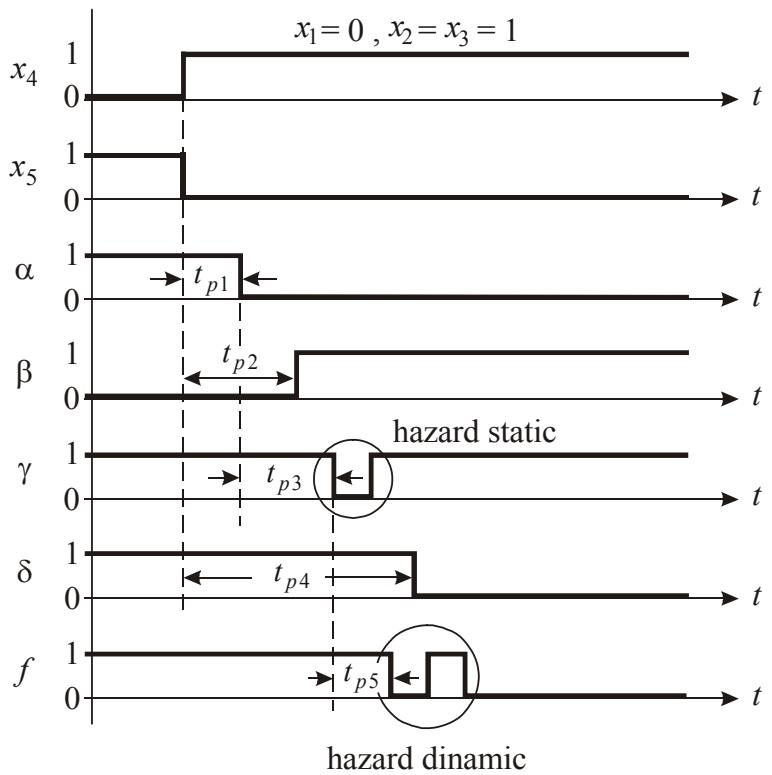
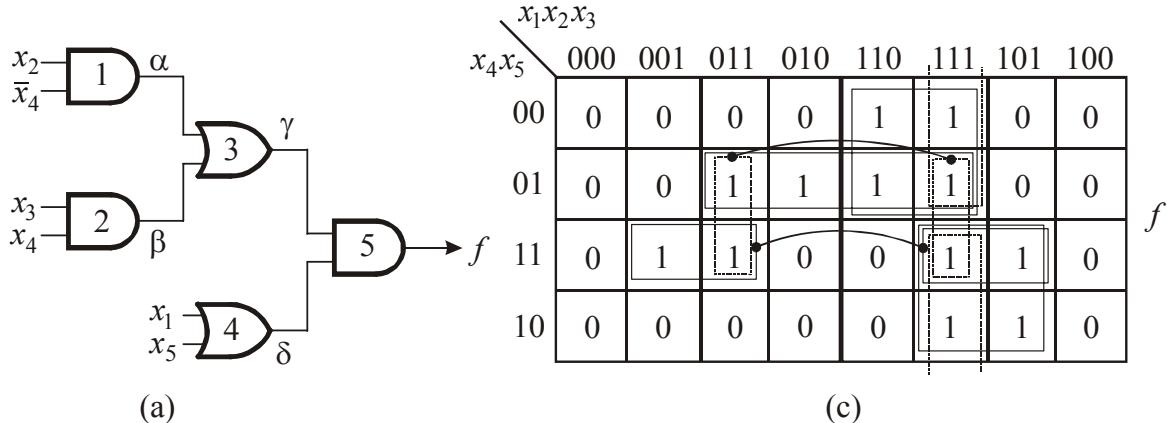


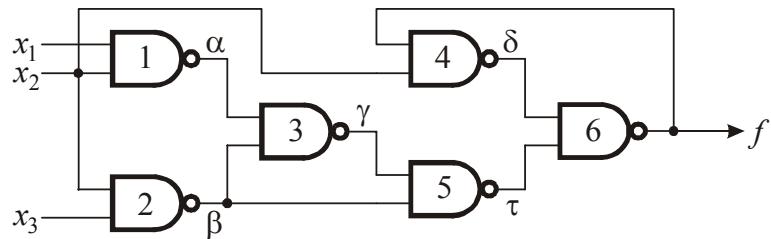
Fig.2.76 Evidențierea hazardului dinamic pentru CLC-ul din exemplul 4.

Exemplul 5. Se consideră schema logică cu 3 variabile de intrare x_1, x_2, x_3 și cu ieșirea f (v. fig. 2.77,a), realizând, cu porți logice ȘI-NU, funcția:

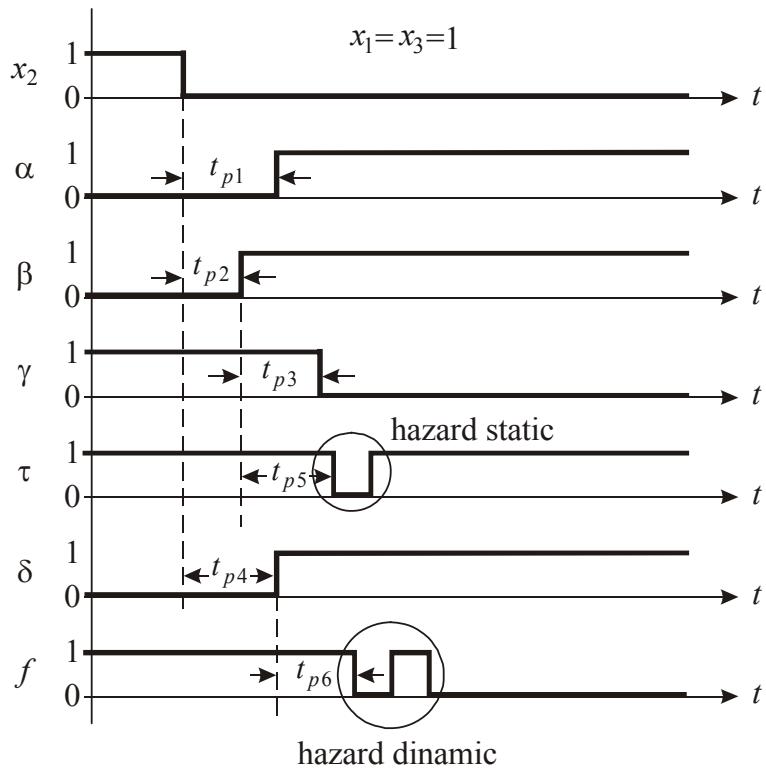
$$f(x_1, x_2, x_3) = (\bar{x}_2 + \bar{x}_3)(x_2 x_3 + x_1 x_2) + x_2 f. \quad (2.148)$$

Se presupune că starea inițială a intrărilor este $x_1 x_2 x_3 = 111$, pentru care ieșirea are valoarea $f = 1$. Se modifică $x_2 : 1 \square 0$, secvența de intrare devenind $x_1 x_2 x_3 = 101$.

Timpii de propagare prin porțile logice řI-NU 1 și řI-NU 2 sunt diferiți ($t_{p1} \neq t_{p2}$) deși, aparent, este vorba de două porți identice. Se consideră că $t_{p1} > t_{p2}$, același rezultat obținându-se și pentru cazul în care $t_{p1} < t_{p2}$.



(a)



(b)

Fig.2.77 Evidențierea hazardului dinamic pentru CLC-ul din exemplul 5.

În funcționarea circuitului vor apărea următoarele etape:

- a) $\alpha : 0 \square 1$, $\beta : 0 \square 1$, $\gamma : 1 \rightarrow 1$, $\delta : 0 \square 1$, $\tau : 1 \rightarrow 1$ și deci $f : 1 \square 0$;

b) $\gamma: 1 \square 0, \tau: 1 \square 0$ și deci $f: 0 \square 1$;

c) $\tau: 0 \square 1$ și deci $f: 1 \square 0$.

În continuare, ieșirea f rămâne în 0 logic, deoarece semnalele s-au transmis prin toate porțile schemei logice și circuitul se află în starea stabilă $f = 0$, cu $x_1x_2x_3 = 101$. Se observă că ieșirea f s-a modificat de două ori: $f: 1 \square 0 \square 1 \square 0$. Diagramele de semnal sunt prezentate în figura 2.77,b.

Hazardul poate provoca, aşa cum s-a mai menționat, erori în funcționarea circuitelor de comutare, de aceea este important să se descopere în faza de proiectare a circuitului, posibilitatea apariției hazardului și să se ia măsuri de prevenire a acestuia.

Se consideră, pentru simplificarea problemei, că nu se admite modificarea, la un moment dat, decât a unei singure variabile de intrare a circuitului. În acest caz, combinația de valori binare ale intrărilor, corespunzătoare stării acestora la un moment dat, se modifică întotdeauna astfel încât să rezulte o combinație de valori binare adiacentă cu prima.

Se poate demonstra [23], că un circuit de comutare combinatorial, căruia îi corespunde o schemă logică cu două nivele de elemente logice řI și SAU, respectiv řI-NU, nu prezintă hazard, dacă fiecare pereche de combinații de valori adiacente ale variabilelor de intrare pentru care funcția realizată are valoarea logică 1, este acoperită de către cel puțin unul dintre termenii prezenti în expresia minimă disjunctivă a funcției asociate schemei logice a circuitului dat. O combinație de valori ale variabilelor x_1, x_2, \dots, x_n , notată cu $(c_1 c_2 \dots c_n)$, unde $c_i \in \{0,1\}$, pentru $i = 0, 1, \dots, n$, este acoperită de către un termen din forma minimă disjunctivă a funcției $f(x_1, x_2, \dots, x_n)$, notat cu T , dacă termenul canonic P_j al funcției, corespondent combinației de valori $(c_1 c_2 \dots c_n)$ este inclus în termenul T , $P_j \subseteq T$.

Pentru funcția reprezentată pe diagrama Karnaugh din figura 2.73,c, perechile de combinații de valori adiacente ale variabilelor de intrare x_1 , x_2 și x_3 sunt: (110 și 100), (110 și 111), (111 și 011). Având în vedere expresia funcției $f(x_1, x_2, x_3)$, se observă că (v. ec. (2.144)):

- termenul elementar $x_1\bar{x}_3$, din expresia (2.144), acoperă perechea (110 și 100), deoarece $x_1\bar{x}_3 \supseteq x_1x_2\bar{x}_3$, respectiv $x_1\bar{x}_3 \supseteq x_1\bar{x}_2\bar{x}_3$;
- termenul elementar x_2x_3 , din expresia (2.144), acoperă perechea (111 și 011), deoarece $x_2x_3 \supseteq x_1x_2x_3$, respectiv $x_2x_3 \supseteq \bar{x}_1x_2x_3$;
- perechea (110 și 111) nu este încă acoperită, prin urmare schema poate avea hazard, cum de altfel s-a văzut.

Pentru a acoperi și această pereche, se poate adăuga în expresia funcției, implicantul prim redundant al acesteia, x_1x_2 , corespunzător conturului desenat cu linie întreruptă în figura 2.73,c, obținându-se pentru funcție următoarea expresie echivalentă cu (2.144):

$$f(x_1, x_2, x_3) = x_1\bar{x}_3 + x_2x_3 + x_1x_2 = \overline{\overline{x_1}\bar{x}_3} \cdot \overline{\overline{x_2}x_3} \cdot \overline{x_1}\overline{x_2}. \quad (2.149)$$

Schema logică asociată expresiei (2.149) este prezentată în figura 2.78,a.

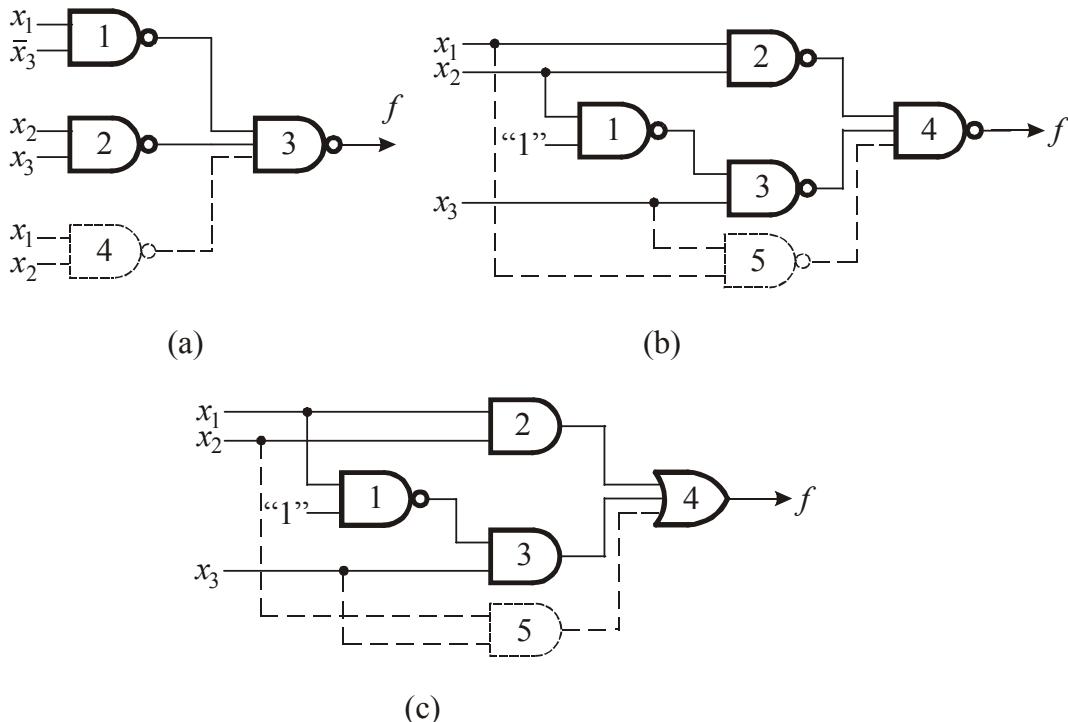


Fig.2.78 Circuite logice fără hazard, corespunzătoare exemplelor 1, 2 și 3.

Introducerea unor termeni redundanți în expresia unei funcții f , permite menținerea ieșirii f a circuitului care realizează această funcție, la valoarea dorită, pe toată durata tranziției din vechea în noua stare a semnalelor de intrare. Aceasta se poate obține printr-o întrepătrundere a mintermilor ce realizează funcția. În cazul circuitului de la exemplul 2, se introduce suplimentar termenul redundant punctat în diagrama Karnaugh (v. fig. 2.74,c) și elementul logic řI-NU 5 (v. fig. 2.78,b), obținându-se astfel funcția:

$$f(x_1, x_2, x_3) = x_1 x_2 + \bar{x}_2 x_3 + x_1 x_3 = \overline{\overline{x}_1} \cdot \overline{\overline{x}_2} \cdot \overline{\overline{x}_3}. \quad (2.150)$$

Pentru circuitul analizat în exemplul 3 este posibilă eliminarea hazardului prin introducerea în expresia logică a funcției a unui termen produs redundant format prin aplicarea operatorului řI între variabilele ce înmulțesc pe x_1 și \bar{x}_1 (în fig. 2.78,c, este implementat acest circuit). Se obține astfel expresia:

$$f(x_1, x_2, x_3) = x_1 x_2 + \bar{x}_1 x_3 + x_2 x_3, \quad (2.151)$$

care ia valoarea logică 1, când $x_2 = x_3 = 1$, evitând astfel apariția unor impulsuri parazite care, propagându-se în circuit ar determina comutări eronate.

Schema logică din figura 2.76,a, prezintă pe lângă hazardul dinamic, evidențiat cu prilejul analizei făcute și hazard static, chiar în condițiile modificării unei singure variabile de intrare la un moment dat. Se poate verifica apariția acestuia în cazul modificării stării variabilelor de intrare x_1, x_2, x_3, x_4, x_5 din 11101 în 11111 (v. fig. 2.79). Urmărind diagrama Karnaugh din figura 2.76,c, se observă că luând în expresia disjunctivă a funcției și termenii redundanți corespunzători contururilor desenate cu linie întreruptă, aceasta va avea forma:

$$f(x_1, x_2, x_3, x_4, x_5) = x_1x_2\bar{x}_4 + x_1x_3x_4 + x_2\bar{x}_4x_5 + x_3x_4x_5 + x_1x_2x_3 + x_2x_3x_5. \quad (2.152)$$

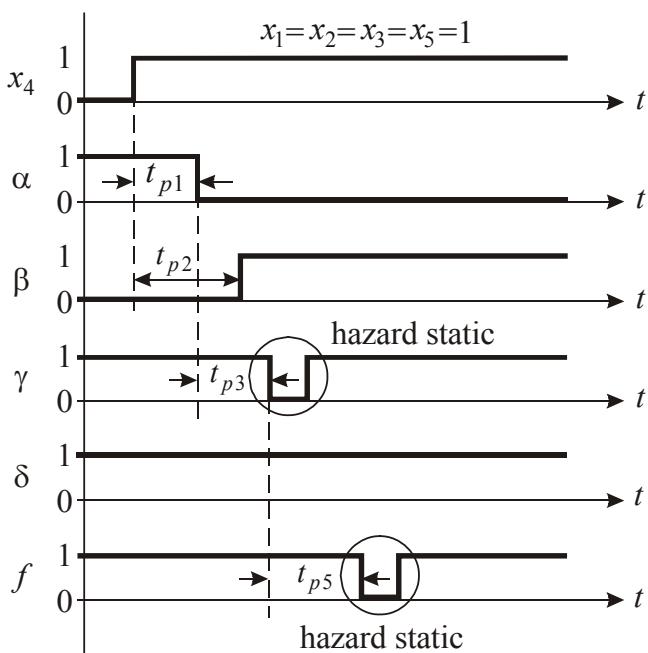


Fig.2.79 Evidențierea hazardului static la CLC-ul din exemplul 4.

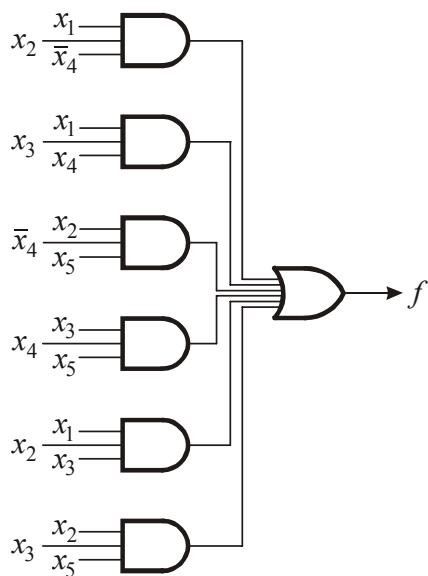


Fig.2.80 Circuit logic fără hazard.

lângă contururile corespunzătoare formei minime disjunctive a funcției și contururile care le intersectează, în cazul în care unele dintre acestea sunt disjuncte, după cum s-a văzut în cazul funcțiilor date în figurile 2.73,c, 2.74,c, 2.75,c și 2.76,c.

Proiectarea schemelor logice fără hazard, în cazul în care se modifică mai multe variabile de intrare simultan, este mai dificilă și uneori imposibilă. În acest caz, în forma disjunctivă a funcției trebuie să existe cel puțin un termen care să acopere orice pereche de combinații de valori ale variabilelor de intrare pentru care funcția ia valoarea 1. În

Termenii expresiei (2.152) acoperă toate perechile de valori adiacente ale variabilelor x_1, x_2, x_3, x_4, x_5 , pentru care funcția $f(x_1, x_2, x_3, x_4, x_5) = 1$. Prin urmare, schema logică asociată expresiei (2.152) (v. fig. 2.80), este fără hazard. Ea nu mai este însă o schemă logică optimă din punctul de vedere al costului, având mai multe elemente logice și mai multe intrări.

De regulă, introducerea termenilor redundanți nu mărește timpul de propagare intrare- ieșire, dar pe lângă creșterea complexității schemei, introducerea acestora poate genera alte fenomene de hazard. De aceea, introducerea termenilor redundanți și, în general, încercarea de a elimina hazardul se va face doar dacă fenomenul de hazard poate conduce la defecțiuni.

Condiția privind acoperirea tuturor perechilor de combinații de valori adiacente ale variabilelor de intrare pentru care funcția ia valoarea logică 1, se poate îndeplini, atunci când aceasta este reprezentată în diagrame Karnaugh, alegând astfel contururile corespunzătoare termenilor din forma disjunctivă a funcției încât să nu existe contururi disjuncte. Aceasta se realizează luând în considerare, pe

cazul funcției din figura 2.73,c, de exemplu, ar trebui să existe, în expresia funcției, câte un termen care să acopere simultan perechea (100 și 011), perechea (110 și 011), respectiv perechea (100 și 111), lucru imposibil.

De asemenea, este dificil sau chiar imposibil să se găsească, în cazul general, o schemă logică fără hazard, cu mai mult de două nivele logice. Totuși, în cazul schemelor cu mai mult de două nivele logice se poate înlătura hazardul dinamic prin aceea că se elimină hazardul static al circuitelor componente, cel puțin pentru perechile de combinații de valori adiacente ale intrărilor. De exemplu, în schema din figura 2.76,a, se poate înlătura hazardul dinamic, dacă în schema parțială cu ieșirea γ se introduce încă un element logic SI, cu intrările x_2, x_3 (vezi contururile desenate cu linie întreruptă din diagrama Karnaugh – fig. 2.76,c).

În circuitele de comutare cu două nivele de porți SAU-NU, poate să apară hazard când se modifică o combinație de valori ale intrărilor, careia îi corespunde valoarea logică 0 a funcției, într-o altă combinație de intrări căreia îi corespunde valoarea logică 0 a funcției. Dacă expresia conjunctivă a funcției se scrie plecând de la forma disjunctivă a negatei acesteia, atunci se poate proiecta un astfel de circuit fără hazard, trasând contururile din diagrama de minimizare care conține zerouri. Se alege apoi, pentru scrierea expresiei asociate schemei logice fără hazard, un număr minim de contururi disjuncte.

Exemplul 6. Se consideră funcția reprezentată pe diagrama Karnaugh din figura 2.81,a și se cere să se proiecteze un circuit de comutare, cu porți SAU-NU, care să realizeze această funcție, fără hazard. Se admite modificarea unei singure variabile de intrare la un moment dat.

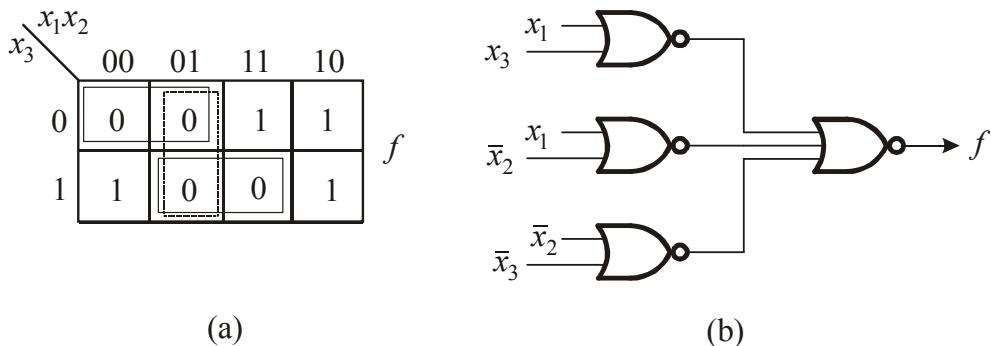


Fig.2.81 Proiectarea circuitelor realizate cu porți logice SAU-NU, fără hazard.

Grupând zerourile din diagramă se constată că trebuie luate toate cele trei contururi pentru a acoperi toate cele trei perechi de combinații de valori pentru care funcția are valoarea logică 0. Prin urmare, expresia minimă disjunctivă a negatei funcției, este:

$$\bar{f}^{\text{FMD}}(x_1, x_2, x_3) = \bar{x}_1\bar{x}_3 + \bar{x}_1x_2 + x_2x_3. \quad (2.153)$$

Din expresia (2.153) rezultă, prin negare, forma minimă conjunctivă a funcției:

$$f^{\text{FMC}}(x_1, x_2, x_3) = (x_1 + x_3)(x_1 + \bar{x}_2)(\bar{x}_2 + \bar{x}_3), \quad (2.154)$$

respectiv:

$$f^{\text{FMC}}(x_1, x_2, x_3) = \overline{x_1 + x_3} + \overline{x_1 + \bar{x}_2} + \overline{\bar{x}_2 + \bar{x}_3}. \quad (2.155)$$

Schema logică asociată expresiei (2.155) este dată în figura 2.81,b.

Eliminarea fenomenelor de hazard din schemele logice combinaționale se mai poate face și prin:

- introducerea unor elemente de întârziere în schemă, care să asigure egalizarea momentelor de apariție a semnalelor la intrarea elementelor logice. Prin aceasta, timpul general de propagare “intrare-ieșire” crește, ceea ce constituie un impediment, iar pe de altă parte, calculul întârzierilor este întotdeauna dificil, elementele logice având tempi de propagare diferiți;

- introducerea unor elemente de întârziere pasive la ieșirea circuitelor respective care să filtreze semnalele de impulsurile datorate comutărilor false. În acest mod se împiedică transferul informației false. Deși această metodă este simplă ea poate afecta viteză de acționare a întregului circuit.

Hazardul joacă un rol important, mai ales în funcționarea circuitelor de comutare combinaționale, care intră în compoziția unor circuite secvențiale asincrone.

2.5 ALTE TIPURI DE CIRCUITE LOGICE COMBINATORIALE

2.5.1 Convertoare de cod

Convertoarele de cod sunt circuite logice combinaționale având, în cazul general, n intrări și m ieșiri, care servesc la transcrierea informației dintr-un cod binar în alt cod binar [2,15,23,26].

La baza proiectării unui convertor de cod se află tabelul de corespondențe dintre cuvintele binare ale celor două coduri. Fiecare poziție din cuvântul binar al codului din care se face conversia se notează cu o variabilă; totalitatea acestor variabile reprezintă variabilele de intrare pentru circuitul combinațional. De asemenea, se notează cu o anumită variabilă fiecare poziție din codul în care se face conversia; totalitatea acestor variabile reprezintă ieșirile circuitului combinațional. Cu acestea notății suplimentare, tabelul de corespondențe, între cuvintele de cod, se transformă în tabel de adevăr pentru funcțiile realizate de circuit și exprimă dependența variabilelor de ieșire de cele de intrare. Fiecare dintre cele m poziții binare ale cuvântului de cod în care se face conversia se consideră deci ca funcții de comutare de cele n poziții binare ale cuvântului de cod care trebuie convertit. Aceste funcții, specificate prin tabelul de corespondențe/adevăr, pot fi realizate cu circuite combinaționale cu porți logice, respectiv FPLA-uri. De multe ori, în special atunci când unul dintre coduri este complet definit, este mai avantajos să se facă transformarea folosind o memorie ROM, în care la adresa reprezentată de cuvântul de cod ce trebuie convertit se află cuvântul de cod corespondent din codul în care se face conversia.

Transformarea codului binar al unui număr într-un alt cod binar, se poate face în serie, folosind circuite secvențiale, respectiv în paralel, folosind circuite combinaționale. În această secțiune se prezintă numai convertoare de cod paralele, realizate cu circuite combinaționale.

Exemplu. Se consideră un convertor de cod pentru conversia din codul binar-zecimal cu ponderile 8, 4, 2, 1, denumit cod *BCD*, în codul binar-zecimal cu ponderile 2,4,2, 1, denumit cod *Aiken* (v. Anexa C). Pozițiile cuvântului binar din codul BCD se notează cu D_3, D_2, D_1 și D_0 , iar pozițiile cuvântului binar din codul Aiken cu A_3, A_2, A_1, A_0 , corespondența între cuvintele de cod fiind dată în tabelul 2.25. Din acest tabel se pot scrie

expresiile funcțiilor $A_i(D_0, D_1, D_2, D_3)$, pentru $i = 0, 1, 2, 3$.

Tab.2.25 Tabelul de adevăr pentru convertorul de cod 8421-2421.

	Intrări 8421				Ieșiri 2421			
	D_3	D_2	D_1	D_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	1	0	0
7	0	1	1	1	1	1	0	1
8	1	0	0	0	1	1	1	0
9	1	0	0	1	1	1	1	1

Pentru minimizarea expresiilor funcțiilor A_0, A_1, A_2 și A_3 , se folosesc diagramele Karnaugh din figura 2.82. În urma minimizării, rezultă expresiile:

$$\begin{aligned} A_0 &= D_0, \\ A_1 &= D_3 + \overline{D}_2 D_1 + D_2 \overline{D}_1 D_0 = \overline{\overline{D}_3 \cdot \overline{D}_2 D_1 \cdot D_2 \overline{D}_1 D_0}, \\ A_2 &= D_3 + D_2 \overline{D}_0 + D_2 D_1 = \overline{\overline{D}_3 \cdot \overline{D}_2 \overline{D}_0 \cdot D_2 D_1}, \\ A_3 &= D_3 + D_2 D_0 + D_2 D_1 = \overline{\overline{D}_3 \cdot \overline{D}_2 D_0 \cdot \overline{D}_2 D_1}, \end{aligned} \quad (2.156)$$

care, descriu structural convertorul de cod BCD-Aiken realizat cu module logice ȘI-NU (v. fig. 2.83).

Cele mai frecvente conversii de cod întâlnite în dispozitivele pentru prelucrarea datelor numerice sunt din codul binar natural în codul binar-zecimal cu ponderile 8, 4, 2, 1 (BCD) și invers. Aceste conversii se realizează cu convertoare BINAR-BCD, respectiv BCD-BINAR.

În familia de circuite integrate există circuite convertoare BINAR-BCD și BCD-BINAR, integrate pe scară medie, de exemplu, circuitele 74184 (BCD-BINAR) și 74185 (BINAR-BCD) [34]. Circuitele menționate mai sus au la bază o memorie ROM cu o capacitate de 32 octeți (256 biți). Având în vedere faptul că cifra cea mai puțin semnificativă (c.m.p.s.) a unui cuvânt din codul binar natural este întotdeauna aceeași cu cifra cea mai puțin semnificativă din codul BCD și invers, rezultă că o astfel de memorie poate servi la conversia numerelor binare cu 6 cifre în codul BCD (o cifră rămâne nemodificată iar cu cinci poziții se pot adresa 32 locații de memorie). Cel mai mare număr binar care poate fi convertit este 63, care se codifică în codul BCD cu două cifre zecimale, prima, c.m.p.s., se codifică cu patru biți (cifra binară care nu se modifică și cele trei cifre obținute la ieșirile Y_1, Y_2 și Y_3 ale circuitului (v. fig. 2.84)), iar a doua, cifra mai semnificativă (c.m.s.) cu numai trei biți (ieșirile Y_4, Y_5 și Y_6), deoarece cifra zecimală de pondere 10^1 (c.m.s.) nu este niciodată mai mare decât sase.

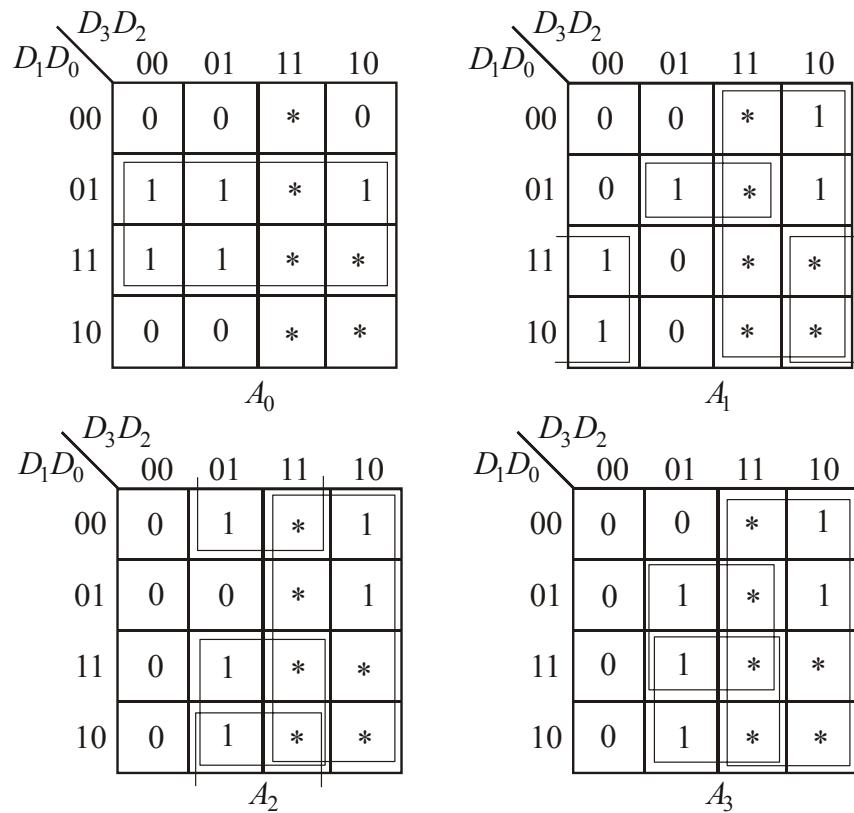


Fig.2.82 Diagramele Karnaugh pentru convertorul de cod 8421-2421.

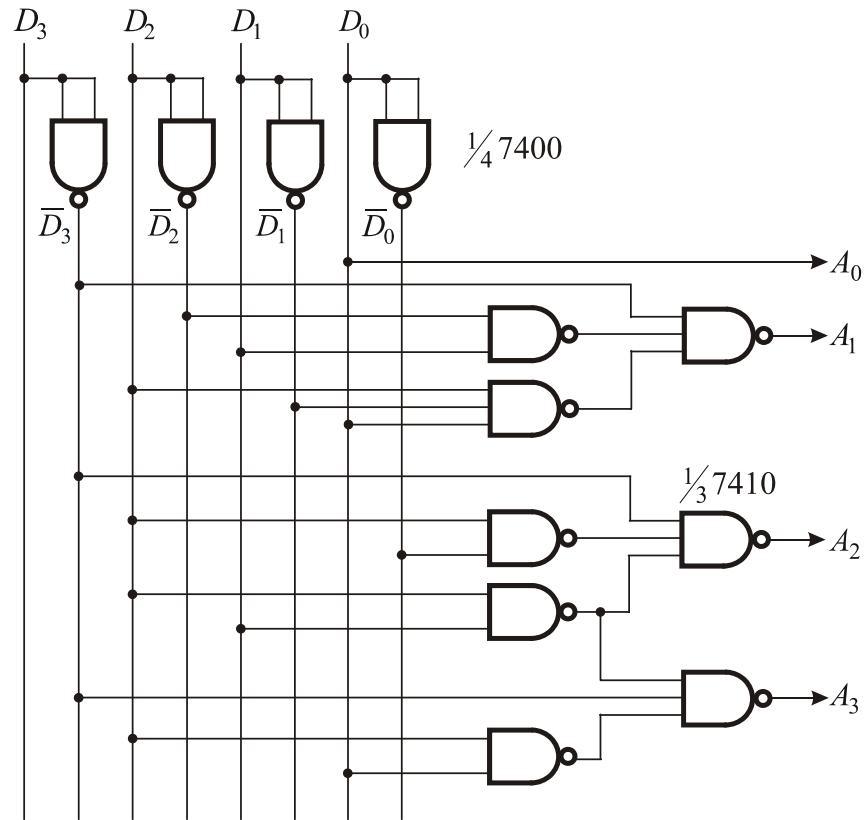


Fig.2.83 Realizarea convertorului de cod 8421-2421 cu porti logice SI-NU.

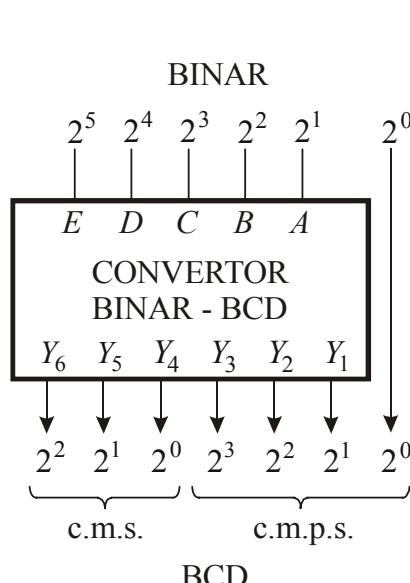


Fig.2.84 Convertor BINAR-BCD.

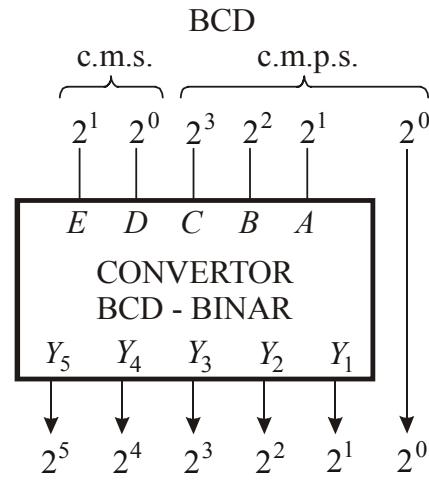


Fig.2.85 Convertor BCD-BINAR.

Corespondența între cuvintele codului binar și cele ale codului BCD sunt date în tabelul 2.26. Dacă se folosește de asemenea o memorie de 32 octeți pentru convertorul BCD-BINAR, având în vedere observația de mai sus privind ultima cifră binară din cuvântul de cod, se pot converti numere reprezentate în codul BCD din domeniul 0 la 39, deoarece rămân pentru cifra zecimală mai semnificativă numai două intrări binare (v. fig. 2.85). Corespondența între cuvintele de cod, este dată în tabelul 2.27.

Convertorul de la șase biți la BCD, generează un cuvânt de șapte biți codat BCD pornind de la un număr binar de șase biți. Pentru conversia a opt biți trebuie interconectate trei astfel de circuite iar 16 circuite sunt necesare pentru 16 biți. Extinderea structurii conduce la o schemă complexă pe mai multe nivele.

Tab.2.26 Tabelul de adevăr pentru convertorul BINAR-BCD.

	Intrări						Ieșiri						
	2^5	2^4	2^3	2^2	2^1	2^0	2^2	2^1	2^0	2^3	2^2	2^1	2^0
	E	D	C	B	A		Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0	1
...
9	0	0	1	0	0	1	0	0	0	1	0	0	1
10	0	0	1	0	1	0	0	0	1	0	0	0	0
...
59	1	1	1	0	1	1	1	0	1	1	0	0	1
60	1	1	1	1	0	0	1	1	0	0	0	0	0
61	1	1	1	1	0	1	1	1	0	0	0	0	1
62	1	1	1	1	1	0	1	1	0	0	0	1	0
63	1	1	1	1	1	1	1	1	0	0	0	1	1

Tab.2.27 Tabelul de adevăr pentru convertorul BCD-BINAR.

	Intrări						Ieșiri					
	2^1	2^0	2^3	2^2	2^1	2^0	2^5	2^4	2^3	2^2	2^1	2^0
	E	D	C	B	A		Y_5	Y_4	Y_3	Y_2	Y_1	
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	1
...
9	0	0	1	0	0	1	0	0	1	0	0	1
10	0	1	0	0	0	0	0	0	1	0	1	0
...
35	1	1	0	1	0	1	1	0	0	0	1	1
36	1	1	0	1	1	0	1	0	0	1	0	0
37	1	1	0	1	1	1	1	0	0	1	0	1
38	1	1	1	0	0	0	1	0	0	1	1	0
39	1	1	1	0	0	1	1	0	0	1	1	1

Deci pentru realizarea convertoarelor cu mai multe poziții binare, respectiv cu mai multe cifre zecimale, se folosesc mai multe circuite convertoare elementare. Schema trebuie astfel concepută încât ponderile intrărilor convertorului să fie în relația dată în tabelul 2.26, respectiv 2.27.

Exemplu. Se cere proiectarea schemei logice a unui convertor BINAR-BCD, care să convertească numerele binare cu opt cifre.

Cel mai mare număr binar de 8 cifre este 255, prin urmare se obțin, la ieșire, trei cifre zecimale din care ultima, c.m.s., codificată cu 2 cifre binare. Sunt necesare, prin urmare, mai multe convertoare BINAR-BCD pentru șase cifre binare. O soluție de rezolvare, cu trei convertoare elementare, este prezentată în figura 2.86.

În această figură s-au notat ponderile cifrelor binare obținute după fiecare nivel de conversie, pentru a se justifica posibilitatea conectării convertorului următor. Astfel, primul convertor transformă numărul binar ($B_7B_6B_5B_4B_3B_2$), având valoarea maximă 63, într-un cod BCD cu două cifre zecimale, prima dintre ele, codificată cu patru cifre binare ($C_3C_2C_1B_2$), de pondere 10^0 , iar următoarea cu trei cifre binare ($D_2D_1D_0$), de pondere 10^1 . Numărul binar ($C_3C_2C_1B_2B_1B_0$), având valoarea maximă $(100111)_2 = 39$, este codificat de către al doilea convertor, în codul BCD, cu două cifre zecimale, ($E_3E_2E_1B_0$), de pondere 10^0 și (F_1F_0), de pondere 10^1 . Numărul binar ($0D_2D_1D_0F_1F_0$), având valoarea maximă $(011011)_2 = 27$, este codificat în codul BCD cu următoarele două cifre zecimale: ($G_3G_2G_1F_0$) care are ponderea 10^1 și (G_5G_4), care are ponderea 10^2 , de către al treilea convertor. În acest fel se obțin cele trei cifre zecimale dorite.

Conversia din codul BCD în codul binar natural se face în mod asemănător.

Exemplu. Se cere conversia codului BCD al unui număr zecimal cu două cifre în codul binar natural.

Deoarece un convertor elementar nu poate să cuprindă decât două dintre pozițiile

binare ale cifrei zecimale mai semnificative din codul BCD, sunt necesare două convertoare conectate în cascadă (v. fig. 2.87). Se vede, din figură, că prin modul de conectare al celor de-al doilea convertor s-a asigurat ca ponderea cifrelor aplicate la intrările D și E să fie mai mare de zece ori decât ponderea cifrelor corespondente din codul cifrei zecimale mai puțin semnificative.

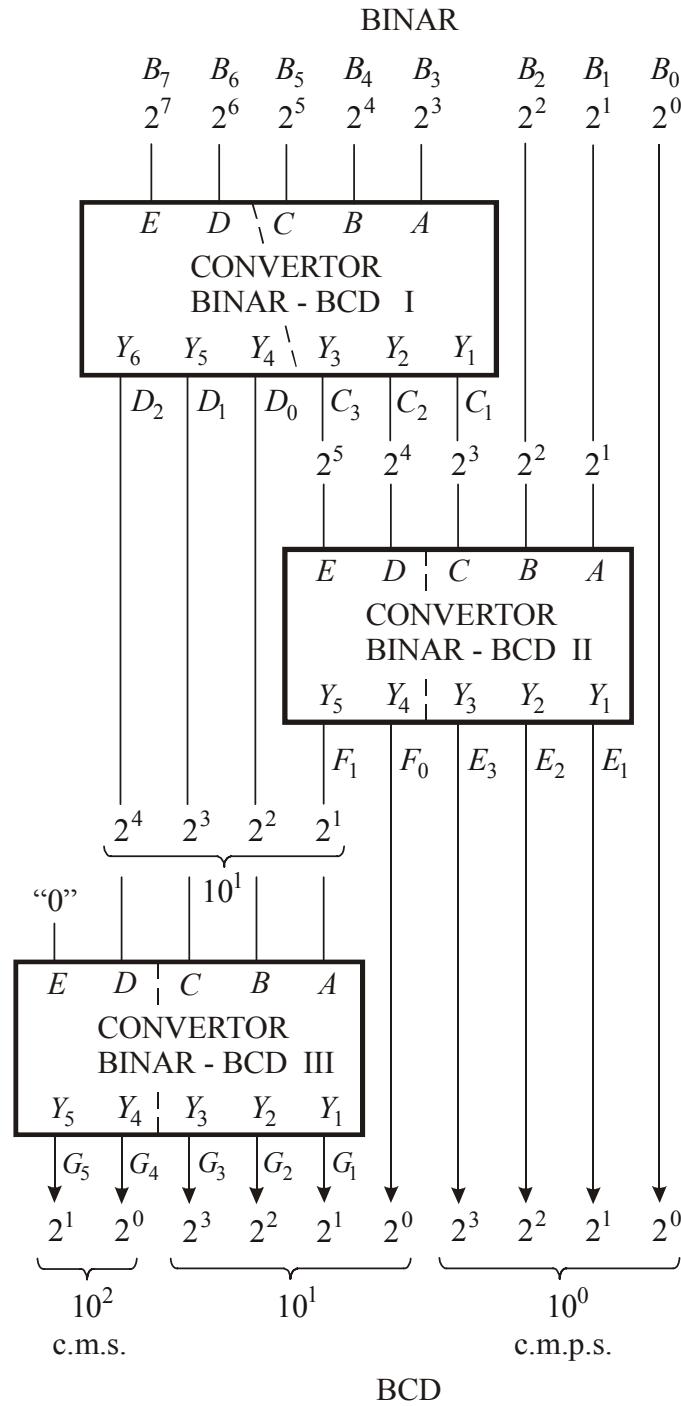


Fig.2.86 Convertor BINAR-BCD.

Convertorul din codul BCD în codul binar natural de şase biți reprezintă jumătate din schema necesară convertirii a două numere de 4 biți codate în BCD într-un număr

binar de şapte biți. Prin interconectarea a şase astfel de circuite se obține conversia a trei numere BCD la un număr binar de nouă biți. Fiecare extindere peste acest nivel presupune dublarea numărului circuitelor folosite.

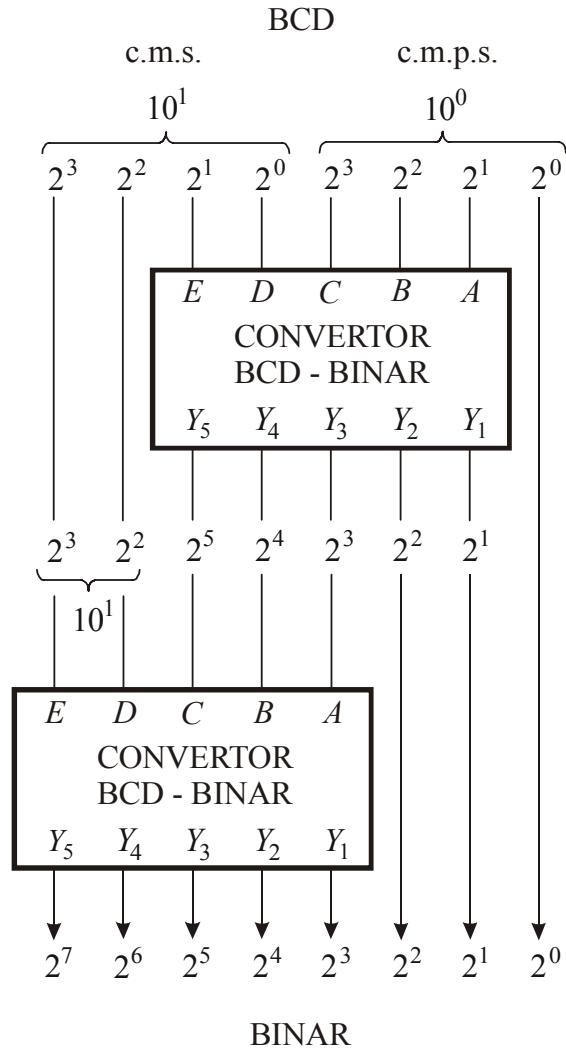


Fig.2.87 Convertor BCD-BINAR.

2.5.2 Decodificatorul BCD-7 segmente

Decodificatorul *BCD-7 segmente* este un circuit de decodificare utilizat la comanda sistemelor de afişare numerică realizate din şapte segmente luminoase: becuri, diode electroluminiscente sau cristale lichide. Unii autori (de exemplu, [27]) consideră că denumirea de decodificator este utilizată impropriu și pentru convertorul de cod BCD-7 segmente.

Decodificatorul studiat este un CLC cu patru intrări, notate cu *D*, *C*, *B*, *A* și şapte ieșiri notate cu *a*, *b*, *c*, *d*, *e*, *f*, *g* (v. fig. 2.88,a) [2,15,26,27]. Tabelul de adevăr al decodificatorului este reprezentat în tabelul 2.28.

Pe cele patru intrări ale circuitului se aplică cuvântul de cod (codul BCD) reprezentând cifra zecimală ce urmează a fi afişată. Cele şapte ieșiri comandă segmentele corespunzătoare, dispuse ca în figura 2.88,b. Cele şapte segmente se consideră inițial stinse. Pentru a se produce cifra zecimală dorită, diferite elemente ale afișajului sunt aprinse.

Pe baza tabelului de adevăr (v. tab. 2.28) se pot completa matricele Karnaugh ale celor șapte segmente (v. fig. 2.89).

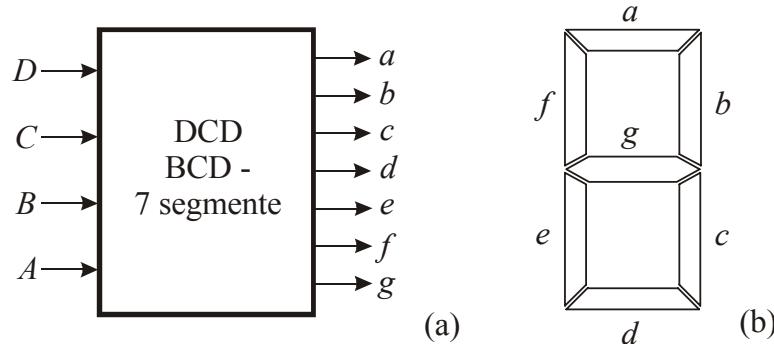


Fig.2.88 Decodificator BCD-7 segmente: (a) schema-bloc; (b) dispozitivul segmentelor.

Tab.2.28 Tabelul de adevăr pentru un DCD BCD-7 segmente.

	Intrări				Ieșiri						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

0 – segment stins, 1 – segment aprins.

Grupând în mod convenabil compartimentele notate cu 1, din aceste matrice, se obțin expresiile logice ale funcțiilor de ieșire care permit întocmirea schemei logice a decodificatorului:

$$\begin{aligned}
 a &= \overline{\overline{D} + B + CA + \overline{C} \overline{A}} = \overline{\overline{D} \cdot \overline{B} \cdot \overline{CA} \cdot \overline{\overline{C} \overline{A}}}, \\
 b &= \overline{\overline{C} + \overline{B} \overline{A} + BA} = \overline{C \cdot \overline{B} \overline{A} \cdot \overline{BA}}, \\
 c &= \overline{\overline{C} + \overline{B} + A} = \overline{\overline{C} \cdot \overline{B} \cdot \overline{A}}, \\
 d &= \overline{\overline{D} + BA + \overline{CB} + \overline{C} \overline{A} + C\overline{BA}} = \overline{\overline{D} \cdot \overline{BA} \cdot \overline{CB} \cdot \overline{C} \overline{A} \cdot \overline{C\overline{BA}}}, \\
 e &= \overline{\overline{C} \overline{A} + \overline{BA}} = \overline{\overline{C} \overline{A} \cdot \overline{BA}}, \\
 f &= \overline{\overline{D} + \overline{B} \overline{A} + CA + \overline{CB}} = \overline{\overline{D} \cdot \overline{B} \overline{A} \cdot CA \cdot \overline{CB}}, \\
 g &= \overline{\overline{D} + CB + \overline{CB} + \overline{BA}} = \overline{\overline{D} \cdot CB \cdot \overline{CB} \cdot \overline{BA}}.
 \end{aligned} \tag{2.157}$$

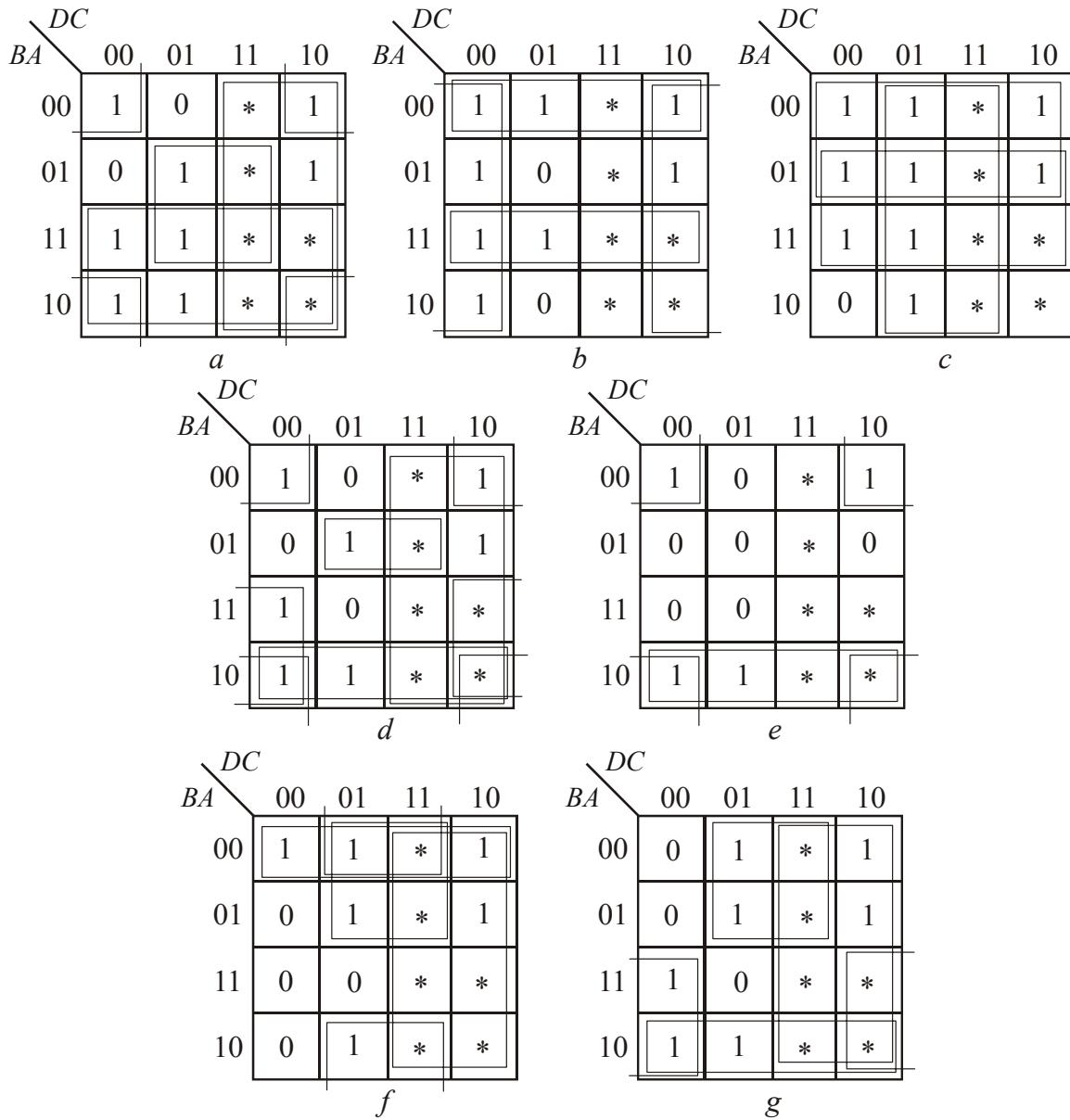


Fig.2.89 Matricele Karnaugh ale decodificatorului BCD-7 segmente.

Se observă, din diagramele Karnaugh, că dacă s-ar considera compartimentele noteate cu zero (deci negatele funcțiilor de ieșire) s-ar obține niște expresii mai simple.

În figura 2.90 este reprezentată schema logică parțială a decodificatorului BCD-7 segmente. În [34] sunt prezentate patru decodificatoare BCD-7 segmente integrate (7446, 7447, 7448, 7449), realizate în diverse variante constructive.

2.5.3 Circuite codificatoare

Circuitul codificator (CD) realizează funcția inversă aceleia a unui decodificator. Codificatorul este un circuit de comutare combinațional având, în cazul general, m intrări X_0, X_1, \dots, X_{m-1} și n ieșiri ($m \leq 2^n$), Z_0, Z_1, \dots, Z_{n-1} (v. fig. 2.91,a) [23,26,27,30]. La aplicarea unui semnal logic activ pe o intrare se obține, la ieșire, un cuvânt de n biți, care reprezintă codul intrării activate.

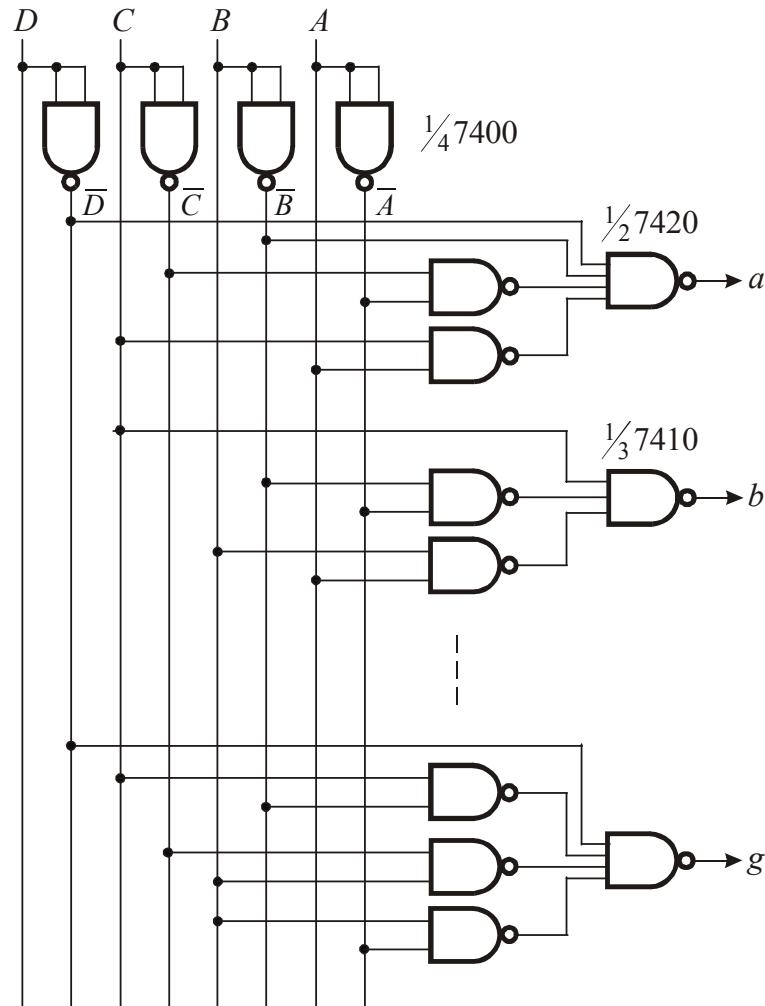


Fig.2.90 Schema logică a decodificatorului BCD-7 segmente.

Acest circuit furnizează deci la ieșire un cuvânt binar de n biți atunci când numai una dintre cele m intrări ale sale este activată. Numărul cuvintelor generate la ieșirea CD-ului este egal cu numărul intrărilor (liniile de cuvânt). Cele m cuvinte de ieșire sunt, în general, dar nu necesar, distințe. Între variabilele de intrare, X și funcțiile de ieșire, Z , se poate scrie următoarea relație:

$$Z_r = \sum_0^{m-1} a_i X_i , \quad (2.158)$$

în care $r = 0,1,\dots,(n-1)$, iar a_i este un factor ce poate lua valorile 1 sau 0 ($a_i \in \{0,1\}$). Relația (2.158) indică posibilitatea realizării fizice a CD-ului folosind porți SAU.

Circuitele codificatoare cu multe intrări se realizează, de obicei, cu circuite elementare MSI prevăzute cu posibilitatea de legare în cascadă.

În continuare va fi examinat codificatorul ZECIMAL-BCD, cu $m=10$ intrări și $n=4$ ieșiri, a cărui schemă-bloc este prezentată în figura 2.91,b și c (se va studia atât cazul în care intrările și ieșirile CD-ului sunt active pe nivelul 1 logic cât și cazul în care acestea sunt active pe nivelul 0 logic). În tabelele de adevar 2.29,a,b se indică posibilitatea codificării cifrelor zecimale 0÷9, în cod binar.

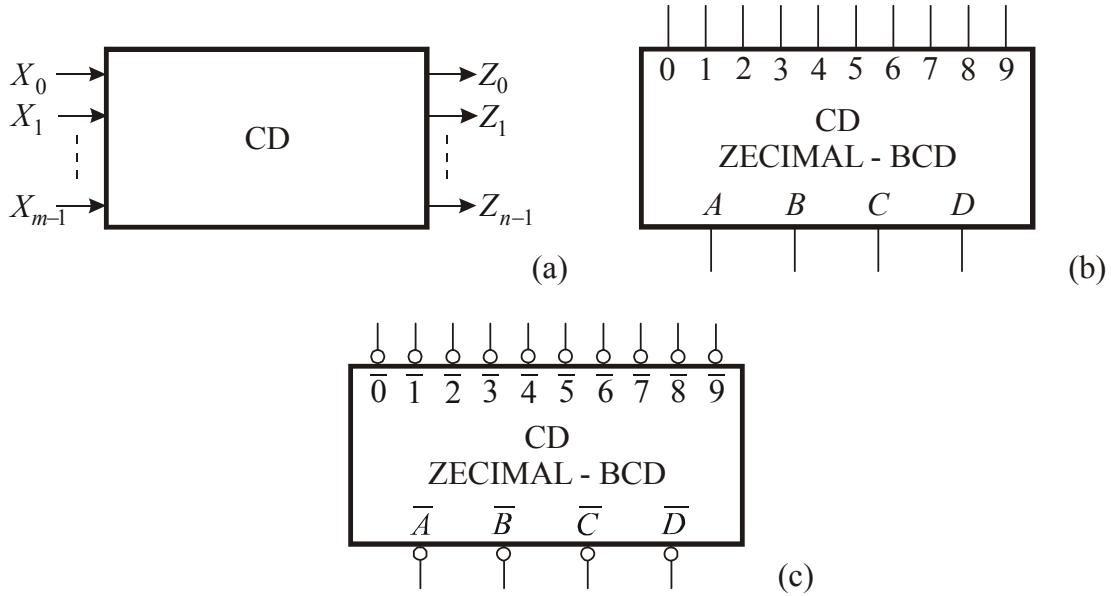


Fig.2.91 Schemele-bloc pentru un CD cu m intrări și n ieșiri (a), respectiv pentru un CD ZECIMAL-BCD cu intrările și ieșirile active pe 1 logic (b) sau pe 0 logic (c).

Tab.2.29 Tabele de adevăr pentru un CD ZECIMAL-BCD.

	Intrări										Ieșiri			
	0	1	2	3	4	5	6	7	8	9	D	C	B	A
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	1	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	0	0	0	0	1	0	1
6	0	0	0	0	0	0	1	0	0	0	0	1	1	0
7	0	0	0	0	0	0	0	1	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	1	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1	0	0	1

(a)

	Intrări										Ieșiri			
	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	$\bar{6}$	$\bar{7}$	$\bar{8}$	$\bar{9}$	\bar{D}	\bar{C}	\bar{B}	\bar{A}
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
2	1	1	0	1	1	1	1	1	1	1	1	1	0	1
3	1	1	1	0	1	1	1	1	1	1	1	1	0	0
4	1	1	1	1	0	1	1	1	1	1	1	0	1	1
5	1	1	1	1	1	0	1	1	1	1	1	0	1	0
6	1	1	1	1	1	1	0	1	1	1	1	0	0	1
7	1	1	1	1	1	1	1	0	1	1	1	0	0	0
8	1	1	1	1	1	1	1	1	0	1	0	1	1	1
9	1	1	1	1	1	1	1	1	1	0	0	1	1	0

(b)

Pe baza acestor tabele se pot deduce următoarele expresii ale funcțiilor de ieșire:

$$\begin{aligned} A &= 1 + 3 + 5 + 7 + 9, \quad B = 2 + 3 + 6 + 7, \\ C &= 4 + 5 + 6 + 7, \quad D = 8 + 9, \end{aligned} \quad (2.159)$$

respectiv:

$$\begin{aligned} \bar{A} &= \bar{1} \cdot \bar{3} \cdot \bar{5} \cdot \bar{7} \cdot \bar{9}, \quad \bar{B} = \bar{2} \cdot \bar{3} \cdot \bar{6} \cdot \bar{7}, \\ \bar{C} &= \bar{4} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7}, \quad \bar{D} = \bar{8} \cdot \bar{9}. \end{aligned} \quad (2.160)$$

Circuitul poate fi realizat cu porți logice SAU având un număr maxim de intrări egal cu 5 (v. fig. 2.92).

Uzual, CD-urile constituie subsisteme ale unor circuite MSI sau LSI (convertoare de cod, ROM-uri, PLA-uri). Se produc însă și circuite integrate având funcția de codificare, ca, de exemplu, 74147 [34] (CD-ul ZECIMAL-BCD, cu 9 intrări și 4 ieșiri toate active pe nivelul 0 logic). La acest circuit, cifra zecimală 0 se codifică implicit când nu este activă nici una dintre cele nouă intrări ale circuitului.

Funcționarea corectă a circuitului codificator presupune, aşa cum s-a precizat mai sus, ca, la un moment dat, să fie activă o singură intrare. Deoarece această condiție nu este respectată întotdeauna, în practică se iau măsuri ca atunci când sunt active mai multe intrări simultan, să se codifice acea intrare căreia i s-a atribuit o prioritate mai mare.

• Codificatorul priorităr

Codificatoarele prioritare, spre deosebire de cele “clasice”, sunt astfel concepute încât, în cazul activării simultane a mai multor intrări, la ieșire să apară doar codul intrării cu prioritatea cea mai mare [2,15,23,27,28,29,30]. În acest scop fiecarei intrări îi este atribuită o prioritate, care crește cu numărul intrării.

Pentru exemplificare se va prezenta circuitul integrat – codificator priorităr 74148 [34] a cărui schemă-bloc este dată în figura 2.93 și a cărui logică de funcționare este redată în tabelul 2.30.

Circuitul dispune de opt intrări informaționale, $\bar{0}, \bar{1}, \dots, \bar{7}$, a căror prioritate crește cu valoarea numărului și de trei ieșiri $\bar{A}_0, \bar{A}_1, \bar{A}_2$, toate active pe 0 logic. Din tabelul de adevăr al circuitului se poate constata că prin activarea, de exemplu, a intrării $\bar{4}$ (linia

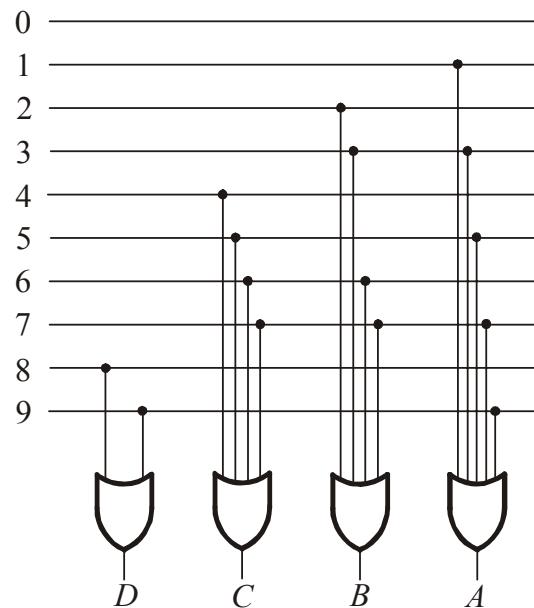


Fig.2.92 Schema logică a CD-ului ZECIMAL-BCD.

0111), simultan cu una sau mai multe dintre intrările $\bar{0}, \bar{1}, \bar{2}, \bar{3}$, numai ieșirea \bar{A}_2 este activată, iar ieșirile \bar{A}_0 și \bar{A}_1 sunt forțate în 1 logic, situație care corespunde intrării având prioritarea cea mai mare, 4. Circuitul prezintă, de asemenea, o intrare de validare \bar{EI} și încă două ieșiri suplimentare \bar{EO} și \bar{GS} .

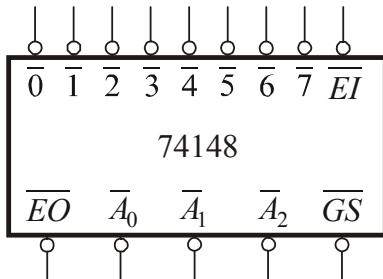


Fig.2.93 Schema-bloc a CD-ului prioritatar 74148.

Când $\bar{EI} = 1$ toate intrările sunt blocate și toate ieșirile sunt menținute în 1 logic (inactive). Ieșirea \bar{GS} devine activă (0), atunci când cel puțin una dintre intrările de date este activată, iar ieșirea \bar{EO} este activată (0) atunci când toate intrările de date sunt inactive (1). Ieșirea \bar{EO} se utilizează în cazul conectării în cascadă a mai multor codificatoare, pentru validarea unui circuit similar, având intrări de date cu prioritate mai mică.

Tab.2.30 Tabelul de adevăr al CD-ului prioritatar 74148.

\bar{EI}	Intrări							Ieșiri					Observații	
	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	$\bar{6}$	$\bar{7}$	\bar{A}_2	\bar{A}_1	\bar{A}_0	\bar{GS}	\bar{EO}	
1	*	*	*	*	*	*	*	*	1	1	1	1	1	Blocat
0	1	1	1	1	1	1	1	1	1	1	1	1	0	Intrări inactive
0	*	*	*	*	*	*	*	0	0	0	0	0	1	
0	*	*	*	*	*	*	0	1	0	0	1	0	1	
0	*	*	*	*	*	0	1	1	0	1	0	0	1	
0	*	*	*	*	0	1	1	1	0	1	1	0	1	
0	*	*	*	0	1	1	1	1	1	0	0	0	1	
0	*	*	0	1	1	1	1	1	1	0	1	0	1	
0	*	0	1	1	1	1	1	1	1	1	0	0	1	
0	0	1	1	1	1	1	1	1	1	1	1	0	1	

Din tabelul 2.30 se pot obține expresiile funcțiilor de ieșire. Pentru \bar{A}_0, \bar{A}_1 și \bar{A}_2 rezultă următoarele expresii:

$$\begin{aligned}\bar{A}_2 &= \bar{EI} \cdot \bar{7} + \bar{EI} \cdot \bar{6} \cdot \bar{7} + \bar{EI} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7} + \bar{EI} \cdot \bar{4} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7}, \\ \bar{A}_1 &= \bar{EI} \cdot \bar{7} + \bar{EI} \cdot \bar{6} \cdot \bar{7} + \bar{EI} \cdot \bar{3} \cdot \bar{4} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7} + \bar{EI} \cdot \bar{2} \cdot \bar{3} \cdot \bar{4} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7}, \\ \bar{A}_0 &= \bar{EI} \cdot \bar{7} + \bar{EI} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7} + \bar{EI} \cdot \bar{3} \cdot \bar{4} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7} + \bar{EI} \cdot \bar{1} \cdot \bar{2} \cdot \bar{3} \cdot \bar{4} \cdot \bar{5} \cdot \bar{6} \cdot \bar{7}.\end{aligned}\quad (2.161)$$

Se observă că, utilizând teorema: $ab + \bar{a}c = ab + \bar{a}c + bc$, se pot simplifica o serie de termeni din expresiile acestor funcții, rezultând:

$$\begin{aligned}\bar{A}_2 &= \bar{EI} \cdot \bar{7} + \bar{EI} \cdot \bar{6} + \bar{EI} \cdot \bar{5} + \bar{EI} \cdot \bar{4}, \\ \bar{A}_1 &= \bar{EI} \cdot \bar{7} + \bar{EI} \cdot \bar{6} + \bar{EI} \cdot \bar{3} \cdot \bar{4} \cdot \bar{5} + \bar{EI} \cdot \bar{2} \cdot \bar{4} \cdot \bar{5}, \\ \bar{A}_0 &= \bar{EI} \cdot \bar{7} + \bar{EI} \cdot \bar{5} \cdot \bar{6} + \bar{EI} \cdot \bar{3} \cdot \bar{4} \cdot \bar{6} + \bar{EI} \cdot \bar{1} \cdot \bar{2} \cdot \bar{4} \cdot \bar{6}.\end{aligned}\quad (2.162)$$

Funcția \overline{GS} se exprimă mai ușor sub formă adevărată:

$$GS = \overline{EO} + EI = \overline{EO} \cdot \overline{EI}, \quad (2.163)$$

unde:

$$EO = \overline{EI} \cdot \overline{0} \cdot \overline{1} \cdot \overline{2} \cdot \overline{3} \cdot \overline{4} \cdot \overline{5} \cdot \overline{6} \cdot \overline{7}. \quad (2.164)$$

Schema logică a CD-ului priorităr 74148 este prezentată în figura 2.94.

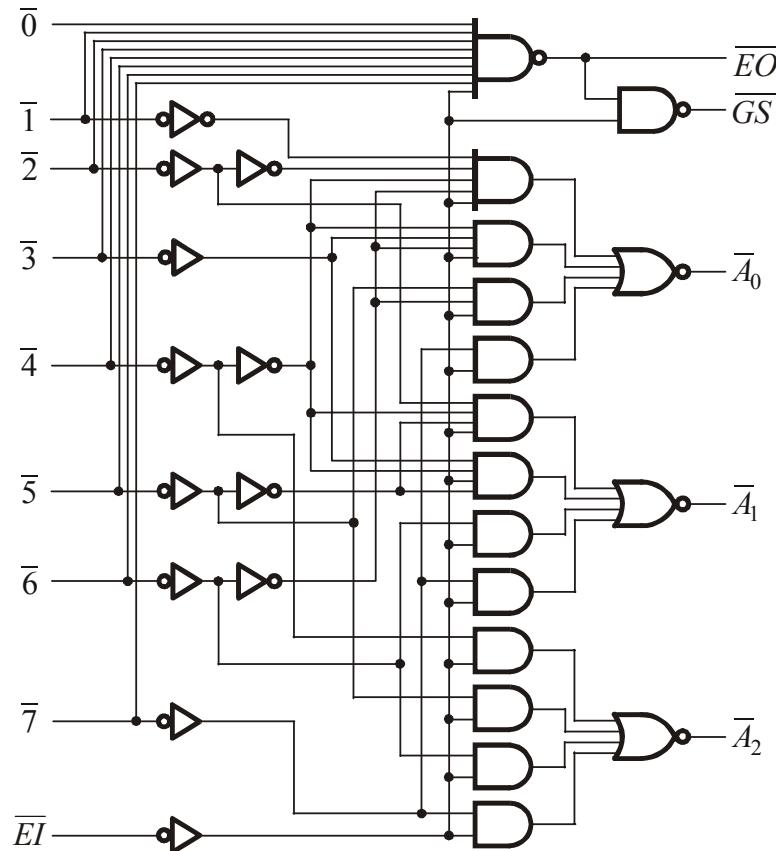


Fig.2.94 Schema logică a CD-ului priorităr 74148.

Codificatorul priorităr are, aşa cum s-a precizat, intrări și ieșiri de selecție ce permit cascadarea acestora pentru obținerea unui număr mai mare de intrări. Aceste scheme de extindere necesită, însă, porți logice adiționale pentru generarea bițiilor de ieșire. Spre exemplu, pentru un număr de 16 intrări sunt necesare două codificatoare și patru porți řI-NU cu două intrări, iar pentru 32 de intrări și cinci ieșiri se folosesc patru codificatoare, s.a.m.d.

În figura 2.95 este prezentată schema logică a unui CD priorităr, utilizat pentru codificarea în binar a numerelor zecimală de la 0 la 15, realizat cu două circuite integrate 74148. Se constată că la activarea uneia sau mai multora dintre liniile $\overline{0}, \overline{1}, \dots, \overline{7}$, primul codificator este validat prin $\overline{EO}_2(0)$, iar atunci când se activează intrările $\overline{8}, \overline{9}, \dots, \overline{15}$, funcționează doar al doilea codificator, \overline{EO}_2 fiind pe 1 logic.

Circuitul din dreapta având prioritate față de cel din stânga, autorizează pe ultimul

și îi cedează prioritatea numai atunci când nici una dintre intrările sale, $\bar{8}$ la $\bar{15}$, nu este activă (nu este în starea logică 0). Ieșirea \overline{EO}_2 reprezintă deci și cifra cea mai semnificativă din cuvântul de cod, A_3 . Partea mai puțin semnificativă a cuvântului de cod, $A_2A_1A_0$, este dată de către circuitul care deține prioritatea în momentul considerat. Selectarea grupului este indicată de semnalul P . Cele patru ieșiri, active pe 1 logic, se obțin așa cum se indică în figură și în conformitate cu relațiile:

$$\begin{aligned} A_0 &= \overline{\bar{A}_0^1 \cdot \bar{A}_0^2} = A_0^1 + A_0^2, \quad A_1 = \overline{\bar{A}_1^1 \cdot \bar{A}_1^2} = A_1^1 + A_1^2, \\ A_2 &= \overline{\bar{A}_2^1 \cdot \bar{A}_2^2} = A_2^1 + A_2^2, \quad A_3 = \overline{EO}_2. \end{aligned} \quad (2.165)$$

Codul binar de la ieșire este dat, așa cum s-a precizat, de secvența $A_3A_2A_1A_0$. Valoarea bițiilor A_0, A_1, A_2, A_3 , funcție de starea activă a liniilor de intrare $\bar{0}, \bar{1}, \dots, \bar{14}, \bar{15}$ și de valoarea bițiilor A_i^j (unde $i = 0, 1, 2$, iar $j = 1, 2$), este prezentată centralizat în tabelul 2.31.

Tab.2.31 Situația intrărilor și ieșirilor CD-ului prioritari din figura 2.95.

$\bar{15}$	$\bar{14}$	$\bar{13}$	$\bar{12}$	$\bar{11}$	$\bar{10}$	$\bar{9}$	$\bar{8}$	$\bar{7}$	$\bar{6}$	$\bar{5}$	$\bar{4}$	$\bar{3}$	$\bar{2}$	$\bar{1}$	$\bar{0}$	A_3	A_2	A_1	A_0
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	A_2^1	A_1^1	A_0^1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	A_2^2	A_1^2	A_0^2

Valorile bițiilor \bar{A}_i^j ($\bar{A}_0^1, \bar{A}_1^1, \bar{A}_2^1$, respectiv $\bar{A}_0^2, \bar{A}_1^2, \bar{A}_2^2$) se pot lua din tabelul 2.30.

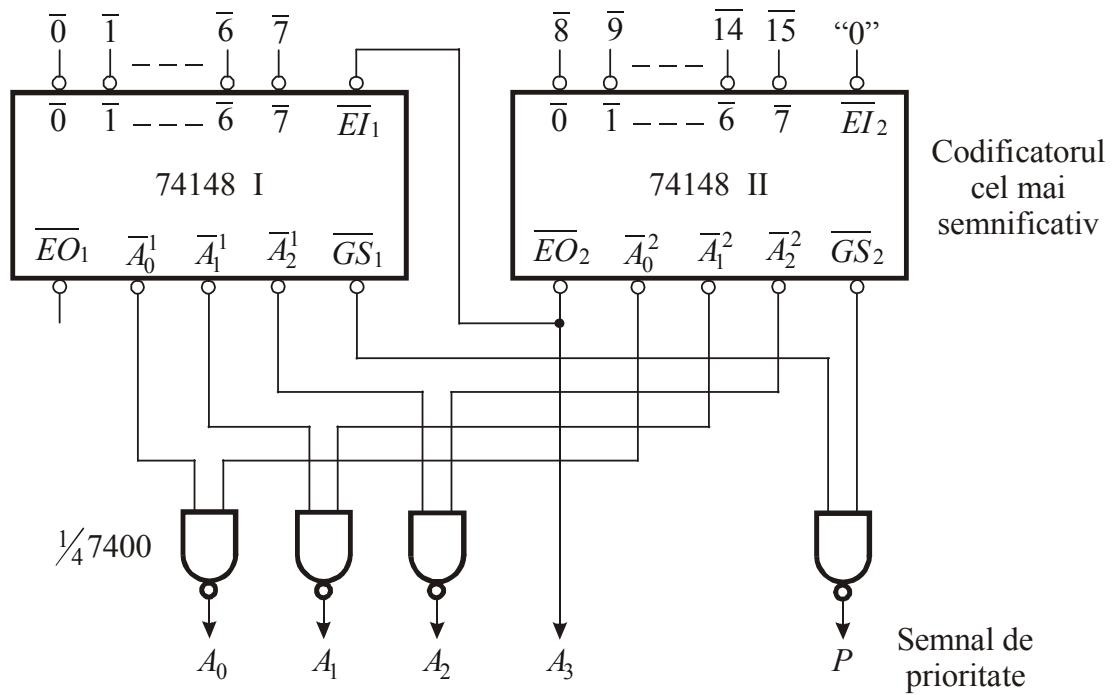


Fig.2.95 Circuit codificator pentru codificarea cu prioritate a 16 semnale de intrare.

2.5.4 Comparatoare numerice

Comparatoarele numerice sunt circuite logice combinaționale care permit determinarea valorii relative a două numere binare [2,15,16,22,23,26,27,30]. Acest tip de circuite realizează compararea în paralel (în același timp), a cifrelor corespondente a două numere binare, A și B , cu n poziții, în vederea stabilirii relației dintre cele două numere, adică pentru a vedea dacă $A < B$, $A = B$, respectiv $A > B$. Cifrele celor două numere binare se aplică la intrările circuitului, iar relația dintre numere este indicată de starea logică a celor trei ieșiri ale comparatorului, notate cu Z_1 , Z_2 și Z_3 . Astfel, dacă ieșirea Z_1 este în starea 1, rezultă că $A < B$, dacă ieșirea Z_2 este în starea 1, rezultă că $A = B$, iar dacă ieșirea Z_3 este în starea 1, rezultă că $A > B$. Ca exemplu, se consideră un comparator pentru numere binare cu 4 cifre, $A(A_3, A_2, A_1, A_0)$ și $B(B_3, B_2, B_1, B_0)$ (v. fig. 2.96). Cifrele cele mai puțin semnificative sunt A_0 și B_0 . Inspectarea cifrelor corespondente ale numerelor, în vederea stabilirii relației dintre ele, se face începând cu cele mai semnificative, în acest caz particular începând cu A_3 și B_3 .

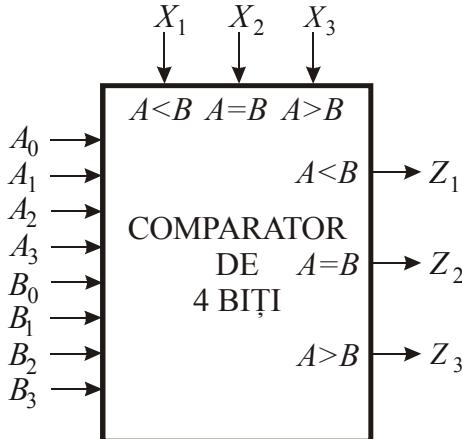


Fig.2.96 Schema-bloc a unui comparator de 4 biți.

Pentru a se obține expresiile algebrice care definesc relațiile între două numere binare, se consideră, pentru început, un comparator de un bit. Funcționarea acestui circuit este descrisă în tabelul 2.32, din care se pot scrie:

$$f_1^{A_i < B_i} = \overline{A_i} B_i, \quad (2.166)$$

$$f_2^{A_i = B_i} = \overline{A_i} \overline{B_i} + A_i B_i = \overline{A_i \oplus B_i} = C_i, \quad (2.167)$$

$$f_3^{A_i > B_i} = A_i \overline{B_i}. \quad (2.168)$$

Schema logică a comparatorului de un bit este reprezentată în figura 2.97. Acest circuit constituie celula de bază a comparatorului pentru numere cu mai mulți biți.

Astfel, prin interconectarea a două comparatoare de câte un bit se poate realiza un comparator de doi biți, aşa cum este arătat în figura 2.98. Cele două numere binare se scriu:

$$A = A_1 \cdot 2^1 + A_0 \cdot 2^0, \quad B = B_1 \cdot 2^1 + B_0 \cdot 2^0. \quad (2.169)$$

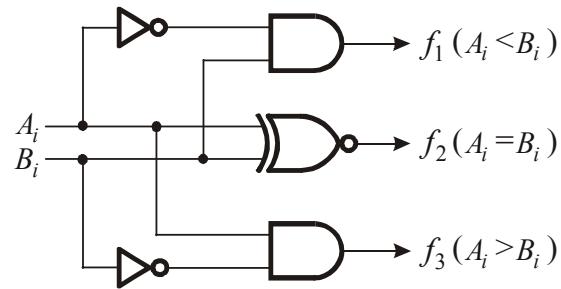


Fig.2.97 Schema logică a comparatorului de un bit.

Tab.2.32 Tabelul de adevăr al unui comparator de un bit.

A_i	B_i	f_1 ($A_i < B_i$)	f_2 ($A_i = B_i$)	f_3 ($A_i > B_i$)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Procesul de comparare începe, aşa cum s-a menţionat, cu compararea biţilor celor mai semnificativi, A_1 cu B_1 . Dacă între aceştia există relaţia $A_1 > B_1$ sau $A_1 < B_1$ acest lucru implică şi faptul că $A > B$ sau $A < B$, indiferent de valoarea biţilor A_0 şi B_0 . În schemă acest lucru se materializează prin legarea ieşirilor portilor 1 respectiv 2, direct la intrările portilor 5 respectiv 6.

Dacă $A_1 = B_1$, pentru determinarea relaţiei dintre numerele A şi B , se impune şi examinarea biţilor A_0 şi B_0 , situaţie în care ieşirea portului 8 asigură deschiderea portilor 3 şi 4.

Dacă $A_0 < B_0$ respectiv $A_0 > B_0$ rezultă că sunt valabile şi relaţiile $A < B$ respectiv $A > B$, ceea ce se obţine prin legarea ieşirilor portilor 3, 4 la intrările portilor 5 respectiv 6. În cazul în care $A_1 = B_1$ şi $A_0 = B_0$ rezultă că cele două numere sunt egale, $A = B$ şi se activează ieşirea portului 7.

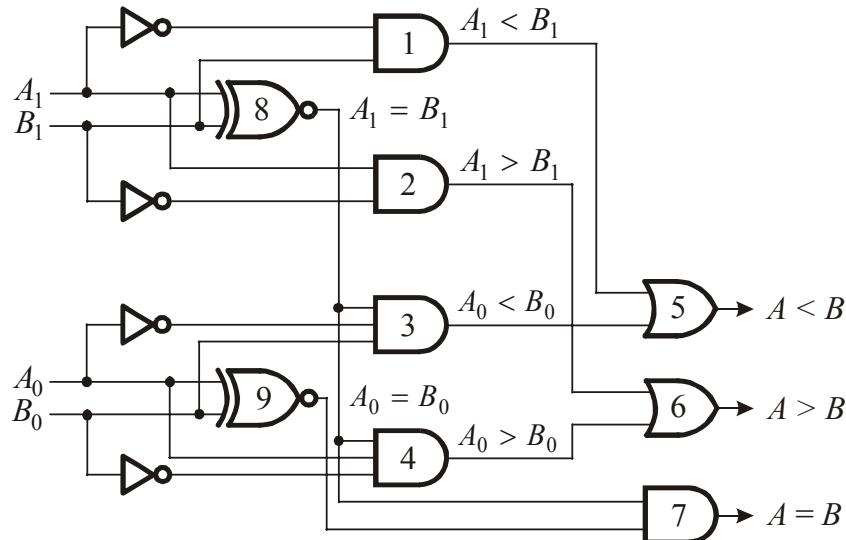


Fig.2.98 Schema logică a comparatorului de doi biți.

Revenind la comparatorul de 4 biți şi având în vedere relaţiile (2.166) ÷ (2.168), se pot scrie expresiile ieşirilor acestui comparator.

Astfel, cele două numere care se compară sunt în relaţia $A < B$, adică $Z_1 = 1$, dacă $A_3 < B_3$, sau dacă A_3 şi B_3 coincid şi $A_2 < B_2$ sau dacă A_3 coincide cu B_3 şi A_2 coincide cu B_2 , iar $A_1 < B_1$ sau dacă A_3 coincide cu B_3 şi A_2 coincide cu B_2 şi A_1 coincide cu B_1 , iar $A_0 < B_0$. Din cele de mai sus rezultă, pentru Z_1 , expresia logică:

$$Z_1 = \overline{A}_3 B_3 + C_3 \overline{A}_2 B_2 + C_3 C_2 \overline{A}_1 B_1 + C_3 C_2 C_1 \overline{A}_0 B_0. \quad (2.170)$$

Cele două numere sunt egale, adică $Z_2 = 1$, dacă toate cifrele corespondente ale celor două numere coincid, prin urmare:

$$Z_2 = C_3 C_2 C_1 C_0. \quad (2.171)$$

Cele două numere sunt în relația $A > B$, adică $Z_3 = 1$, dacă $A_3 > B_3$, sau dacă A_3 coincide cu B_3 și $A_2 > B_2$ sau dacă A_3 coincide cu B_3 și A_2 coincide cu B_2 , iar $A_1 > B_1$ sau dacă A_3 coincide cu B_3 și A_2 coincide cu B_2 și A_1 coincide cu B_1 , iar $A_0 > B_0$. Din cele de mai sus rezultă pentru Z_3 expresia logică:

$$Z_3 = A_3 \bar{B}_3 + C_3 A_2 \bar{B}_2 + C_3 C_2 A_1 \bar{B}_1 + C_3 C_2 C_1 A_0 \bar{B}_0. \quad (2.172)$$

Comparatoarele realizate sub formă de circuite integrate MSI, sunt construite pentru compararea numerelor cu patru cifre binare. Pentru ca ele să poată fi folosite la realizarea circuitelor de comparare a numerelor cu mai multe poziții, comparatoarele pentru patru poziții sunt prevăzute cu trei intrări suplimentare, X_1 , X_2 și X_3 (v. fig. 2.96), care indică relația stabilită între cele două numere A și B , pe baza comparării cifrelor corespondente de pondere inferioară. Intrarea X_1 corespunde relației $A < B$ și se leagă la ieșirea Z_1 a comparatorului precedent, intrarea X_2 corespunde relației $A = B$ și se leagă la ieșirea Z_2 a comparatorului precedent, s.a.m.d.

Circuitul comparator prevăzut cu aceste trei intrări suplimentare este descris structural de expresile:

$$Z_1 = \bar{A}_3 B_3 + C_3 \bar{A}_2 B_2 + C_3 C_2 \bar{A}_1 B_1 + C_3 C_2 C_1 \bar{A}_0 B_0 + C_3 C_2 C_1 C_0 X_1, \quad (2.173)$$

$$Z_2 = C_3 C_2 C_1 C_0 X_2, \quad (2.174)$$

$$Z_3 = A_3 \bar{B}_3 + C_3 A_2 \bar{B}_2 + C_3 C_2 A_1 \bar{B}_1 + C_3 C_2 C_1 A_0 \bar{B}_0 + C_3 C_2 C_1 C_0 X_3. \quad (2.175)$$

Circuitul integrat MSI 7485 [34] este un exemplu de comparator pentru patru biți realizat pe baza expresiilor (2.173) ÷ (2.175). Numerele care trebuie comparate se aplică la bornele $A0 \div A3$ și $B0 \div B3$. La acest circuit ieșirea Z_1 și intrarea X_1 sunt marcate cu $A < B$, ieșirea Z_2 și intrarea X_2 cu $A = B$, iar ieșirea Z_3 și intrarea X_3 cu $A > B$. Tabloul de adevăr al comparatorului 7485 este prezentat în tabelul 2.33.

Prin legarea în cascadă a unui număr n de circuite de acest tip se obține un comparator pentru numere cu $n \times 4$ cifre binare. De exemplu, pentru cuvinte de 24 de biți se construiește o structură cu două nivele folosind șase comparatoare. Un arbore cu trei nivele poate compara până la 120 de biți, s.a.m.d.

Exemplu. Să se realizeze un comparator de cuvinte de 7 biți utilizând două comparatoare de 4 biți, 7485.

Este vorba de extinderea domeniului de comparare a două cuvinte binare. Soluția este prezentată în figura 2.99. Se distribuie cei mai semnificativi 4 biți pe primul nivel, rezultatul comparării acestora aplicându-se celui de al doilea comparator pe pozițiile cele mai semnificative.

Tab.2.33 Tabelul de adevăr al comparatorului de 4 biți 7485.

Intrări				Ieșiri					
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$A > B$	$A < B$	$A = B$	$A > B$	$A < B$	$A = B$
$A_3 > B_3$	*	*	*	*	*	*	1	0	0
$A_3 < B_3$	*	*	*	*	*	*	0	1	0
$A_3 = B_3$	$A_2 > B_2$	*	*	*	*	*	1	0	0
$A_3 = B_3$	$A_2 < B_2$	*	*	*	*	*	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	*	*	*	*	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	*	*	*	*	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	*	*	*	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	*	*	*	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	0	0	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	1	0	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	*	*	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	1	0	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	0	1	1	0

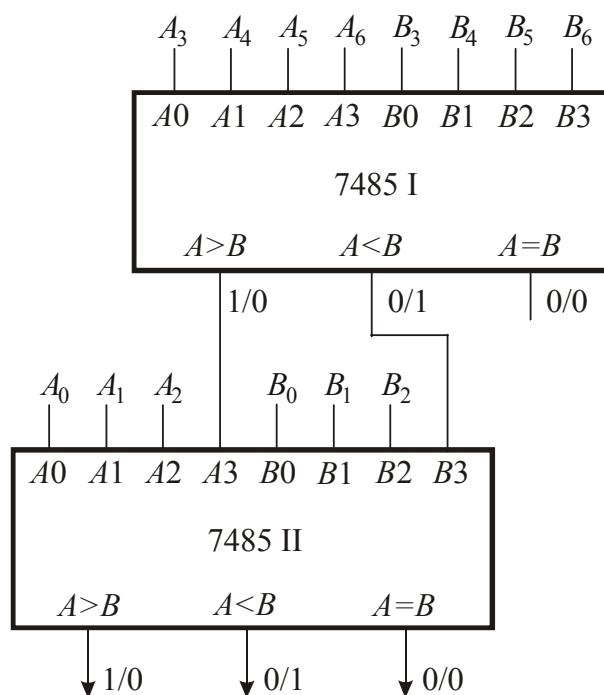


Fig.2.99 Schema de principiu a unui comparator de 7 biți.

În momentul în care, comparând biții cei mai semnificativi, 6, 5, 4, 3 (operație efectuată de comparatorul I) se va obține că $A > B$ (deci la ieșirea $A > B$ se va obține 1 logic, iar la ieșirile $A < B$ și $A = B$, 0 logic) și deoarece ieșirea $A > B$ a circuitului I de comparare este conectată la intrarea A_3 (bitul cel mai semnificativ al primului număr) a comparatorului II, iar ieșirea $A < B$ a comparatorului I la ieșirea B_3 (bitul cel mai semnificativ al celui de al doilea număr) a comparatorului II, este evident că $A_3 > B_3$ și se activează ieșirea $A > B$ a circuitului comparator II. Explicația este asemănătoare și în cazul în care, prin compararea primilor patru biți cei mai semnificativi, se va obține că $A < B$.

Când însă, prin compararea bițiilor celor mai semnificativi 6, 5, 4, 3 se va obține că $A = B$, această ieșire a comparatorului I nu trebuie conectată la nici o intrare a comparatorului II, deoarece în acest caz relația dintre numerele A și B este dată de rezultatul comparației celorlalți biți, mai puțin semnificativi (2,1,0), operație efectuată de comparatorul II. Se observă că schema s-a obținut, fără a se folosi intrările suplimentare.

Exemplu. Să se realizeze comparația a 2 cuvinte de 8 biți utilizând două comparatoare de 4 biți, 7485.

Schema logică este prezentată în figura 2.100. Funcționarea acesteia se poate deduce imediat dacă se are în vedere tabelul de adevar al comparatorului 7485 (v. tab. 2.33). Se observă din acest tabel că, de exemplu, pentru prima linie, dacă $A_3 > B_3$, indiferent de semnalele aplicate pe intrările suplimentare, $A < B$, $A = B$, $A > B$, va fi activată (1 logic) ieșirea $A > B$. Pentru comparația celor 2 cuvinte de 8 biți se utilizează 2 comparatoare de 4 biți, 7485, conectate în cascadă. Dintre cele două comparații, primul va fi folosit pentru comparația primelor 4 biți de rang inferior (0,1,2,3), iar cel de-al doilea evident, pentru comparația celorlalți 4 biți ai numerelor (4,5,6,7 – rangurile superioare). Intrările suplimentare ale comparatorului II „își fac simțită prezența“ doar în momentul în care primii 4 biți (cei mai semnificativi) ai celor 2 numere, care se compara, sunt identici.

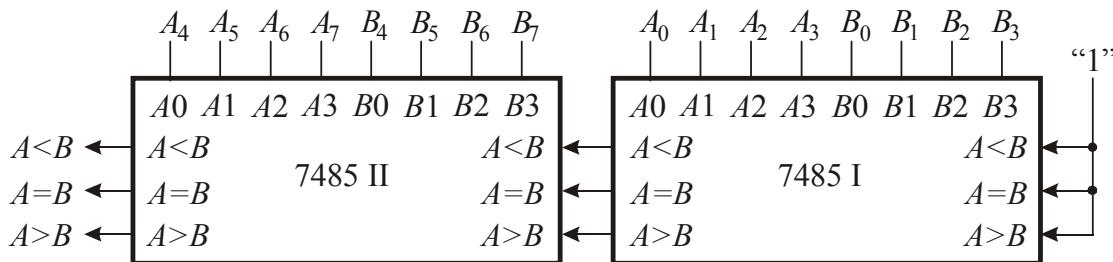


Fig.2.100 Comparator numeric pentru numere de 8 biți.

Exemplu. În figura 2.101 este date schema logică a unui circuit comparator pentru numerele A și B cu 16 cifre binare. Intrările suplimentare ($A < B$, $A = B$, $A > B$) pentru comparatorul primelor patru cifre mai puțin semnificative ale celor două numere, trebuie să fie în starea logică 1.

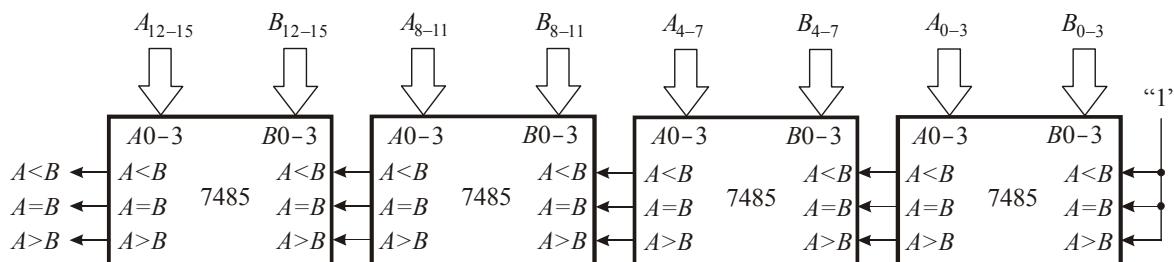


Fig.2.101 Circuit comparator pentru 16 ranguri.

2.5.5 Detectoare și generatoare de paritate

În procesul transmiterii informațiilor numerice pot apărea erori. O metodă simplă de detectare a acestora constă în utilizarea *codurilor detectoare de erori* (cu verificare de paritate sau imparitate, v. Anexa C). Aceste coduri se bazează pe faptul că la emisie se formează un nou cuvânt de cod, prin adăugarea unui bit suplimentar la cei existenți (biții

informaționali), astfel încât numărul de 1-uri din cuvântul nou format să fie par (sau impar). La recepție se verifică paritatea sau imparitatea numărului de 1-uri din cuvântul recepționat și în funcție de rezultatul verificării se decide asupra corectitudinii cuvântului recepționat.

Operațiile susmenționate se realizează cu CLC-uri numite *detectoare* și *generatoare de paritate* [2,22,26,27].

Realizarea detectoarelor de paritate se bazează pe proprietatea cunoscută a porții SAU-EXCLUSIV de a răspunde prin 1 logic atunci când una dintre intrări este pe 1 și cea de a doua pe 0 (imparitate), sau prin 0 logic atunci când ambele intrări sunt pe 1, respectiv 0 (paritate). Detectorul de paritate elementar (pentru cuvinte de doi biți) este deci circuitul de anticoincidentă (sumatorul modulo doi).

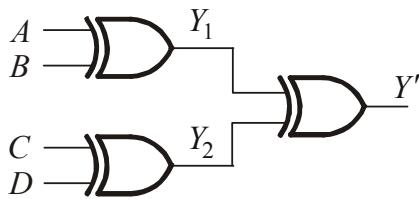


Fig.2.102 Detector de paritate de 4 biți.

Extinzând proprietatea menționată mai sus la circuitul cu trei porți din figura 2.102, se constată, din tabelul 2.34, că variabila de ieșire Y' este egală cu 1 dacă numărul variabilelor de intrare (A, B, C, D) egale cu 1, este impar și are valoarea 0 pentru un număr par de variabile de intrare care au valoarea 1. Deci circuitul poate fi folosit ca detector de paritate (imparitate) pentru 4 variabile de intrare.

Tab.2.34 Tabelul de adevăr pentru detectorul de paritate de 4 biți.

	A	B	C	D	Y_1	Y_2	Y'		A	B	C	D	Y_1	Y_2	Y'
0	0	0	0	0	0	0	0	8	1	0	0	0	1	0	1
1	0	0	0	1	0	1	1	9	1	0	0	1	1	1	0
2	0	0	1	0	0	1	1	10	1	0	1	0	1	1	0
3	0	0	1	1	0	0	0	11	1	0	1	1	1	0	1
4	0	1	0	0	1	0	1	12	1	1	0	0	0	0	0
5	0	1	0	1	1	1	0	13	1	1	0	1	0	1	1
6	0	1	1	0	1	1	0	14	1	1	1	0	0	1	1
7	0	1	1	1	1	0	1	15	1	1	1	1	0	0	0

Un detector de paritate (imparitate) pentru 8 variabile de intrare se poate obține folosind două circuite identice cu cel din figura 2.102 și aplicând ieșirile Y' și Y'' ale acestora la intrările unei porți SAU-EXCLUSIV. La ieșirea porții se obține funcția Y egală cu 1 în caz de imparitate și nulă în caz de paritate. Prin dublarea circuitului astfel obținut și atașarea a încă unei porți, se obține un detector de paritate (imparitate) pentru 16 biți, s.a.m.d.

Introducerea unei noi porți SAU-EXCLUSIV la ieșirea schemei din figura 2.102, împreună cu comanda P (v. fig. 2.103), asigură pe ieșirea Y nivel logic 1 sau 0, în funcție de numărul unităților din cuvântul de cod și de comanda P aplicată, după cum urmează:

– pentru $P = 0$, $Y = Y'$ și circuitul este un generator de paritate (pe ieșirea Y apare nivel logic 1 dacă numărul unităților din cuvântul de intrare este impar);

– pentru $P = 1$, $Y = \bar{Y}'$ și circuitul este un generator de imparitate (pe ieșirea Y apare nivel logic 1 când numărul de unități din cuvântul de intrare este par).

Acest generator de paritate/imparitate (realizat conform schemei din figura 2.103)

este folosit la calcularea valorii bitului de paritate ce se adaugă biților informaționali în codul mesajului transmis la stația de emisie.

Bitul de paritate este folosit ca bit redundant în realizarea codului detector de erori de transmisie (așa numitul cod cu bit de paritate). Variabilei P îi se dau, așa cum s-a menționat, două valori: $P = 0$, în cazul folosirii criteriului de paritate pară și $P = 1$, în cazul folosirii criteriului de paritate impară.

În cazul criteriului de paritate pară, bitul de paritate (variabila Y din schemă) ia valoarea:

$$Y = Y' \oplus P = Y' \oplus 0 = \begin{cases} 0, & \text{dacă numărul biților informaționali 1 este par } (Y' = 0), \\ 1, & \text{dacă numărul biților informaționali 1 este impar } (Y' = 1), \end{cases}$$

În cazul criteriului de paritate impară, bitul de paritate (variabila Y din schemă) ia valoarea:

$$Y = Y' \oplus P = Y' \oplus 1 = \begin{cases} 1, & \text{dacă numărul biților informaționali 1 este par } (Y' = 0), \\ 0, & \text{dacă numărul biților informaționali 1 este impar } (Y' = 1), \end{cases}$$

Un exemplu de utilizare a circuitului logic din figura 2.103, făcând uz de dubla sa funcție, la detectarea erorilor de transmisie a informației binare, este prezentat în figura 2.104.

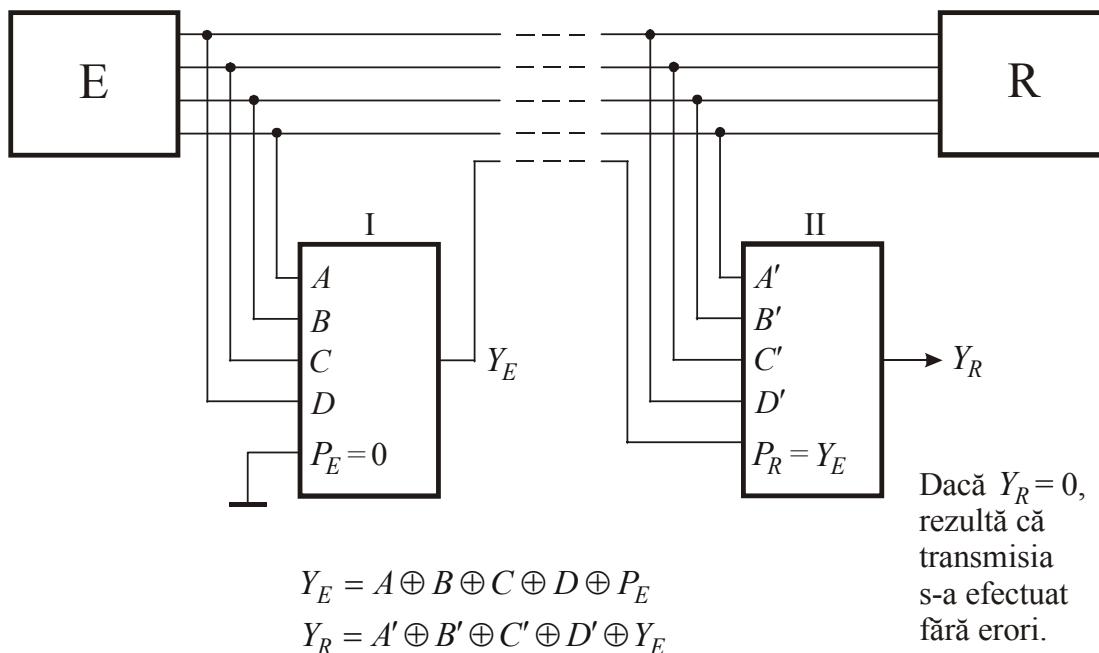


Fig.2.104 Lanț de transmisie a informației binare cu verificare de paritate.

Circuitul I funcționează, la locul de emisie al informației (E), ca generator de paritate, furnizând la ieșirea Y_E un bit de paritate, de valoare 1 în toate cazurile în care numărul variabilelor de intrare (A, B, C, D) care au valoarea logică 1 este impar și de valoare 0 în cazul în care acest număr este par (schema aplică criteriul parității pare). Bitul de paritate este apoi încorporat în codul transmîs, pe poziția c.m.p.s. din cod. Cuvântul de cod obținut va avea un număr par de biți egali cu 1.

La punctul de recepție (R) cei cinci biți ai cuvântului recepționat sunt introdusi în circuitul II, care de această dată funcționează ca un detector de paritate, validând sau nu, prin bitul său de paritate, recepționarea cuvântului de cod. Pentru transmisia fără erori $Y_R = 0$, iar apariția unei erori de transmitere care produce modificarea parității este semnalizată prin $Y_R = 1$.

Pentru cazul unor circuite de mai mulți biți se poate utiliza circuitul integrat specializat 74180 [34], care este de fapt un generator/detector de paritate (imparitate) de 8 biți (8 biți de date plus 1 bit de paritate). În figura 2.105 se prezintă schema logică a detectorului și generatorului de paritate integrat.

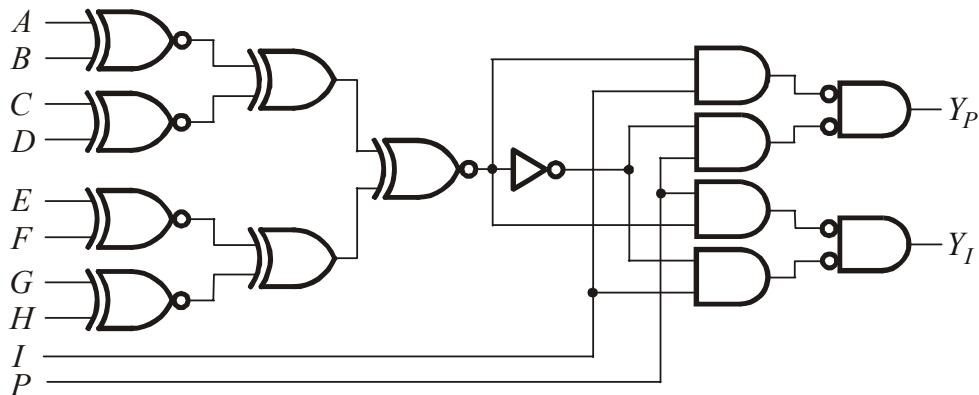


Fig.2.105 Generator/detector de paritate de 8 biți.

Tab.2.35 Tabelul de adevăr pentru detectorul/generatorul de paritate de 8 biți 74180.

Intrări		Ieșiri		
Numărul de biți 1 la intrările $A \div H$	P	I	Y_P	Y_I
Par	1	0	1	0
Impar	1	0	0	1
Par	0	1	0	1
Impar	0	1	1	0
Indiferent	1	1	0	0
Indiferent	0	0	1	1

Pe lângă intrările de date (A, \dots, H) sunt prevăzute intrările P (par) și I (impar) care: permit funcționarea ca detector/generator de paritate cât și de imparitate; pot servi, de asemenea, pentru invalidarea circuitului și asigură posibilitatea interconectării mai multor circuite, în cazul aplicațiilor în care cuvântul de intrare are o lungime mai mare de 8 biți. Sunt prevăzute două ieșiri Y_P și Y_I , pentru a crește versatilitatea circuitului și pentru asigurarea interconectabilității cu circuite similare, fără circuite auxiliare. Funcționarea circuitului rezultă și din tabelul 2.35.

În figura 2.106 este exemplificat modul în care se pot interconecta mai multe circuite integrate 74180, în scopul realizării unui detector/generator de paritate pentru un cuvânt de intrare cu mai mult de 8 biți (24).

Acest circuit poate fi conectat într-o rețea arborescentă cu două nivele pentru a se

obține o structură pentru $8 \times 9 = 72$ biți, utilizând $9 + 1 = 10$ capsule integrate. Cu o rețea parțială, ce nu folosește complet intrările celui de al doilea nivel, se poate obține un circuit pentru $8 + 8 + 7 = 23$ de biți, numai cu trei capsule, s.a.m.d. În plus, cu aceleși tipuri de circuite poate fi realizată o rețea care să genereze coduri „Hamming“ (v. Anexa C), ce indică nu numai apariția unei erori dar și bitul eronat (de exemplu, 74280 [34]).

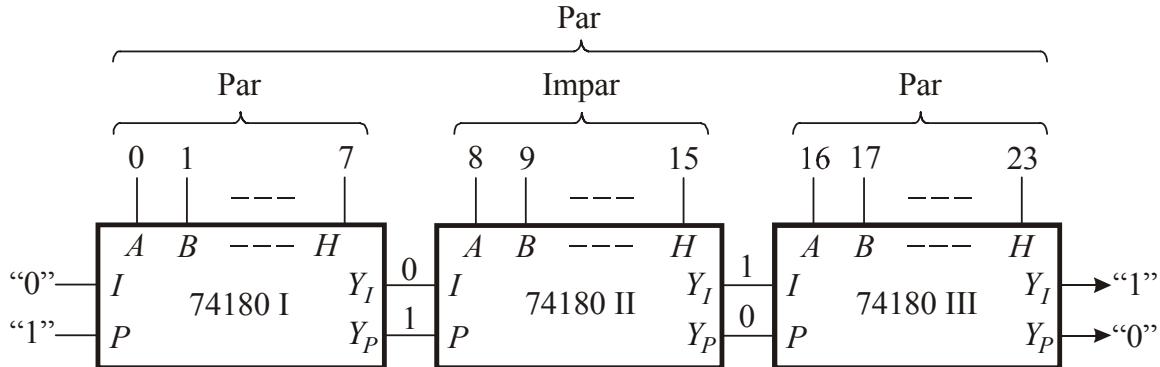


Fig.2.106 Generator/detector de paritate de 24 biți.

2.5.6 Sumatoare binare

Subsistemu logic combinațional care asigură, direct sau indirect, efectuarea tuturor operațiilor aritmetice într-un sistem de calcul este *sumatorul*.

În principiu, există două posibilități de realizare a unui sumator binar:

- *sumatorul numit serie*, la care biții de același rang, ai celor două numere care trebuie adunate, sunt aplicăți la intrările sumatorului în mod succesiv/în serie, rang după rang; pentru realizarea unui astfel de sumator se folosește un singur circuit elementar (sumatorul complet pentru un rang – v. explicațiile de mai jos) [26,33];

- *sumatorul paralel/derivație*, format, în cazul general, din n circuite elementare identice (sumatoare complete pentru un rang) la care cei n biți, ai celor două numere care trebuie adunate, se aplică simultan/în paralel la intrările celor n circuite elementare [2,19,22,23,26÷28,30,33].

Prima soluție, deși este mai economică din punct de vedere al componentelor (este necesar un singur circuit elementar în loc de n circuite elementare pentru numere de n cifre) este mult prea lentă în raport cu cea de a doua, motiv pentru care se folosește foarte rar. În continuare, în lucrare se vor trata doar sumatoarele binare paralele.

Sumatorul binar paralel este un CLC cu $2n+1$ intrări și $n+1$ ieșiri, care însumează cifrele, aflate pe poziții corespondente, a două numere binare cu n cifre, A_1, A_2, \dots, A_n și B_1, B_2, \dots, B_n , ținând cont de un eventual transport inițial C_0 și generând un număr binar cu $n+1$ cifre, cifrele sumei S_1, S_2, \dots, S_n și cifra transportului C_n (v. fig. 2.107). Ponderea cifrei cu indicele i , pentru $i = 1, 2, \dots, n$, este 2^i . De obicei, sumatorul paralel se rea-

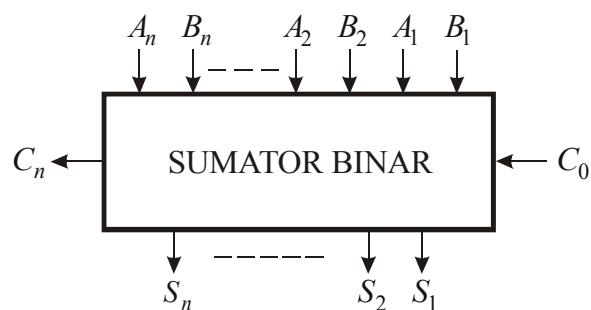


Fig.2.107 Schema-bloc a unui sumator cu n ranguri.

lizează sub forma unui circuit iterativ¹⁵, în care celula de bază este un sumator complet pentru un rang (sumatorul elementar) sau pentru mai multe ranguri, de obicei patru.

CLC-ul care servește pentru efectuarea sumei a două numere binare cu câte un bit, fără a ține seama de transportul de la bitul de rang imediat inferior, este denumit *semisumator* [19,22,23,26÷28,30,33].

Tabelul 2.36 prezintă funcționarea semisumatorului având ca variabile de intrare numerele A_i și B_i , de câte un bit și ca variabile de ieșire suma parțială S'_i și transportul C_i . Din tabel rezultă:

$$S'_i = \overline{A}_i B_i + A_i \overline{B}_i = A_i \oplus B_i \quad (2.176)$$

și

$$C_i = A_i B_i, \quad (2.177)$$

relații care conduc la schema logică din figura 2.108,b, a cărei reprezentare simbolică este redată în figura 2.108,a.

Tab.2.36 Tabelul de adevar al semisumatorului.

Intrări		Ieșiri	
A_i	B_i	S'_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

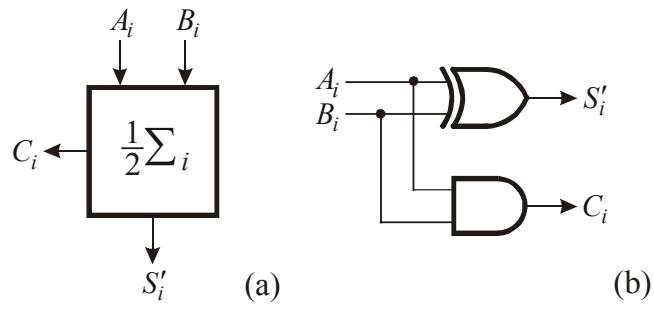


Fig.2.108 Reprezentarea simbolică (a) și schema logică (b) a unui semisumator

Sumatorul binar complet sau *sumatorul elementar* realizează însumarea a două cifre binare de același rang, A_i și B_i cu cifra transportului pentru rangul respectiv C_{i-1} , generând cifra sumei de rang i , S_i și transportul pentru rangul superior, C_i [19,22,23,26÷28, 30,33]. Funcțiile booleene $S_i(A_i, B_i, C_{i-1})$ și $C_i(A_i, B_i, C_{i-1})$ sunt definite în tabelul 2.37, întocmit pe baza regulii de adunare binară și au următoarele forme canonice disjunctive:

$$S_i = \overline{A}_i \overline{B}_i C_{i-1} + \overline{A}_i B_i \overline{C}_{i-1} + A_i B_i C_{i-1} + A_i \overline{B}_i \overline{C}_{i-1}, \quad (2.178)$$

$$C_i = \overline{A}_i B_i C_{i-1} + A_i B_i \overline{C}_{i-1} + A_i B_i C_{i-1} + A_i \overline{B}_i C_{i-1}. \quad (2.179)$$

Expresiile funcțiilor S_i și C_i se mai pot scrie:

$$\begin{aligned} S_i &= (\overline{A}_i B_i + A_i \overline{B}_i) \overline{C}_{i-1} + (A_i B_i + \overline{A}_i \overline{B}_i) C_{i-1} = (A_i \oplus B_i) \overline{C}_{i-1} + (\overline{A}_i \oplus B_i) C_{i-1} = \\ &= A_i \oplus B_i \oplus C_{i-1}, \end{aligned} \quad (2.180)$$

¹⁵ Circuitele iterative sunt alcătuite dintr-un număr mare de circuite elementare identice conectate între ele sub forma unei rețele regulate. Ele permit realizarea sistemelor de funcții de comutare, de un număr mare de variabile, mai avantajos decât cu circuite combinaționale cu mai multe ieșiri, construite cu două sau mai multe nivele de porți logice. Avantajul constă în reducerea numărului de circuite logice folosite, precum și în posibilitatea de standardizare a acestora.

respectiv:

$$C_i = A_i B_i (C_{i-1} + \bar{C}_{i-1}) + (\bar{A}_i B_i + A_i \bar{B}_i) C_{i-1} = A_i B_i + (A_i \oplus B_i) C_{i-1}. \quad (2.181)$$

Tab.2.37 Tabelul de adevar al sumatorului complet.

Intrari			Iesiri	
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$A_i B_i$	00	01	11	10
C_{i-1}	0	1	0	1
	1	0	1	0

(a)

$A_i B_i$	00	01	11	10
C_{i-1}	0	0	1	0
	1	1	1	1

(b)

Fig.2.109 Diagramele Karnaugh pentru functiile S_i si C_i ale sumatorului complet.

Având în vedere diagrama Karnaugh din figura 2.109,b se poate scrie, pentru funcția C_i , următoarea formă minimă:

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1} = A_i B_i + (A_i + B_i) C_{i-1}. \quad (2.182)$$

În figura 2.110,b este reprezentată schema logică a unui sumator complet (s-au implementat relațiile (2.180) și (2.181)).

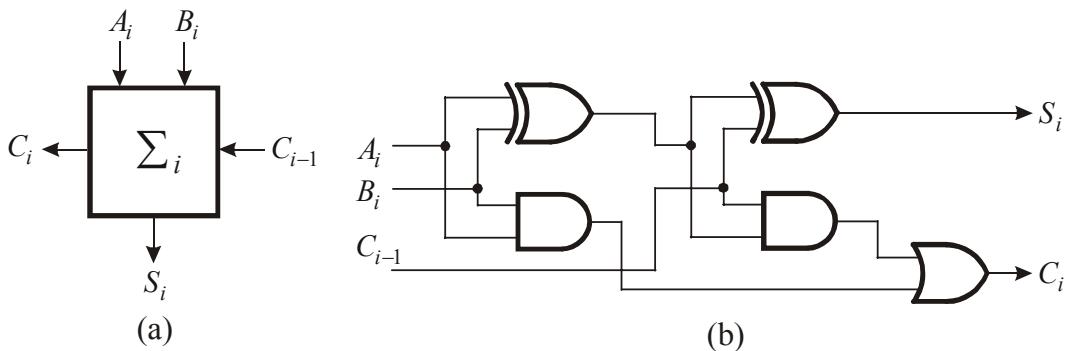


Fig.2.110 Sumatorul complet: (a) reprezentarea simbolica; (b) schema logica.

Se observă că sumatorul complet pentru bitul de rangul i se poate realiza utilizând două semisumatoare.

Prin interconectarea mai multor sumatoare complete pentru un bit pot fi realizate sumatoare pentru cuvinte binare cu mai mulți biți. De exemplu, sumatorul binar paralel pentru patru ranguri realizează însumarea a două numere binare cu câte patru cifre, notează A_1, A_2, A_3 și A_4 , respectiv B_1, B_2, B_3 și B_4, A_4 și B_4 , fiind cifrele cele mai semnificative. Însumarea acestor patru cifre se face în paralel. Sumatoarele binare paralele se clas-

fică, în funcție de modul în care se obține transportul pentru rangurile superioare, în *sumatoare cu propagarea transportului în serie* și *sumatoare cu propagarea transportului în paralel*, denumite și *sumatoare cu transport anticipat* sau *cu transport simultan* [2,15, 19,22,23,26,28,33].

Într-un sumator cu propagarea transportului în serie, însumarea a două cifre de rang i , A_i și B_i , se face după ce s-a obținut transportul C_{i-1} , din rangul inferior $i-1$. Schema sumatorului se bazează pe un sumator complet pentru un rang repetat de un număr de ori egal cu numărul de poziții binare ale operanzilor. În cazul a patru cifre binare se obține schema din figura 2.111, în care fiecare bloc notat cu Σ_i , ($i=1,2,3,4$) reprezintă un sumator binar complet pentru un rang.

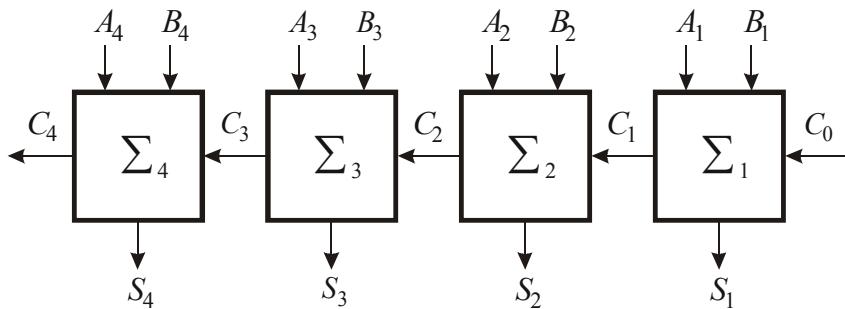


Fig.2.111 Sumator de 4 biți cu propagarea transportului în serie.

Timpul maxim în care se efectuează adunarea, este determinat de timpul în care se propagă transportul până la rangul cel mai semnificativ. Pentru schema din figura 2.111 acest timp, notat cu t_{ad} , este dat de relația:

$$t_{ad} = 3t_T + t_s, \quad (2.183)$$

unde, t_T este timpul în care se realizează funcția C_i , iar t_s este timpul în care se realizează funcția S_i , într-un sumator pentru un rang. În general, pentru n ranguri, se obține pentru timpul de adunare, t_{ad} , expresia:

$$t_{ad} = (n-1)t_T + t_s. \quad (2.184)$$

Din relația (2.184) rezultă că pentru reducerea timpului de adunare este importantă reducerea la minimum a timpului de propagare a transportului pe un rang, t_T .

Pentru reducerea întârzierii datorate propagării transportului în serie se folosește sumatorul cu transport anticipat.

Analizând relația bitului de transport (v. ec. (2.181)), se observă că transportul C_i devine:

$$C_i = \begin{cases} 1 & \text{și este independent de } C_{i-1}, \text{ când } A_i = B_i = 1, \\ 0 & \text{și este independent de } C_{i-1}, \text{ când } A_i = B_i = 0, \\ C_{i-1} & \text{când } A_i \neq B_i. \end{cases}$$

Această constatare a condus la ideea calculării anticipate a cifrei binare de transport

în funcție de datele de intrare, simultan pentru toate rangurile. În felul acesta, nu se mai așteaptă efectuarea calculelor pe rangul mai puțin semnificativ în vederea generării cifrei de transport ce se adună la cifrele rangului următor. Aceasta înseamnă, în cazul sumatorului pentru patru ranguri, că trebuie realizate funcțiile:

$$\begin{aligned} C_1 &= f_1(A_1, B_1, C_0), \\ C_2 &= f_2(A_2, A_1, B_2, B_1, C_0), \\ C_3 &= f_3(A_3, A_2, A_1, B_3, B_2, B_1, C_0), \\ C_4 &= f_4(A_4, A_3, A_2, A_1, B_4, B_3, B_2, B_1, C_0). \end{aligned} \quad (2.185)$$

Pentru obținerea unei forme cât mai simple a funcțiilor (2.185) se pleacă de la expresia transportului pentru un rang (v. ec.(2.182)), dată mai jos:

$$C_i = A_i B_i + C_{i-1}(A_i + B_i) = G_i + C_{i-1} P_i, \quad (2.186)$$

în care s-au introdus funcțiile auxiliare G_i , transportul generat în rangul i (dacă $A_i B_i = 1$, rezultă $C_i = 1$ indiferent de valoarea lui C_{i-1}) și P_i , condiția de propagare a transportului generat în rangurile inferioare lui i , peste rangul i ($P_i = 1$ va determina „traversarea“ celei de către valoarea bitului de transport C_{i-1}). Relația (2.186) arată că C_i poate avea valoarea 1 dacă se generează în celula i ($G_i = 1$) sau ca urmare a propagării ($P_i = 1$) a unui C_{i-1} de la intrare. Scriind pe C_{i-1}, \dots, C_1 sub forma (2.186), rezultă expresiile:

$$\begin{aligned} C_{i-1} &= A_{i-1} B_{i-1} + C_{i-2}(A_{i-1} + B_{i-1}) = G_{i-1} + C_{i-2} P_{i-1}, \\ &\dots \\ C_1 &= A_1 B_1 + C_0(A_1 + B_1) = G_1 + C_0 P_1, \end{aligned} \quad (2.187)$$

unde C_0 este transportul pentru rangul cel mai puțin semnificativ.

Înlocuind în relația (2.186) pe C_{i-1} , apoi C_{i-2}, \dots, C_1 cu expresiile (2.187) se obține expresia generală a funcției C_i :

$$\begin{aligned} C_i &= G_i + P_i(G_{i-1} + P_{i-1}(G_{i-2} + \dots + P_2(G_1 + C_0 P_1))) = \\ &= G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \dots + P_i P_{i-1} \dots P_2 G_1 + P_i P_{i-1} \dots P_2 P_1 C_0. \end{aligned} \quad (2.188)$$

Particularizând relația (2.188) pentru $i = 1, 2, 3, 4$ se obțin expresiile funcțiilor transporturilor C_1, C_2, C_3 și C_4 , pentru sumatorul de patru ranguri:

$$C_1 = G_1 + P_1 C_0, \quad (2.189)$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0, \quad (2.190)$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0, \quad (2.191)$$

$$C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0, \quad (2.192)$$

Pe baza relațiilor (2.189) \div (2.192) și (2.180) se obține schema logică a sumatorului de 4 biți cu transport anticipat din figura 2.112.

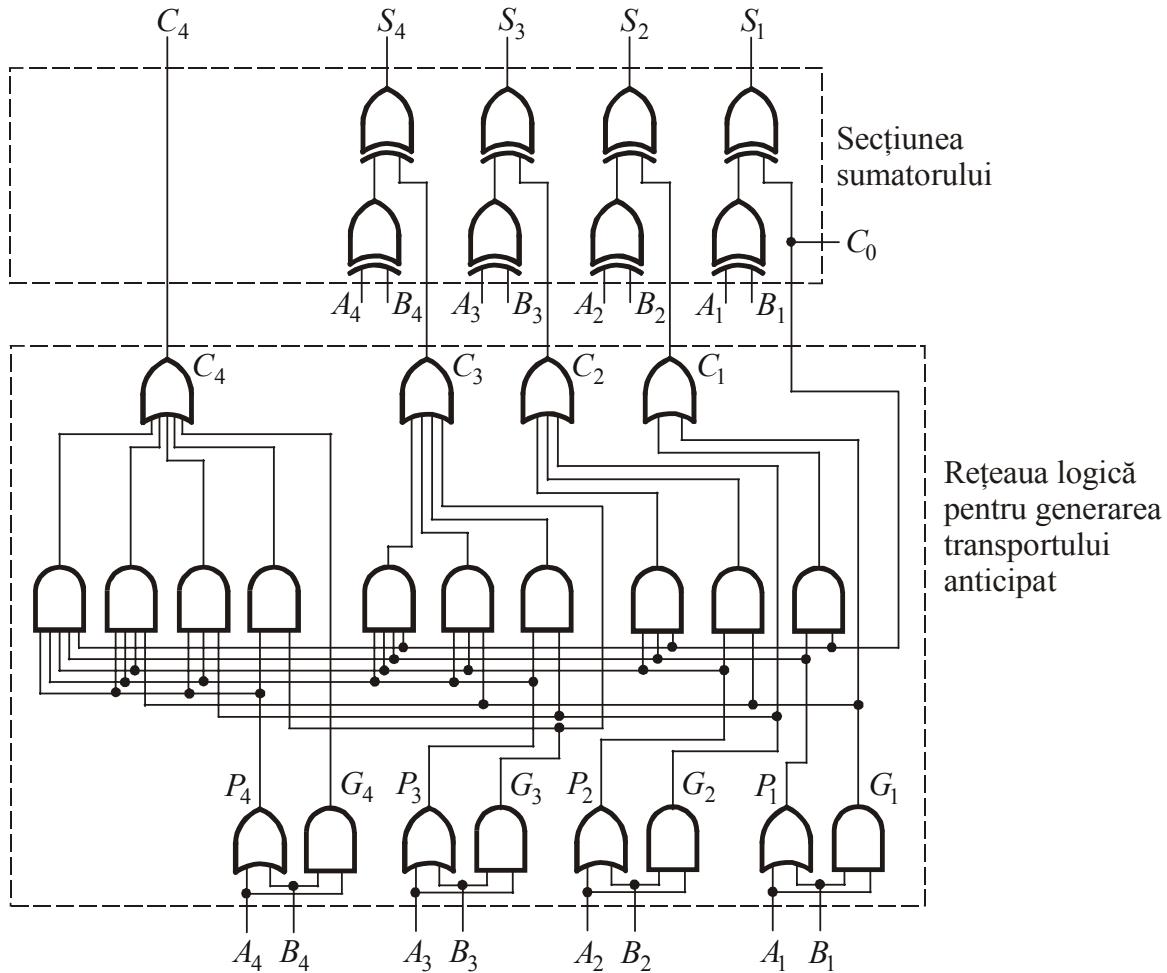


Fig.2.112 Schema logică a unui sumator binar de 4 biți cu transport anticipat.

Deoarece se urmărește realizarea în timp cât mai scurt a funcțiilor (2.189) \div (2.192), cele mai indicate circuite sunt porțile logice řI-SAU-NU. Pentru sinteza cu acest tip de porți sunt necesare formele normale disjunctive ale negatelor funcțiilor. De aceea, în practică, se realizează funcțiile auxiliare \bar{G}_i și \bar{P}_i , respectiv funcțiile:

$$\begin{aligned} \bar{C}_i &= \overline{A_i B_i + (A_i + B_i) C_{i-1}} = \overline{A_i B_i} [(\overline{A_i + B_i}) + \bar{C}_{i-1}] = \\ &= \bar{P}_i + \bar{C}_{i-1} \bar{G}_i. \end{aligned} \quad (2.193)$$

Pe baza relației (2.193) se obțin, pentru patru ranguri:

$$C_1 = \overline{\bar{P}_1 + \bar{G}_1 \bar{C}_0}, \quad (2.194)$$

$$C_2 = \overline{\bar{P}_2 + \bar{G}_2 \bar{P}_1 + \bar{G}_2 \bar{G}_1 \bar{C}_0}, \quad (2.195)$$

$$C_3 = \overline{\bar{P}_3 + \bar{G}_3 \bar{P}_2 + \bar{G}_3 \bar{G}_2 \bar{P}_1 + \bar{G}_3 \bar{G}_2 \bar{G}_1 \bar{C}_0}, \quad (2.196)$$

$$C_4 = \overline{P_4} + \overline{G_4}\overline{P_3} + \overline{G_4}\overline{G_3}\overline{P_2} + \overline{G_4}\overline{G_3}\overline{G_2}\overline{P_1} + \overline{G_4}\overline{G_3}\overline{G_2}\overline{G_1}\overline{C_0}. \quad (2.197)$$

Dezavantajul sumatoarelor cu transport anticipat constă în faptul că odată cu creșterea numărului de ranguri crește considerabil complexitatea circuitelor pentru obținerea transportului. Din acest motiv, pentru adunarea unor operanzi cu număr mare de cifre, se folosesc mai multe sumatoare cu propagarea transportului în paralel pentru un număr limitat de ranguri, de obicei patru. Transportul între aceste sumatoare se propagă fie în serie, fie în paralel.

Unele circuite sumatoare MSI, pentru patru biți, de exemplu circuitul 7483 [34], a cărui schemă logică este prezentată în figura 2.113, realizează anticipat numai transportul final, notat cu C_4 , transporturile intermediare \bar{C}_1, \bar{C}_2 , și \bar{C}_3 , fiind realizate succesiv, pe baza relației (2.193). Anticiparea ultimului transport permite circuitului sumator pentru rangurile superioare să înceapă lucrul mai repede, reducându-se astfel substanțial timpul de adunare al numerelor cu multe cifre binare.

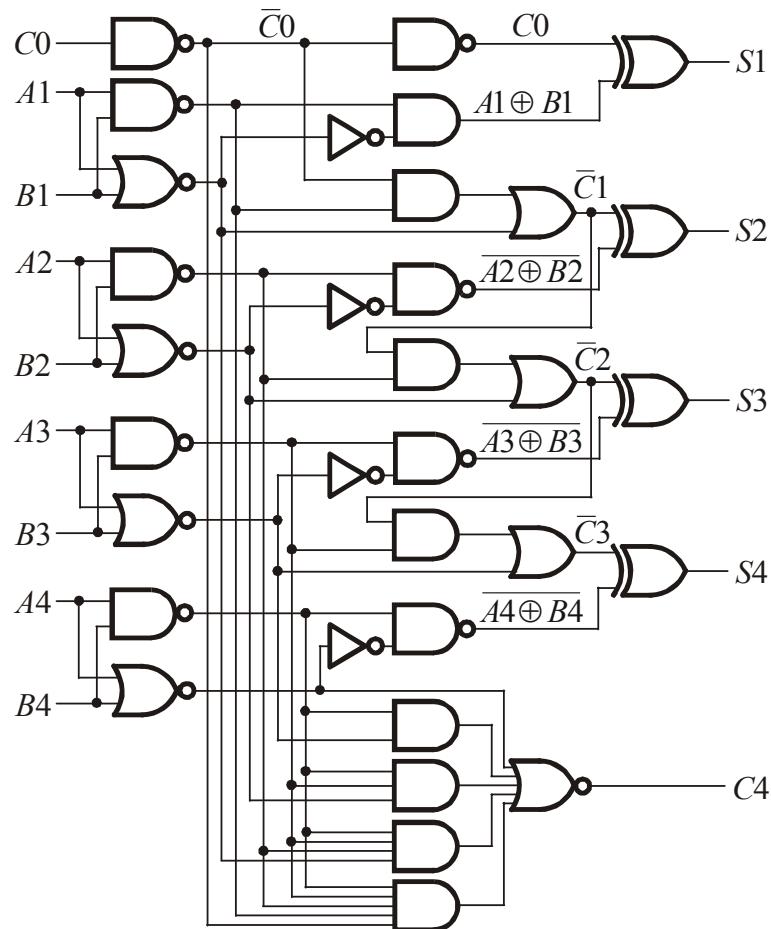


Fig.2.113 Schema logică a circuitului sumator 7483.

Așa cum s-a mai precizat, sumatoarele pentru mai multe ranguri se pot realiza prin legarea în cascadă a mai multor sumatoare complete pentru 4 biți. Astfel, cu circuitul integrat 7483 se poate realiza un sumator de numere întregi de n biți, conectând în serie un număr $k = n/4$ de circuite integrate, unde n este întotdeauna o putere a lui 2. În figura 2.114 este dată schema logică a unui sumator pentru 16 ranguri binare.

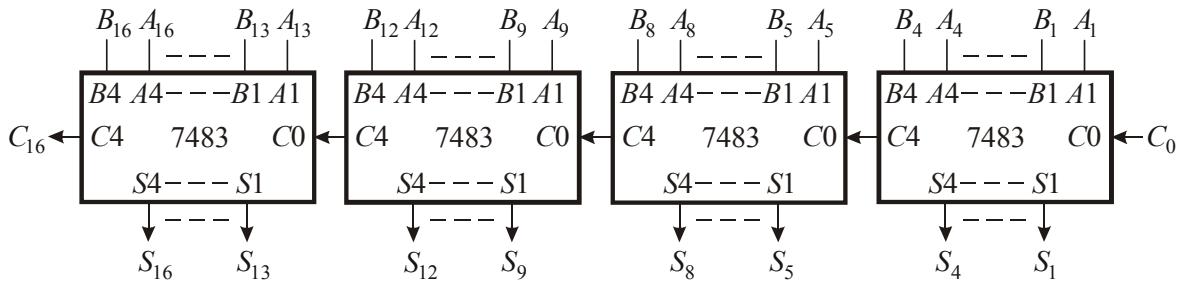


Fig.2.114 Schema logică a unui sumator pentru 16 ranguri binare, realizat cu circuite 7483.

Pentru a se folosi transporturile intermedii, la realizarea sumei, fără inversarea lor, la sumatorul din figura 2.113 se realizează și semisuma cifrelor de același rang, S'_i , sub forma negată, iar pentru calculul sumei pentru un rang, S_i , se folosește expresia:

$$S_i = \bar{S}'_i \oplus \bar{C}_{i-1}, \quad (2.198)$$

care se bazează pe următoarea proprietate a funcției SAU-EXCLUSIV:

$$a \oplus b = \bar{a} \oplus \bar{b}. \quad (2.199)$$

Pentru realizarea transportului anticipat între mai multe sumatoare de patru biți este necesar ca acestea să realizeze în locul transportului C_4 condiția de generare a transportului în circuitul $j(G_j)$ și condiția de propagare a transportului peste circuitul $j(P_j)$, având expresiile:

$$G_j = G_{j4} + G_{j3}P_{j4} + G_{j2}P_{j3}P_{j4} + G_{j1}P_{j2}P_{j3}P_{j4} \quad (2.200)$$

și

$$P_j = P_{j4}P_{j3}P_{j2}P_{j1}. \quad (2.201)$$

În felul acesta, considerându-se sumatorul cu 4 celule ca un bloc, pe care s-au definit cele două funcții G_j și P_j , este posibilă extinderea principiului prezentat anterior, pe un alt doilea nivel, peste mai multe sumatoare de 4 biți, utilizând un circuit de transport anticipat suplimentar descris tot de relațiile (2.189) ÷ (2.192). Dacă se prevede transportul anticipat pentru patru circuite sumatoare de patru biți (v. fig. 2.115, în care, pentru simplificare, nu s-au mai reprezentat intrările pentru operanzi), expresiile transporturilor pentru fiecare circuit sunt date de relațiile (2.189) ÷ (2.192). Schema prezentată poate fi extinsă, simplu, pentru numere de lungime mai mare de 16 biți (32, 48, 64).

În locul funcției C_4 se preferă adesea, aşa cum s-a precizat mai sus, funcția G , condiția de generare a transportului în grupul de patru sumatoare, respectiv funcția P , condiția de propagare a transportului peste acest grup. Expresiile acestor funcții sunt (v. ec. (2.200) și (2.201)):

$$G = G_4 + G_3P_4 + G_2P_3P_4 + G_1P_2P_3P_4 \quad (2.202)$$

și

$$P = P_4P_3P_2P_1. \quad (2.203)$$

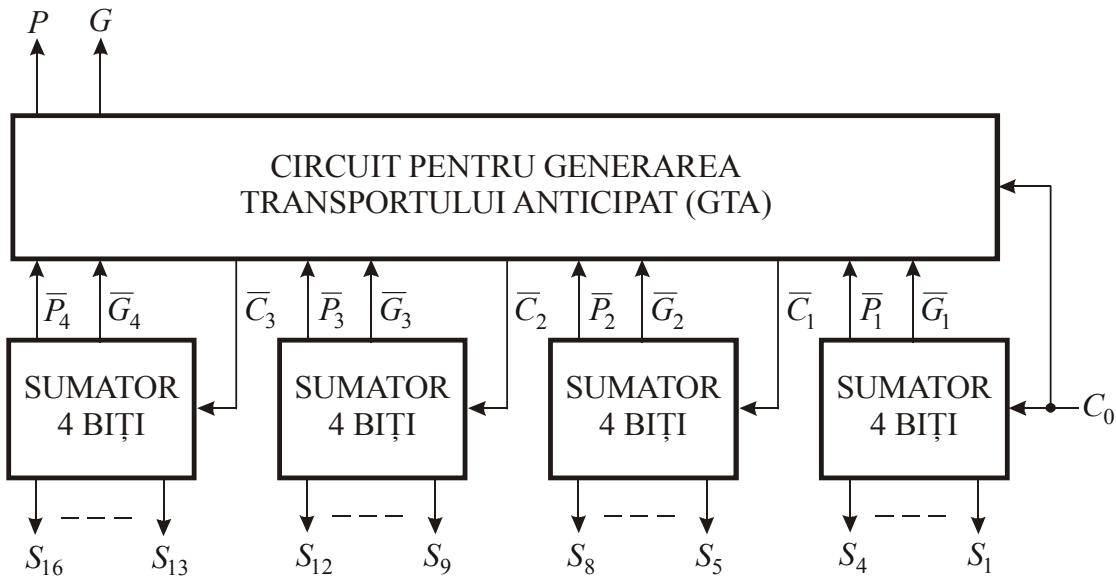


Fig.2.115 Schema-bloc a unui sumator cu anticiparea transportului.

Din motivele arătate mai sus, se realizează astfel funcțiile auxiliare \bar{G}_j și \bar{P}_j , pentru $j = 1, \dots, 4$. Pe de altă parte, este de dorit ca și funcțiile C_1 , C_2 și C_3 să fie realizate cât mai repede, utilizându-se în acest scop circuite řI-SAU-NU. În consecință, sunt necesare formele normale disjunctive ale funcțiilor \bar{C}_1 , \bar{C}_2 și \bar{C}_3 . Negând expresiile (2.189) ÷ (2.192), se obține:

$$\bar{C}_1 = \bar{G}_1(\bar{P}_1 + \bar{C}_0), \quad (2.204)$$

$$\bar{C}_2 = \bar{G}_2(\bar{P}_2 + \bar{G}_1)(\bar{P}_2 + \bar{P}_1 + \bar{C}_0), \quad (2.205)$$

$$\bar{C}_3 = \bar{G}_3(\bar{P}_3 + \bar{G}_2)(\bar{P}_3 + \bar{P}_2 + \bar{G}_1)(\bar{P}_3 + \bar{P}_2 + \bar{P}_1 + \bar{C}_0), \quad (2.206)$$

$$\bar{C}_4 = \bar{G}_4(\bar{P}_4 + \bar{G}_3)(\bar{P}_4 + \bar{P}_3 + \bar{G}_2)(\bar{P}_4 + \bar{P}_3 + \bar{P}_2 + \bar{G}_1)(\bar{P}_4 + \bar{P}_3 + \bar{P}_2 + \bar{P}_1 + \bar{C}_0). \quad (2.207)$$

Din expresiile (2.204) ÷ (2.207) se obțin, în urma desfacerii parantezelor și a negării, expresiile:

$$C_1 = \overline{\bar{G}_1 \bar{P}_1 + \bar{G}_1 \bar{C}_0}, \quad (2.208)$$

$$C_2 = \overline{\bar{G}_2 \bar{P}_2 + \bar{G}_2 \bar{G}_1 \bar{P}_1 + \bar{G}_2 \bar{G}_1 \bar{C}_0}, \quad (2.209)$$

$$C_3 = \overline{\bar{G}_3 \bar{P}_3 + \bar{G}_3 \bar{G}_2 \bar{P}_2 + \bar{G}_3 \bar{G}_2 \bar{G}_1 \bar{P}_1 + \bar{G}_3 \bar{G}_2 \bar{G}_1 \bar{C}_0}, \quad (2.210)$$

$$C_4 = \overline{\bar{G}_4 \bar{P}_4 + \bar{G}_4 \bar{G}_3 \bar{P}_3 + \bar{G}_4 \bar{G}_3 \bar{G}_2 \bar{P}_2 + \bar{G}_4 \bar{G}_3 \bar{G}_2 \bar{G}_1 \bar{P}_1 + \bar{G}_4 \bar{G}_3 \bar{G}_2 \bar{G}_1 \bar{C}_0}. \quad (2.211)$$

Procedând în același mod cu expresiile (2.202) și (2.203), se obține:

$$\bar{G} = \bar{G}_4 \bar{P}_4 + \bar{G}_4 \bar{G}_3 \bar{P}_3 + \bar{G}_4 \bar{G}_3 \bar{G}_2 \bar{P}_2 + \bar{G}_4 \bar{G}_3 \bar{G}_2 \bar{G}_1 \quad (2.212)$$

și

$$\bar{P} = \bar{P}_4 + \bar{P}_3 + \bar{P}_2 + \bar{P}_1. \quad (2.213)$$

Un exemplu de circuit pentru anticiparea transportului, pentru patru poziții binare, este circuitul integrat MSI 74182 [34], a cărui schemă-bloc este prezentată în figura 2.116 (în logică pozitivă (a), respectiv negativă (b)). Bornele circuitului au următoarele semnificații: $G_0 \div G_3$ – transporturile generate pe grupe de câte 4 biți ale numerelor adunate; $P_0 \div P_3$ – transporturile propagate pe grupe de câte 4 biți ale numerelor adunate; C_n – intrare pentru transport; C_{n+x} – transportul spre al doilea grup de 4 biți; C_{n+y} – transportul spre al treilea grup de 4 biți; C_{n+z} – transportul spre al patrulea grup de 4 biți; G – transportul generat de cele 16 perechi de biți, adunate; P – transportul propagat prin cele 16 perechi de biți, adunate. Acest circuit realizează funcțiile C_{n+x} , C_{n+y} și C_{n+z} în forme (2.208) \div (2.210) (respectiv (2.204) \div (2.206)) și funcțiile G și P în forme (2.212) și (2.213) (respectiv (2.202) și (2.203)). Se realizează G și P pentru a se putea folosi același circuit în vederea anticipării transportului între grupele de sumatoare.

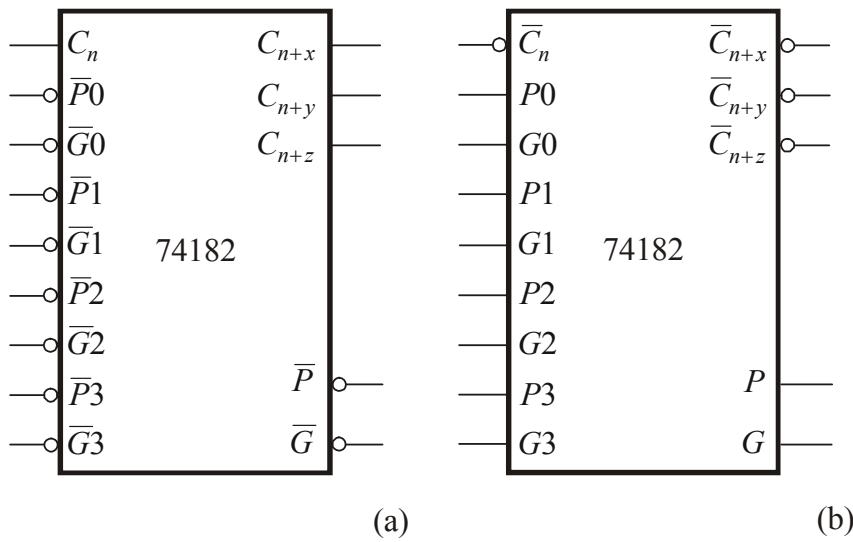


Fig.2.116 Schema-bloc a circuitului pentru generarea transportului anticipat 74182.

Dacă se urmărește o reducere suplimentară a timpului de adunare, se recurge la propagarea simultană a transportului în mai multe nivele, utilizând ieșirile G și P ale generatoarelor de transport anticipat ca intrări pentru generatoarele de transport anticipat din al doilea nivel.

În figura 2.117 este prezentată schema-bloc a unui sumator de 64 biți, realizat cu circuite sumatoare pentru 4 biți, cu anticiparea transportului între circuitele sumatoare și între grupele de câte patru circuite sumatoare [34]. Ca sumator pentru 4 biți poate fi utilizat circuitul 74181 (UAL – unitate aritmetică și logică și generator de funcții (v. §2.5.8)), comandat să efectueze adunări. Circuitul pentru anticiparea transportului (GTA) poate fi 74182.

În blocurile aritmetice ale sistemelor numerice, operația de scădere este „privită” ca o adunare, operația realizată fiind dictată de jocul semnelor. De asemenea, în cazul efectuării operațiilor de adunare și scădere, trebuie avute în vedere și cele trei moduri de reprezentare a numerelor binare cu semn: prin mărime și semn (cod direct), prin complement față de 2 (cod complementar), prin complement față de 1 (cod invers) (v. Anexa C).

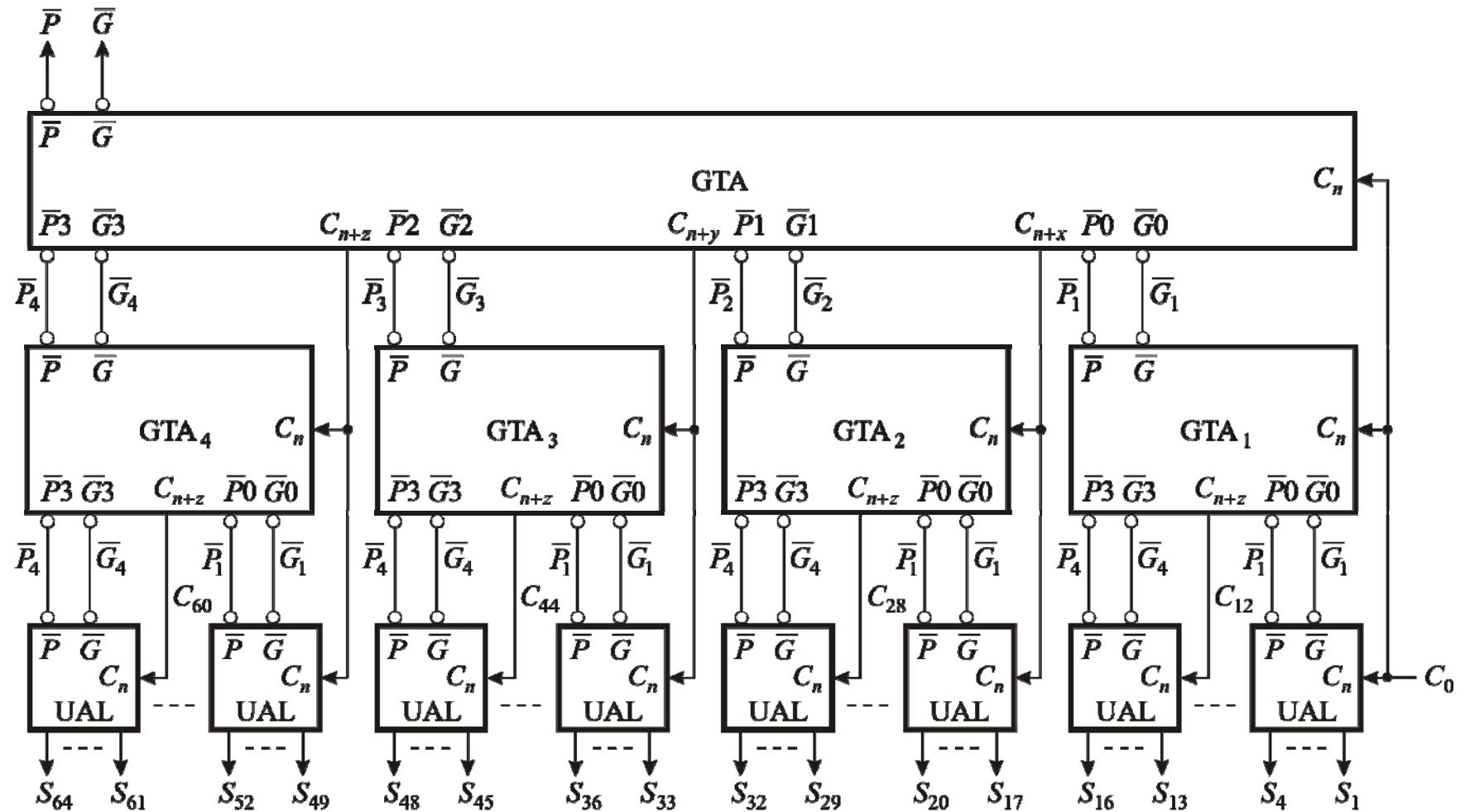


Fig.2.117 Schema-bloc a unui sumator de 64 biți cu anticiparea transportului.

Exemplu. Să se realizeze o unitate aritmetică pentru adunarea/scăderea numerelor binare cu semn reprezentate în cod direct, utilizându-se sumatorul binar integrat 7483 [19,26].

Trebuie deci să se efectueze operațiile aritmetice:

A plus B și A minus B ,

în care A și B sunt numere binare de trei biți (biții informaționali) la care se adaugă bitul de semn, adică:

$$A = A_4 A_3 A_2 A_1, \quad B = B_4 B_3 B_2 B_1,$$

cu A_4 și $B_4 = 0$, dacă numerele sunt pozitive și A_4 și $B_4 = 1$, dacă numerele sunt negative. Acest mod de reprezentare binară conține atât mărimea cât și semnul (v. Anexa C).

În continuare se vor prezenta, pentru un exemplu numeric cu cinci biți, regulile adunării respectiv scăderii numerelor binare reprezentate în cod direct:

a. Numerele au același semn:

A	$+ 7$	0.0111	$- 7$	1.0111	
B	$+ 6$	0.0110	$- 6$	1.0110	
<hr/> A plus B		<hr/> +13	0.1101	<hr/> -13	1.1101

Se adună numerele și se copiază semnele.

b. Numerele au semne diferite:

$$A > B$$

A	$+7$	0.0111	-7	1.0111
B	-6	1.0110	$+6$	0.0110
<hr/> A plus B		<hr/> +1	<hr/> -1	

Tab.2.38 Semnul, rezultatul și transportul pentru A plus B .

A_4	B_4	Posibi-lități	Semnul rezultat	Transp. global $C_3 = C_0$	Suma
0	0	$A \gtrless B$	0	0	$S = A + B$
0	1	$A > B$	0	1	$S = A + \bar{B}$
		$A \leq B$	1	0	$\bar{S} = A + \bar{B}$
1	0	$A > B$	1	1	$S = A + \bar{B}$
		$A \leq B$	0	0	$\bar{S} = A + \bar{B}$
1	1	$A \gtrless B$	1	0	$S = A + B$

Pentru scăderea numerelor binare reprezentate în cod direct, procedeul uzual este de a se schimba semnul scăzătorului și de a se efectua adunarea.

În mod asemănător au fost analizate cazurile care pot să apară la scădere, în tabelul 2.39.

 Tab.2.39 Semnul, rezultatul și transportul pentru A minus B .

A_4	B_4	Posibi-lități	Semnul rezultat	Transp. global $C_3 = C_0$	Suma
0	0	$A > B$	0	1	$S = A + \bar{B}$
		$A \leq B$	1	0	$\bar{S} = A + \bar{B}$
0	1	$A \gtrless B$	0	0	$S = A + B$
		$A \gtrless B$	1	0	$S = A + B$
1	1	$A > B$	1	1	$S = A + \bar{B}$
		$A \leq B$	0	0	$\bar{S} = A + \bar{B}$

Din cele două tabele rezultă că, în funcție de semnele celor două numere (biții A_4 și B_4), B trebuie introdus fie direct, fie complementat iar S rezultă, de asemenea, fie directă, fie complementată.

La circuitul 7483 (c.i. sumator de 4 biți, cu calculul anticipat al transportului $C4$), transportul $C3$ nu este accesibil în exterior. Acest inconvenient poate fi depășit dacă, în expresia transportului $C4$ (v. ec. (2.182)):

$$C4 = A4B4 + (A4 + B4)C3, \quad (2.214)$$

se conectează borna $A4$ la 1 logic și borna $B4$ la 0 logic lucrul posibil deoarece biții de rangul 4 ai sumatorului nu sunt folosiți în acest caz de adunare cu numai 3 biți informaționali (biții A_3, A_2, A_1 și B_3, B_2, B_1). În acest mod se obține $C4 = C3$, adică $C3$ va fi accesibil la borna $C4$.

Cu aceste noțiuni definite se poate concepe, destul de rapid, o unitate aritmetică capabilă să adune și să scadă numere binare cu semn, reprezentate în cod direct.

Pe baza tabelelor 2.38 și 2.39 s-au întocmit diagramele Karnaugh din figurile 2.118 și 2.119. Diagramele Karnaugh din figura 2.118 permit obținerea expresiei funcției s care dă semnul rezultatului, sumă sau diferență.

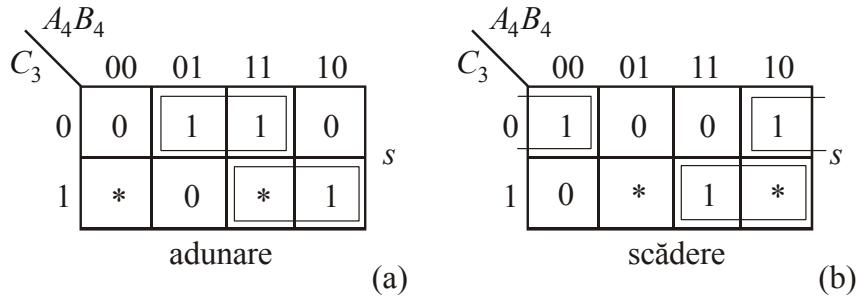


Fig.2.118 Diagramale Karnaugh pentru semn la adunare (a) și scădere (b).

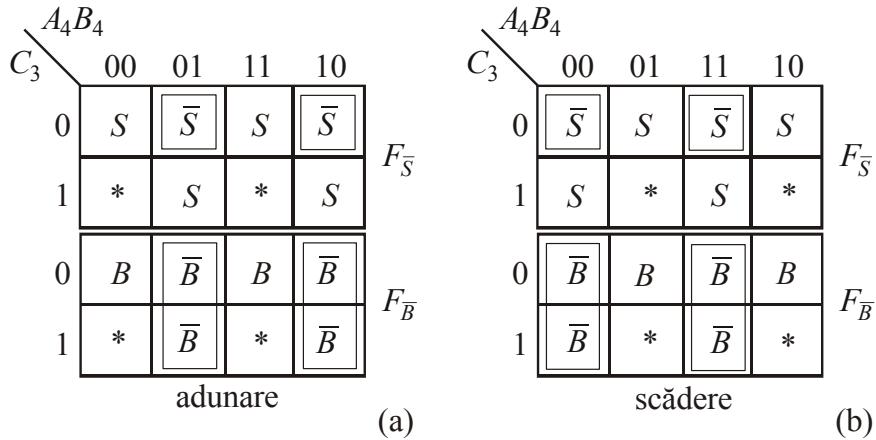


Fig.2.119 Diagramale Karnaugh pentru complementarea lui B și S , la adunare (a) și scădere (b).

Funcția pentru semnul rezultat în urma adunării are expresia:

$$s = B_4 \bar{C}_3 + A_4 C_3 = \overline{\overline{B_4} \overline{\bar{C}_3}} \cdot \overline{\overline{A_4} \overline{C_3}}, \quad (2.215)$$

iar pentru scădere:

$$s = \bar{B}_4 \bar{C}_3 + A_4 C_3 = \overline{\overline{\bar{B}_4} \overline{\bar{C}_3}} \cdot \overline{\overline{A_4} \overline{C_3}}. \quad (2.216)$$

Circuitul logic care realizează funcțiile de mai sus este prezentat în figura 2.120,a. Pentru implementare s-au folosit porți logice ȘI-NU și INVERSOARE.

De asemenea, din diagramele Karnaugh prezентate în figura 2.119 se pot determina funcțiile $F_{\bar{B}}$ și $F_{\bar{S}}$ care exprimă complementarea numărului binar B , respectiv a sumei S . Astfel, din diagrama din figura 2.119,a, se observă că, în cazul adunării, B trebuie complementat în conformitate cu funcția:

$$F_{\bar{B}} = \bar{A}_4 B_4 + A_4 \bar{B}_4 = A_4 \oplus B_4, \quad (2.217)$$

iar suma S trebuie complementată în conformitate cu funcția:

$$F_{\bar{S}} = \overline{A_4}B_4\overline{C}_3 + A_4\overline{B}_4\overline{C}_3 = (A_4 \oplus B_4)\overline{C}_3. \quad (2.218)$$

În cazul scăderii, B trebuie complementat când (v. fig. 2.119,b):

$$F_{\bar{B}} = \overline{A_4}\overline{B}_4 + A_4B_4 = \overline{A_4 \oplus B_4} = A_4 \oplus \overline{B}_4, \quad (2.219)$$

iar S , când:

$$F_{\bar{S}} = \overline{A_4}B_4\overline{C}_3 + A_4B_4\overline{C}_3 = (A_4 \oplus \overline{B}_4)\overline{C}_3. \quad (2.220)$$

Circuitul care generează aceste funcții este prezentat în figura 2.120,b.

Din cele de mai sus rezultă că pentru B_4 , B și S trebuie introduse niște inversoare suplimentare comandate. În figura 2.120,c este prezentat un astfel de inversor (cel mai simplu astfel de inversor este poarta logică SAU-EXCLUSIV) pentru alegerea modului de lucru. Într-adevăr, când:

$M = 0$, $F = B_4$, circuitul efectuează adunări;

$M = 1$, $F = \overline{B}_4$, circuitul efectuează scăderi.

Inversorul comandat prezentat în figura 2.120,c înlocuiește și inversoarele din schemele logice (a) și (b) din aceeași figură.

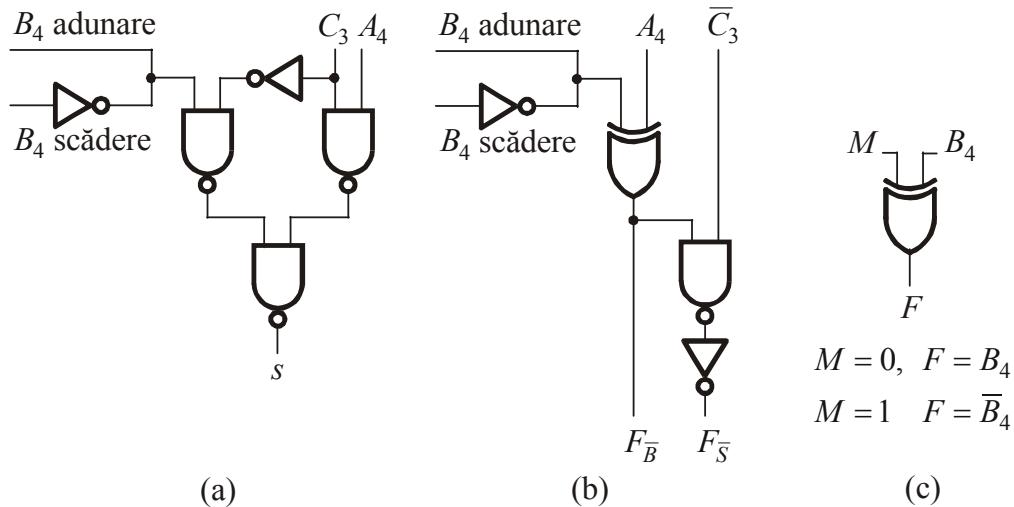
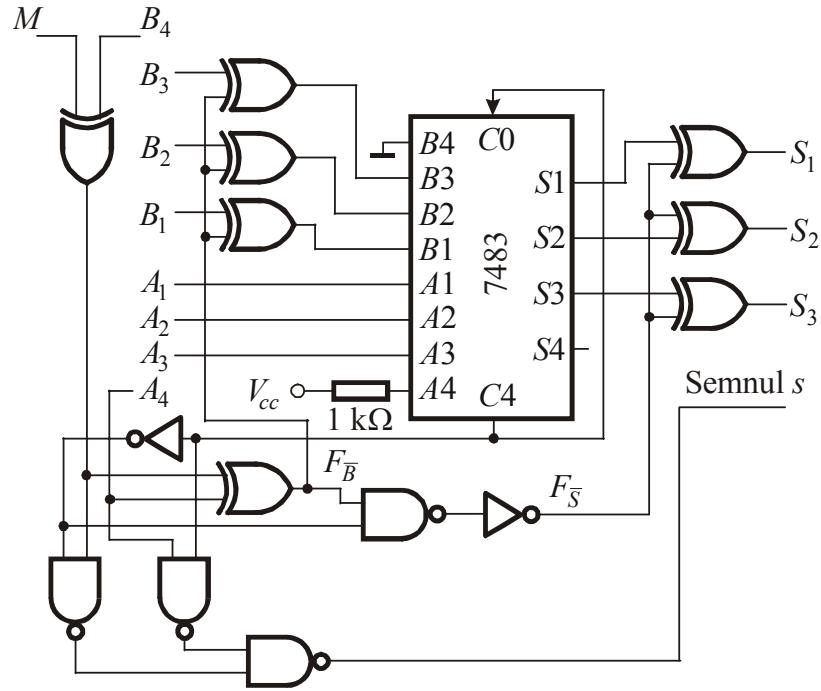


Fig.2.120 Schema logică pentru semn (a), pentru complementarea lui B și S (b) și pentru controlul de mod (c).

Schema logică completă, a unității aritmetice proiectate, este prezentată în figura 2.121.

Trebuie menționat că există încă multe posibilități de a se efectua operații aritmetice folosind codul complementar, codul invers sau alte coduri decât cel binar, pentru reprezentarea numerelor.


 Fig.2.121 Unitate aritmetică pentru adunare ($M = 0$) și scădere ($M = 1$).

2.5.7 Circuite generatoare de funcții

Circuitele generatoare de funcții sunt proiectate astfel încât să poată realiza mai multe funcții logice din care se selecteză funcția dorită, la ieșirea circuitului, prin intermediul unor intrări de selecție [22,23].

După cum se cunoaște, există 4 funcții logice de o singură variabilă: funcția constantă 0, funcția identitate, funcția negație și funcția constantă 1. Circuitul generator de funcții logice de o singură variabilă permite efectuarea concomitentă a același funcții pentru 4 biți (v. fig. 2.122), conform tabelului 2.40.

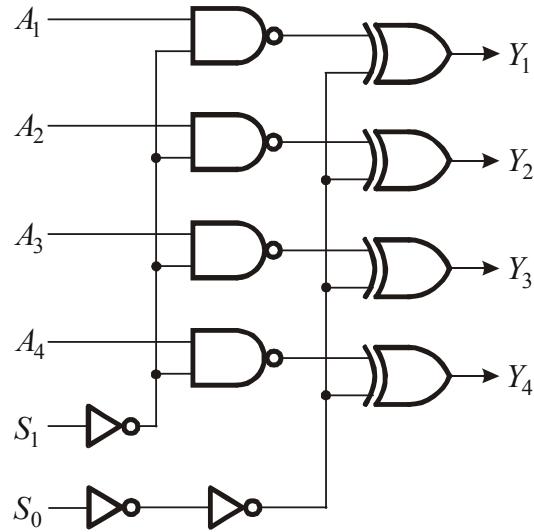


Fig.2.122 Schema logică a circuitului generator de funcții de o variabilă.

Tab.2.40 Tabelul de adevăr pentru circuitul generator de funcții de o variabilă.

Intrări selecție		Ieșiri				Funcția realizată
S_1	S_0	Y_1	Y_2	Y_3	Y_4	
0	0	\bar{A}_1	\bar{A}_2	\bar{A}_3	\bar{A}_4	negație
0	1	A_1	A_2	A_3	A_4	identitate
1	0	1	1	1	1	constantă 1
1	1	0	0	0	0	constantă 0

Circuitul descris poate fi utilizat în calitate de circuit de complementare față de 1 pentru numere de 4 biți.

Se consideră, în continuare, un circuit de comutare combinatorial care generează toate funcțiile logice de două variabile, notate cu A și B . Există 16 funcții logice de două variabile, prin urmare sunt necesare patru intrări de selecție, notate cu S_0, S_1, S_2 și S_3 . Funcția dorită este selectată în modul indicat în tabelul 2.41.

Tab.2.41 Tabelul de adevăr al unui circuit generator de funcții de două variabile.

Intrări selecție				Funcția logică	Intrări selecție				Funcția logică
S_3	S_2	S_1	S_0		S_3	S_2	S_1	S_0	
0	0	0	0	$F = \bar{A}$	1	0	0	0	$F = \bar{A} + B$
0	0	0	1	$F = \bar{A}\bar{B}$	1	0	0	1	$F = AB + \bar{A}\bar{B}$
0	0	1	0	$F = \bar{A}B$	1	0	1	0	$F = B$
0	0	1	1	$F = 0$	1	0	1	1	$F = AB$
0	1	0	0	$F = \bar{A} + \bar{B}$	1	1	0	0	$F = 1$
0	1	0	1	$F = \bar{B}$	1	1	0	1	$F = A + \bar{B}$
0	1	1	0	$F = A\bar{B} + \bar{A}B$	1	1	1	0	$F = A + B$
0	1	1	1	$F = A\bar{B}$	1	1	1	1	$F = A$

Pe baza tabelului 2.41 se completează diagrama de minimizare a funcției F , din figura 2.123. Din această diagramă se obține expresia funcției:

$$F = ABS_3 + A\bar{B}S_2 + \bar{A}\bar{B}\bar{S}_1 + \bar{A}B\bar{S}_0, \quad (2.221)$$

pornind de la care se poate obține schema logică a circuitului generator de funcții, realizată cu elemente logice ȘI-NU.

2.5.8 Unitatea aritmetică și logică

Odată cu creșterea densității de integrare a circuitelor digitale, funcțiile aritmetice/logice se realizează în circuite tot mai complexe, care execută, pe baza unor comenzi, diverse funcții de tip aritmetic sau logic. Aceste circuite se pot utiliza ca atare în sistemele numerice complexe sau se regăsesc, ca elemente componente, în unitățile de prelucrare ale sistemelor de calcul.

$BS_1 S_0 \backslash AS_3 S_2$	000	001	011	010	110	111	101	100	F
000	1	1	1	1	0	1	1	0	
001	1	1	1	1	0	1	1	0	
011	0	0	0	0	0	1	1	0	
010	0	0	0	0	0	1	1	0	
110	1	1	1	1	1	1	0	0	
111	0	0	0	0	1	1	0	0	
101	0	0	0	0	1	1	0	0	
100	1	1	1	1	1	1	0	0	
	\bar{Y}				X				

 Fig.2.123 Diagrama Karnaugh pentru funcția logică F din tabelul 2.41.

În practică, circuitele generatoare de funcții logice, descrise mai sus (v. §2.5.7), se realizează împreună cu circuitele care, la rândul lor, realizează funcțiile aritmetice asupra variabilelor de intrare, constituind aşa numitele *unități aritmetice și logice* (UAL-uri) [2, 15, 22, 23, 26, 28, 30]. Aceste circuite multifuncționale constituie unele dintre cele mai complexe module din categoria circuitelor integrate MSI. UAL-ul este un CLC complex, cu următoarele funcții:

- execută o operație aritmetică/logică asupra unuia sau a doi operanzi de intrare cu un anumit număr de biți;
- furnizează indicații asupra rezultatului.

În continuare, se vor prezenta câteva considerații teoretice care vor permite înțelegerea modului de funcționare a unui UAL [23]. Astfel, funcția (2.221) se poate exprima sub forma:

$$F = A(\bar{B}S_2 + BS_3) + \bar{A}(\bar{B}\bar{S}_1 + B\bar{S}_0). \quad (2.222)$$

Avându-se în vedere relația:

$$Aa + \bar{A}b = Aa \oplus \bar{A}b, \quad (2.223)$$

unde a și b sunt expresii logice, expresia (2.222) se mai poate scrie și sub forma:

$$F = A(\bar{B}S_2 + BS_3) \oplus \bar{A}(\bar{B}\bar{S}_1 + B\bar{S}_0). \quad (2.224)$$

Ținând cont de proprietățile, de mai jos, ale operației SAU-EXCLUSIV:

$$\bar{a} \oplus b = a \oplus \bar{b} = \overline{a \oplus b}, \quad (2.225)$$

$$\bar{a} \oplus \bar{b} = a \oplus b, \quad (2.226)$$

$$a \oplus 1 = \bar{a}, \quad (2.227)$$

unde a și b sunt expresii logice, funcția (2.224) se poate scrie sub forma:

$$\begin{aligned} F &= A(\bar{B}S_2 + BS_3) \oplus \overline{A(\bar{B}\bar{S}_1 + B\bar{S}_0)} \oplus C' = \\ &= (A\bar{B}S_2 + ABS_3) \oplus (A + \bar{B}S_1 + BS_0) \oplus C' = X \oplus Y \oplus C', \end{aligned} \quad (2.228)$$

unde X și \bar{Y} sunt expresiile reprezentate pe jumătatea dreaptă respectiv jumătatea stângă a diagramei de minimizare din figura 2.123, iar $C' = 1$. Forma (2.228) a funcției F prezintă mai multe avantaje. În primul rând variabilele de selecție apar în această expresie numai sub formă directă (nenegată). În al doilea rând, faptul că ea conține operatorul SAU-EXCLUSIV, echivalent cu suma modulo 2, permite realizarea operațiilor aritmetice asupra operanzilor X , Y și C' . Astfel, dacă se ia $C' = C$, unde C este transportul pentru rangul considerat, iar variabilele de selecție au valorile $S_3 = 1$, $S_2 = 0$, $S_1 = 0$, $S_0 = 1$, funcția F devine, în urma înlocuirii acestor valori în expresia (2.228):

$$F = AB \oplus (A + B) \oplus C. \quad (2.229)$$

Expresia (2.229), care se mai poate scrie și sub forma $F = A \oplus B \oplus C$, este o formă a sumei pentru un rang. Se observă că, în acest caz, expresia X devine condiția de generare a transportului în rangul respectiv (G), iar expresia Y , condiția de propagare a transportului peste rangul respectiv (P). Aceasta este un alt avantaj al acestei forme de exprimare a funcției F , deoarece funcțiile auxiliare G și P pot fi utilizate și pentru generarea transportului anticipat, în cazul când se dorește realizarea operațiilor aritmetice. Un ultim avantaj al acestei forme de exprimare a funcției F îl constituie și faptul că se pot folosi în circuitul care o realizează elemente logice ȘI-SAU-NU, deoarece, în conformitate cu proprietatea (2.226) expresia (2.228), se poate scrie sub forma:

$$F = \overline{(ABS_3 + A\bar{B}S_2)} \oplus \overline{(A + \bar{B}S_1 + BS_0)} \oplus C' = \bar{X} \oplus \bar{Y} \oplus C'. \quad (2.230)$$

Schema logică a circuitului care realizează funcția F , pe baza expresiei (2.230) este dată în figura 2.124. Pentru $S_3 = 1$, $S_2 = 0$, $S_1 = 0$, $S_0 = 1$, circuitul din figura 2.124 realizează funcția:

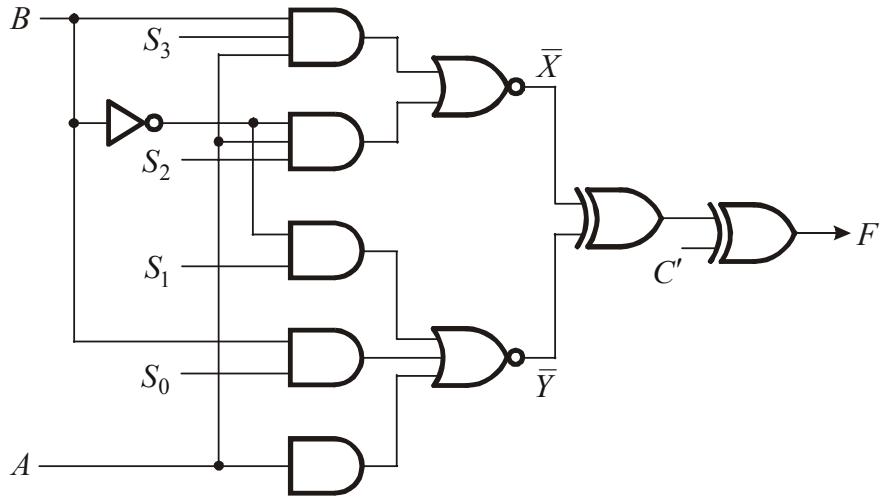
$$F = A \oplus B \oplus C'. \quad (2.231)$$

Expresia (2.231) reprezintă suma aritmetică a variabilelor binare A , B și C' , dacă se consideră C' transportul pentru rangul respectiv, notat cu C :

$$F = A \text{ plus } B \text{ plus } C, \quad (2.232)$$

iar dacă se ia $C' = 1$, ea reprezintă funcția logică denumită COINCIDENȚĂ:

$$F = A \oplus B \oplus 1 = \overline{A \oplus B}. \quad (2.233)$$


 Fig.2.124 Schema logică a circuitului care realizează funcția F .

Modul de lucru al circuitului aritmetic, respectiv logic, se stabilește prin intermediul unei variabile de comandă, notată cu M . Se face convenția ca pentru $M = 1$ să se realizeze funcția logică, iar pentru $M = 0$, funcția aritmetică. În această situație se poate scrie următoarea expresie:

$$C' = M + \overline{M}C, \text{ sau } C' = \overline{M}\overline{C}. \quad (2.234)$$

Unitățile aritmetice și logice se realizează, sub formă integrată, pentru mai multe ranguri binare, de exemplu patru. Se observă că pentru valorile considerate ale variabilelor de selecție, expresiile \bar{X} și \bar{Y} reprezintă negația condițiilor de generare, respectiv de propagare a transportului. Prin urmare, pentru ca variabila C' , din expresia (2.231) să reprezinte, în cazul efectuării operației aritmetice de însumare, transportul pentru un anumit rang i , aceasta se poate exprima, pe baza relațiilor (2.194) ÷ (2.197), sub forma:

$$C'_0 = \overline{M}\overline{C}_0, \quad (2.235)$$

$$C'_1 = \overline{M}\overline{Y}_1 + \overline{M}\overline{X}_1\overline{C}_0, \quad (2.236)$$

$$C'_2 = \overline{M}\overline{Y}_2 + \overline{M}\overline{X}_2\overline{Y}_1 + \overline{M}\overline{X}_2\overline{X}_1\overline{C}_0, \quad (2.237)$$

$$C'_3 = \overline{M}\overline{Y}_3 + \overline{M}\overline{X}_3\overline{Y}_2 + \overline{M}\overline{X}_3\overline{X}_2\overline{Y}_1 + \overline{M}\overline{X}_3\overline{X}_2\overline{X}_1\overline{C}_0, \quad (2.238)$$

$$C'_4 = \overline{M}\overline{Y}_4 + \overline{M}\overline{X}_4\overline{Y}_3 + \overline{M}\overline{X}_4\overline{X}_3\overline{Y}_2 + \overline{M}\overline{X}_4\overline{X}_3\overline{X}_2\overline{Y}_1 + \overline{M}\overline{X}_4\overline{X}_3\overline{X}_2\overline{X}_1\overline{C}_0, \quad (2.239)$$

unde C_0 reprezintă transportul inițial. Din relațiile (2.235) ÷ (2.239), rezultă că la ieșirea unui asemenea circuit se obține, atunci când $M = 0$:

$$F_i = A_i \oplus B_i \oplus C_{i-1}, \quad (2.240)$$

unde C_{i-1} este transportul pentru rangul i ($i = 1, \dots, 4$), respectiv suma F a celor două numere de câte patru cifre binare, A și B , cu transportul inițial C_0 :

$$F = A \text{plus} B \text{plus} C_0. \quad (2.241)$$

În mod asemănător se pot obține operațiile aritmetice și logice și pentru celelalte combinații de valori ale variabilelor de selecție.

Un exemplu de circuit integrat realizat în modul prezentat mai sus este unitatea aritmetică și logică/generatorul de funcții de patru biți 74181, produs de firma Texas Instruments [34] și a cărei schemă-bloc este prezentată în figura 2.125 (s-a reprezentat atât cazul în care semnalele de date sunt active pe nivel 0 logic, (a), cât și cazul în care acestea sunt active pe nivelul 1 logic, (b)). Intrările și ieșirile UAL-ului au următoarele semnificații:

- intrările operanzilor $(A, B) = (A_0 \div A_3, B_0 \div B_3)$;
- intrarea pentru bitul de transport C_n ;
- ieșirea funcției $F = (F_0 \div F_3)$;
- ieșirea pentru bitul de transport C_{n+4} ;
- ieșirea indicatoare a stării de egalitate a operanzilor $(A = B)$;
- ieșiri pentru transport anticipat, P, G ;
- comenzi:
 - selectarea funcției $S = (S_0 \div S_3)$;
 - comanda „modului“ M : $M = 0 = \text{mod ,aritmetic“},$
 $M = 1 = \text{mod ,logic“}$.

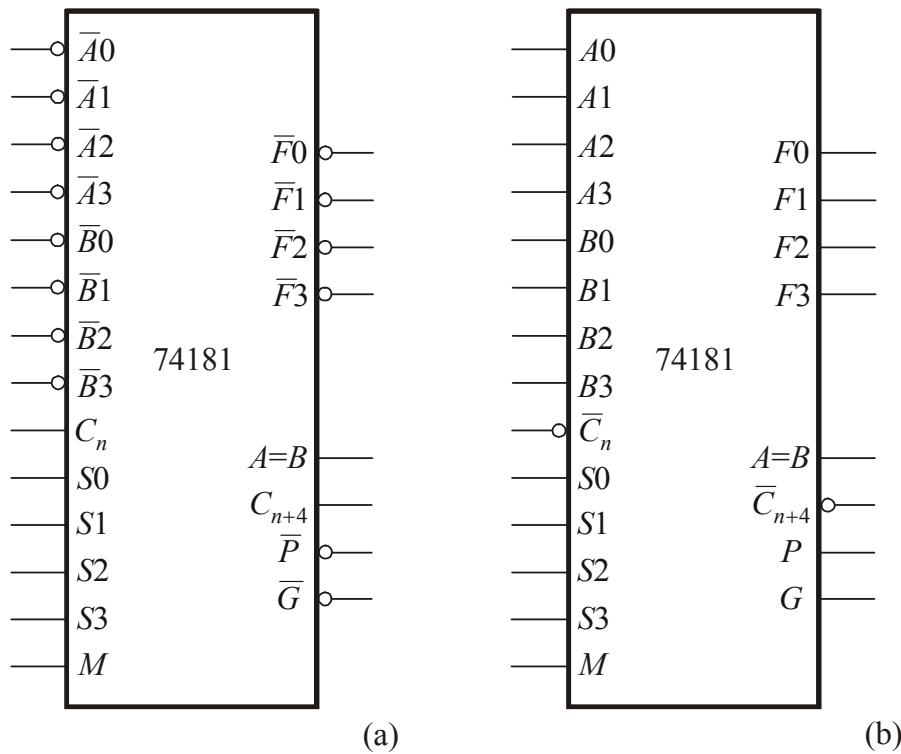


Fig.2.125 Unitate aritmetică și logică (UAL) de 4 biți.

Tabelele 2.42,a și b descriu complet funcționarea circuitului atât pentru cazul în care semnalele sunt active pe nivelul 0 logic (a) cât și în cazul în care acestea sunt active pe nivelul 1 logic (b).

CIRCUITE LOGICE COMBINATORIALE

Tab.2.42,a Tabelul de adevăr pentru o unitate aritmetică și logică de 4 biți (date active pe zero logic).

Selectia				Date active pe zero logic		
				$M = 1$: funcții logice	$M = 0$: operații aritmetice	
$C_n = 0$ (fără transport)		$C_n = 1$ (cu transport)				
S_3	S_2	S_1	S_0			
0	0	0	0	$F = \bar{A}$	$F = A \text{ MINUS } 1$	$F = A$
0	0	0	1	$F = \bar{A}B$	$F = AB \text{ MINUS } 1$	$F = AB$
0	0	1	0	$F = \bar{A} + B$	$F = \bar{A}\bar{B} \text{ MINUS } 1$	$F = A\bar{B}$
0	0	1	1	$F = 1$ COMPLEMENT)	$F = \text{MINUS } 1$ (2)	$F = \text{ZERO}$
0	1	0	0	$F = \overline{A+B}$	$F = A \text{ PLUS } (A+\bar{B})$	$F = A \text{ PLUS } (A+\bar{B}) \text{ PLUS }$
0	1	0	1	$F = \bar{B}$	$F = AB \text{ PLUS } (A+\bar{B})$	$F = AB \text{ PLUS } (A+\bar{B}) \text{ PLUS }$
0	1	1	0	$F = \overline{A \oplus B}$ MINUS 1	$F = A \text{ MINUS } B$	$F = A \text{ MINUS } B$
0	1	1	1	$F = A + \bar{B}$	$F = A + \bar{B}$	$F = (A + \bar{B}) \text{ PLUS } 1$
1	0	0	0	$F = \bar{A}B$	$F = A \text{ PLUS } (A+B)$	$F = A \text{ PLUS } (A+B) \text{ PLUS } 1$
1	0	0	1	$F = A \oplus B$	$F = A \text{ PLUS } B$	$F = A \text{ PLUS } B \text{ PLUS } 1$
1	0	1	0	$F = B$	$F = \bar{A}\bar{B} \text{ PLUS } (A+B)$	$F = A\bar{B} \text{ PLUS } (A+B) \text{ PLUS }$
1	0	1	1	$F = A + B$	$F = A + B$	$F = (A + B) \text{ PLUS } 1$
1	1	0	0	$F = 0$	$F = A \text{ PLUS } A *$	$F = A \text{ PLUS } A \text{ PLUS } 1$
1	1	0	1	$F = A\bar{B}$	$F = AB \text{ PLUS } A$	$F = AB \text{ PLUS } A \text{ PLUS } 1$
1	1	1	0	$F = AB$	$F = \bar{A}\bar{B} \text{ PLUS } A$	$F = A\bar{B} \text{ PLUS } A \text{ PLUS } 1$
1	1	1	1	$F = A$	$F = A$	$F = A \text{ PLUS } 1$

a)

*) Fiecare bit este deplasat în următoarea poziție semnificativă.

ALTE TIPURI DE CIRCUITE LOGICE COMBINATORIALE

Tab.2.42,b Tabelul de adevăr pentru o unitate aritmetică și logică de 4 biți (date active pe unu logic).

Selectia				$M = 1$: funcții logice	Date active pe unu logic	
S_3	S_2	S_1	S_0		$\bar{C}_n = 1$ (fără transport)	$\bar{C}_n = 0$ (cu transport)
0	0	0	0	$F = \bar{A}$	$F = A$	$F = A$ PLUS 1
0	0	0	1	$F = \bar{A} + B$	$F = A + B$	$F = (A + B)$ PLUS 1
0	0	1	0	$F = \bar{A}B$	$F = A + \bar{B}$	$F = (A + \bar{B})$ PLUS 1
0	0	1	1	$F = 0$	$F = \text{MINUS } 1$ (2 COMPLEMENT)	$F = \text{ZERO}$
0	1	0	0	$F = \bar{A}\bar{B}$	$F = A$ PLUS $A\bar{B}$	$F = A$ PLUS $A\bar{B}$ PLUS 1
0	1	0	1	$F = \bar{B}$	$F = (A + B)$ PLUS $A\bar{B}$	$F = (A + B)$ PLUS $A\bar{B}$ PLUS 1
0	1	1	0	$F = A \oplus B$	$F = A$ MINUS B MINUS 1	$F = A$ MINUS B
0	1	1	1	$F = A\bar{B}$	$F = A\bar{B}$ MINUS 1	$F = A\bar{B}$
1	0	0	0	$F = \bar{A} + B$	$F = A$ PLUS AB	$F = A$ PLUS AB PLUS 1
1	0	0	1	$F = \bar{A} \oplus B$	$F = A$ PLUS B	$F = A$ PLUS B PLUS 1
1	0	1	0	$F = B$	$F = (A + \bar{B})$ PLUS AB	$F = (A + \bar{B})$ PLUS AB PLUS 1
1	0	1	1	$F = AB$	$F = AB$ MINUS 1	$F = AB$
1	1	0	0	$F = 1$	$F = A$ PLUS A *)	$F = A$ PLUS A PLUS 1
1	1	0	1	$F = A + \bar{B}$	$F = (A + B)$ PLUS A	$F = (A + B)$ PLUS A PLUS 1
1	1	1	0	$F = A + B$	$F = (A + \bar{B})$ PLUS A	$F = (A + \bar{B})$ PLUS A PLUS 1
1	1	1	1	$F = A$	$F = A$ MINUS 1	$F = A$

*) Fiecare bit este deplasat în următoarea poziție semnificativă.

b)

Circuitul integrat 74181 va avea datele active pe nivelul 1 logic dacă marcarea pinilor este interpretată după cum urmează (v. și fig. 2.125):

Date active pe nivelul 0 logic (Tabelul 2.42,a)	$\bar{A}0$	$\bar{B}0$	$\bar{A}1$	$\bar{B}1$	$\bar{A}2$	$\bar{B}2$	$\bar{A}3$	$\bar{B}3$
	$\bar{F}0$	$\bar{F}1$	$\bar{F}2$	$\bar{F}3$	C_n	C_{n+4}	\bar{P}	\bar{G}
Date active pe nivelul 1 logic (Tabelul 2.42,b)	$A0$	$B0$	$A1$	$B1$	$A2$	$B2$	$A3$	$B3$
	$F0$	$F1$	$F2$	$F3$	\bar{C}_n	\bar{C}_{n+4}	P	G

Acest circuit este astfel proiectat încât să genereze transportul C_{n+4} ¹⁶:

$$C_{n+4} = \overline{Y_3 + X_3\bar{Y}_2 + X_3\bar{X}_2\bar{Y}_1 + X_3\bar{X}_2\bar{X}_1\bar{Y}_0 + X_3\bar{X}_2\bar{X}_1\bar{X}_0\bar{C}_n}, \quad (2.242)$$

condiția de generare a transportului în cele patru ranguri:

$$\bar{G} = \bar{X} = \bar{X}_3\bar{X}_2\bar{X}_1\bar{X}_0 \quad (2.243)$$

și condiția de propagare a transportului peste cele patru ranguri:

$$\bar{P} = \bar{Y} = \bar{Y}_3 + \bar{X}_3\bar{Y}_2 + \bar{X}_3\bar{X}_2\bar{Y}_1 + \bar{X}_3\bar{X}_2\bar{X}_1\bar{Y}_0. \quad (2.244)$$

Ieșirea $A = B$ a UAL-ului permite detectarea situațiilor de egalitate a operanzilor, atunci când $S_3S_2S_1S_0 = 0110$, UAL-ul este în mod „aritmetic“ ($M = 0$), cu funcția de scădere și $C_n = 1$.

UAL-ul poate realiza, de asemenea și funcția de deplasare la stânga, $F = A$ PLUS B ($S_3S_2S_1S_0 = 1100$), cu sau fără transport, ceea ce permite utilizarea sa în circuite pentru înmulțire, respectiv împărțire.

UAL-urile pot fi conectate în paralel, pentru a se forma un UAL pentru operanzi de lungimi mai mari; în acest caz C_n este transportul de la UAL-ul de rang inferior iar C_{n+4} transport de ieșire spre UAL-ul de rang imediat superior. În cazul utilizării unui circuit extern de transport anticipat (v. fig. 2.117), atunci intrarea C_n este furnizată de GTA, iar P și G se conectează la GTA.

2.5.9 Circuite multiplicatoare

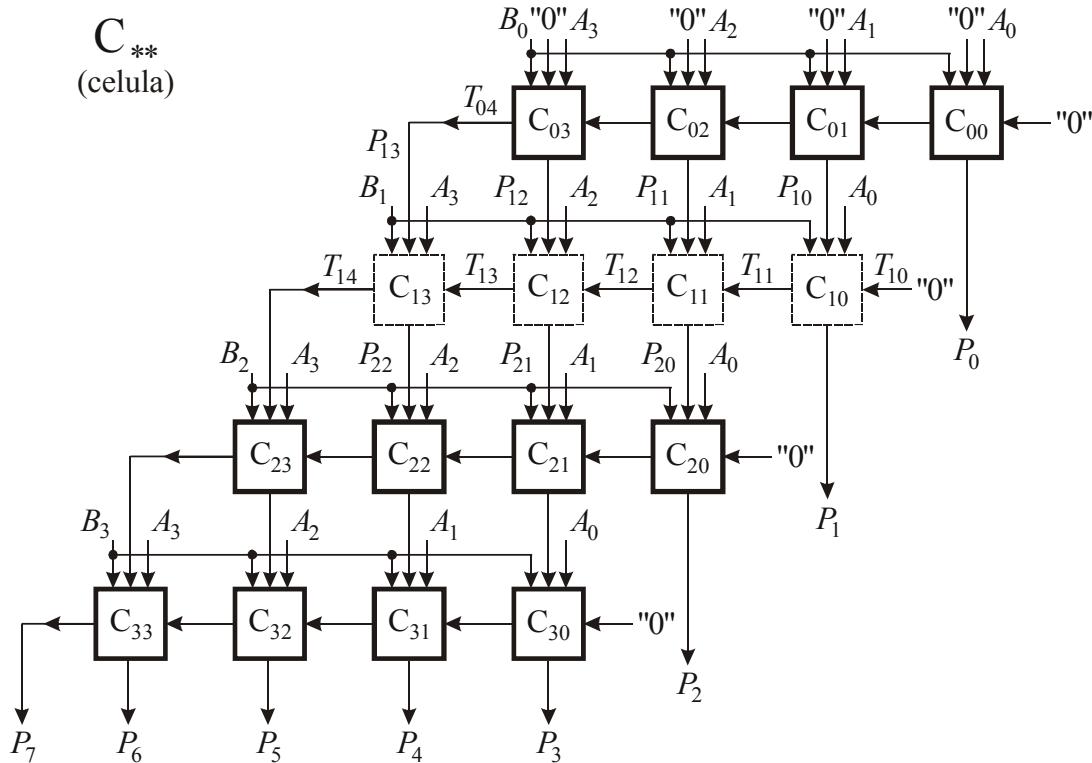
Circuitele pentru înmulțirea rapidă a numerelor binare se realizează, de obicei, pe principiul circuitelor iterative (v. §2.5.6) [2,23,26,28,30]. De exemplu, în cazul înmulțirii a două numere binare cu patru poziții A_0, A_1, A_2 și A_3 și B_0, B_1, B_2 și B_3 , se obține un produs cu 8 poziții binare $P_0, P_1, P_2, P_3, P_4, P_5, P_6$ și P_7 . Simbolic, operația de înmulțire are loc conform regulii următoare:

¹⁶ Deoarece la circuitul integrat 74181 cifrele celor doi operanzi, A și B , sunt notate cu $A0\div A3$, respectiv $B0\div B3$, indicii zecimali ai expresiilor X și Y sunt mai mici cu o unitate decât în cazul general tratat (v., de exemplu, ec. (2.239)).

	A_3	A_2	A_1	A_0	\times
	B_3	B_2	B_1	B_0	
		$A_3 B_0$	$A_2 B_0$	$A_1 B_0$	$A_0 B_0$
		$A_3 B_1$	$A_2 B_1$	$A_1 B_1$	$A_0 B_1$
	$A_3 B_2$	$A_2 B_2$	$A_1 B_2$	$A_0 B_2$	
	$A_3 B_3$	$A_2 B_3$	$A_1 B_3$	$A_0 B_3$	
P_7	P_6	P_5	P_4	P_3	P_2
					P_1
					P_0

Însumarea produselor $A_i B_j$ se face ținând cont și de semnalul de transport ce poate apărea la fiecare sumă binară.

Realizarea celor opt funcții P_i ($i = 0, \dots, 7$), direct, ca funcții de cele 8 variabile ($A_0 \div A_3$) și ($B_0 \div B_3$) este destul de dificilă. Din acest motiv se face însumarea deînmulțitului la produsul parțial, fie combinațional, ca un circuit iterativ alcătuit din mai multe sumatoare paralele, fie secvențial cu un singur sumator paralel. Se consideră primul caz, iar schema logică a circuitului iterativ care realizează această înmulțire este prezentată în figura 2.126.



$$P_{0,i} = 0, \text{ unde } i = 0 \div 3 \text{ (produsul parțial inițial este nul);}$$

$$T_{j,0} = 0, \text{ unde } j = 0 \div 3 \text{ (transportul inițial este nul);}$$

$T_{j,i_{\max}+1} = P_{j+1,i_{\max}}$ (produsul parțial aplicat celulei de rang maxim de pe un rând este egal cu transportul de la celula de rang maxim de pe rândul precedent).

Fig.2.126 Circuit iterativ pentru înmulțirea a două numere cu câte 4 cifre binare.

Primul rând de celule, din acest circuit, însumează produsul deînmulțitului ($A_3A_2A_1A_0$), cu cifra cea mai puțin semnificativă a înmulțitorului, B_0 , la produsul parțial inițial (0) și generează primul produs parțial. Rândul următor de celule adună la acest produs parțial deînmulțitul înmulțit cu următoarea cifră a înmulțitorului (B_1) și generează următorul produs parțial, și.a.m.d. Pentru a se projecța un multiplicator/înmulțitor combinațional pentru cuvinte de n biți va trebui să se interconecteze n^2 celule, într-o matrice bidimensională sugerată de algoritmul de înmulțire binară a două cuvinte de n biți.

Fiecare celulă din cadrul circuitului iterativ din figura 2.126 este un circuit care realizează înmulțirea a două cuvinte de un bit (înmulțitorul/multiplicatorul elementar). Structura acestui circuit trebuie să satisfacă trei cerințe:

- să realizeze operația propriu-zisă de înmulțire a celor doi biți;
- să asigure operația de adunare presupusă de algoritmul de înmulțire al numerelor reprezentate pe mai mulți biți;
- să permită interconectarea într-o rețea bidimensională care asigură propagarea operanzilor și a rezultatelor parțiale astfel încât să se poată realiza combinațional operația de înmulțire.

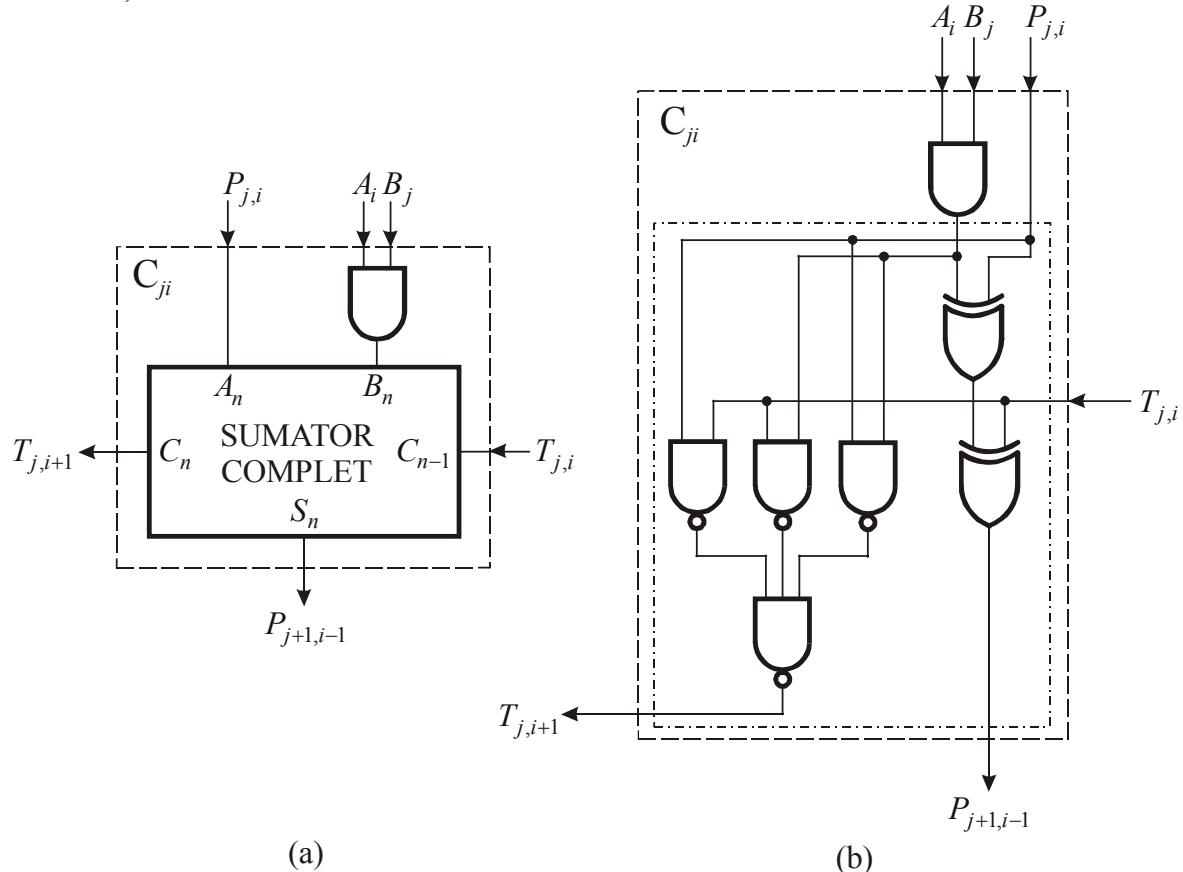


Fig.2.127 Schema-bloc (a) și schema logică (b) a unei celule C_{ji} .

O asemenea celulă, a cărei schemă logică este prezentată în figura 2.127¹⁷, însumează cifra A_i a deînmulțitului, dacă cifra B_j a înmulțitorului este 1, cu cifra de același rang a

¹⁷ De această dată, în desene și în text, se va folosi pentru transport notația T și nu C , ca până în acest moment, pentru a nu se confunda cu notația pentru celulă.

produsului parțial, $P_{j,i}$ (rezultat de pe rândul precedent) și cu transportul $T_{j,i}$ (provenit de la o însumare asemănătoare efectuată pe rangul precedent ($i-1$) al aceluiași rând).

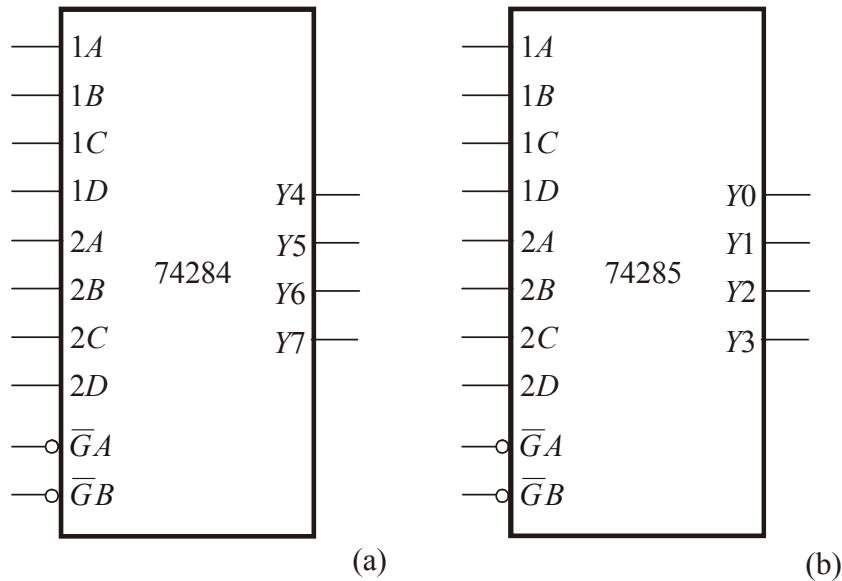


Fig.2.128 Schema-bloc a unui multiplicator binar paralel.

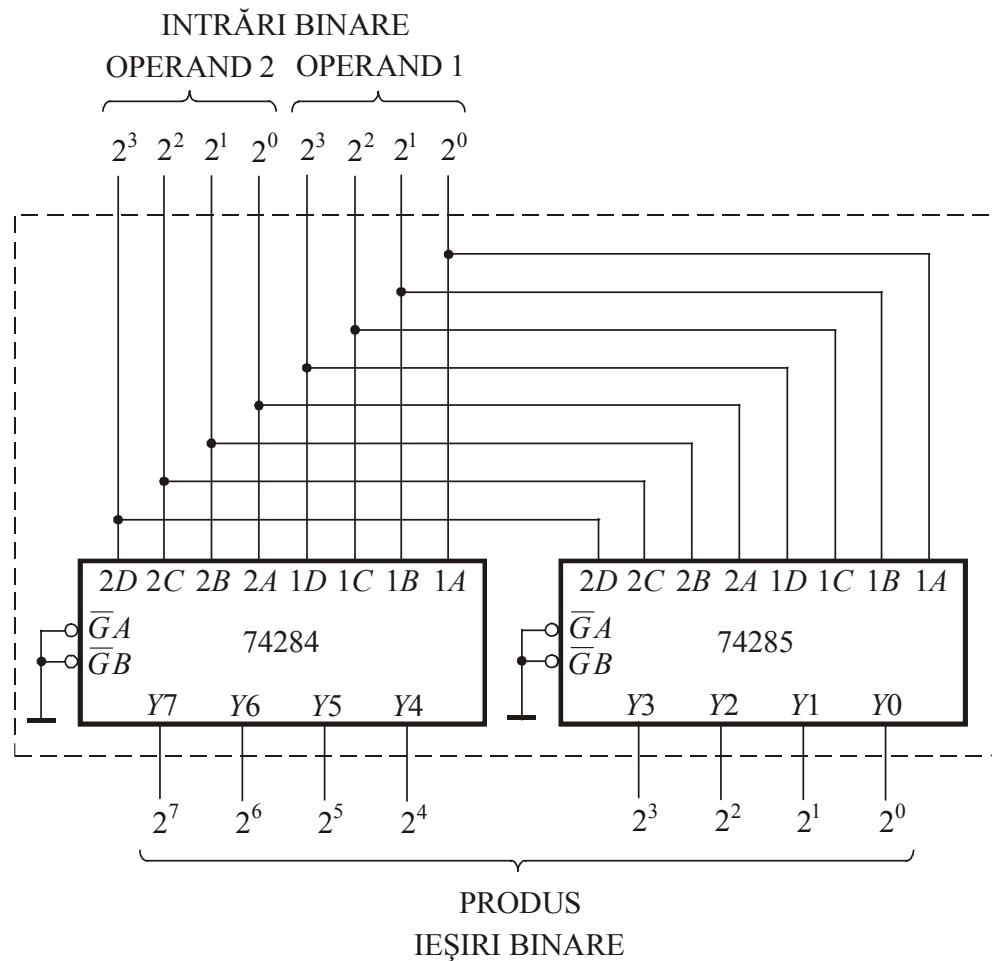


Fig.2.129 Circuit pentru înmulțirea a două numere binare cu câte patru cifre.

CIRCUITE LOGICE COMBINATORIALE

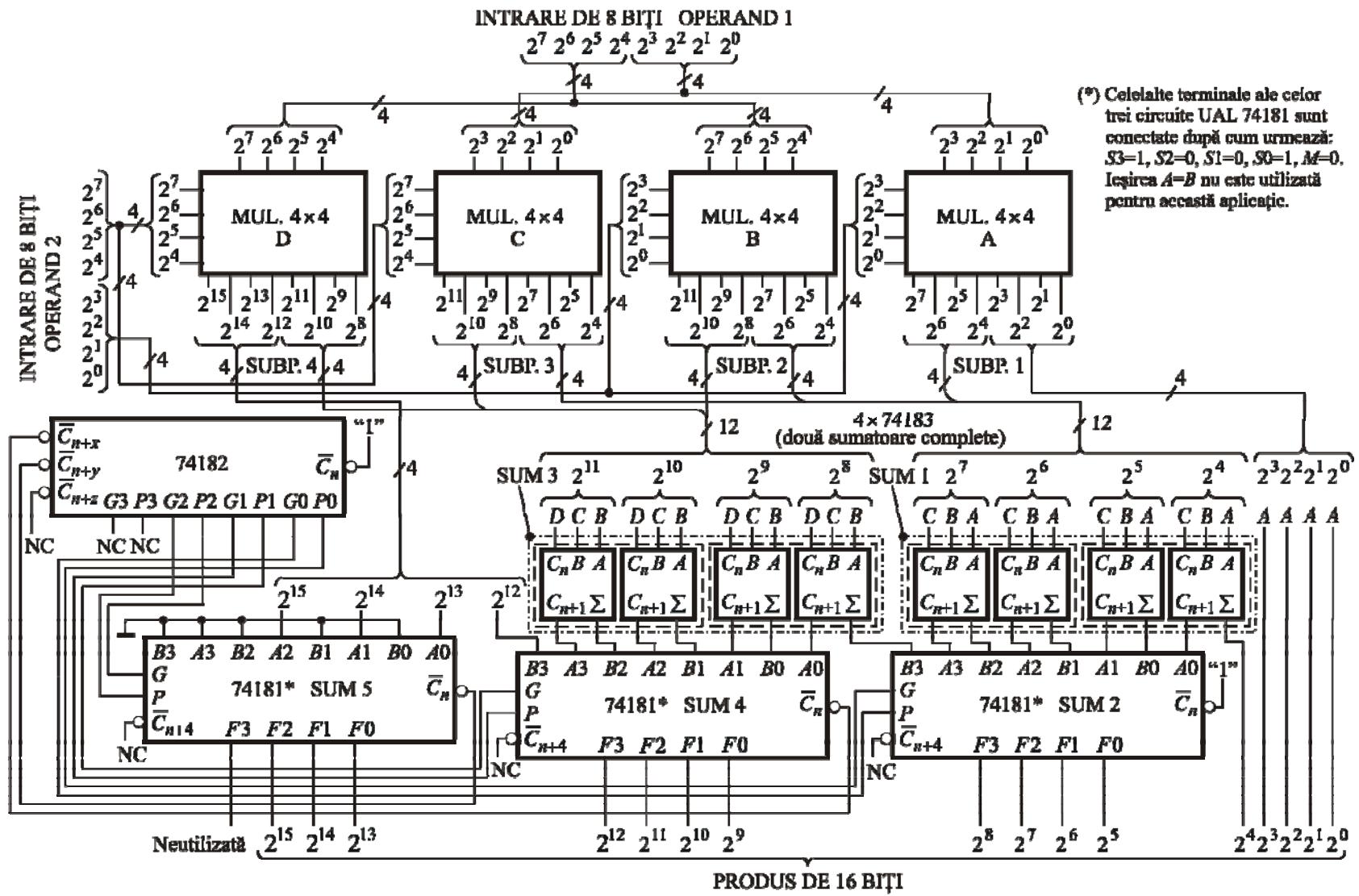


Fig.2.130 Circuit pentru înmulțirea a două numere cu căte 8 cifre binare.

Ca rezultat al acestei însumări, celula generează cifra de rangul i a produsului parțial următor, $P_{j+1,i-1}$ și transportul pentru rangul superior $T_{j,i+1}$. În concluzie, se poate aprecia că circuitele care realizează înmulțirea rapidă a numerelor binare se pot obține folosindu-se elemente integrate construite după principiul prezentat mai sus. Un exemplu de element integrat de acest tip îl reprezintă circuitele integrate 74284 și 74285 [34], cu schemele-bloc date în figura 2.128, care generează partea mai semnificativă, respectiv mai puțin semnificativă a produsului a două numere binare de câte patru cifre (v. fig. 2.129, [34]).

Aceste circuite pot fi folosite și pentru realizarea produsului a două numere cu mai mult de patru poziții binare fiecare, dacă se realizează subproduse cu părți de câte patru cifre ale acestora, care se însumează apoi decalate, în funcție de ponderea lor. Deci expandarea funcției de multiplicare cu astfel de circuite nu este directă fiind necesare circuite adiționale. Spre exemplu, un multiplicator de numere de opt biți cu 16 ieșiri necesită 16 capsule, dintre care numai opt sunt multiplicatoare. În figura 2.130 este prezentată o schemă pentru înmulțirea a două numere pozitive de câte opt cifre fiecare [34]. Cei doi operanzi sunt împărțiți în câte două părți, de patru cifre fiecare, care se înmulțesc între ele rezultând patru subproduse, SUBP1, SUBP2, SUBP3, SUBP4. Partea mai puțin semnificativă (p.m.p.s.) din SUBP1 reprezintă primele patru cifre m.p.s. din produs. Următoarele patru cifre mai semnificative se obțin însumând, în sumatorul SUM1, format din patru sumatoare complete pentru un rang, p.m.p.s. din SUBP2, partea mai semnificativă (p.m.s.) din SUBP1 și p.m.p.s. din SUBP3, iar apoi asimilând transporturile, în rangurile corespunzătoare ponderii acestora, cu ajutorul sumatorului pentru patru cifre binare, SUM2 (sumatoarele 2,4 și 5 sunt de fapt circuite UAL). P.m.s. a subprodusului SUBP2, p.m.s a produsului SUBP3 și p.m.p.s. a subprodusului SUBP4 sunt tratate asemănător în sumatoarele SUM3 (format, de asemenea, din patru sumatoare complete pentru un rang) și SUM4. Sumatorul SUM5 adună p.m.s. a subprodusului SUBP4 și transportul provenit din rangurile inferioare. Pentru ca timpul de efectuare a înmulțirii să fie afectat cât mai puțin de timpul de propagare a transportului, în grupul sumatoarelor SUM2, SUM4 și SUM5 se folosește o schemă de anticipare a transportului.

Această schemă este expandabilă pentru implementarea unor circuite multiplicatoare de $N \times M$ biți.

ANEXA A

REPREZENTAREA PE HIPERCUB A FUNCȚIILOR BOOLEENE

O funcție booleană de n variabile poate fi reprezentată pe un hipercub n -dimensional cu latura egală cu unitatea, în care fiecare variabilă îi corespunde una dintre cele n dimensiuni, dacă se adoptă un mod de marcare distinctă a vârfurilor n -hipercubului ale căror coordonate sunt identice cu n -uplele aplicate de funcție în zero și a vârfurilor n -hipercubului ale căror coordonate sunt identice cu n -uplele aplicate de funcție în 1 [20,23].

Pe cubul din figura A.1 s-a reprezentat funcția booleană:

$$f^{\text{FCD}}(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3 + x_1 x_2 x_3, \quad (\text{A.1})$$

marcându-se printr-un punct vârfurile cărora le corespunde valoarea 1 a funcției.

Urmărind figura A.1 se vede că funcția booleană poate fi reprezentată nu numai prin vârfurile cubului ci și prin muchiile sale însemnate cu linie îngroșată care unesc două vârfuri marcate, sau prin fețele sale delimitate numai de muchiile însemnate. De exemplu, cele două vârfuri de coordonate $x_1 = 1, x_2 = 1$ și $x_3 = 1$, respectiv $x_1 = 1, x_2 = 1$ și $x_3 = 0$ pot fi reprezentate și prin muchia paralelă cu axa x_3 , pentru care $x_1 = 1, x_2 = 1$.

Se consideră vârfurile hipercubului n -dimensional ca subcuburi 0-dimensionale, notate prin n -uple. Muchiile hipercubului n -dimensional se consideră subcuburi 1-dimensionale și sunt notate cu partea comună a celor două subcuburi 0-dimensionale din care este format. Pe poziția variabilei care diferă în cele două subcuburi componente se scrie simbolul “-”. Fețele hipercubului n -dimensional se consideră subcuburi r -dimensionale formate din 2^r vârfuri adiacente și sunt notate cu partea comună a celor 2^r subcuburi 0-

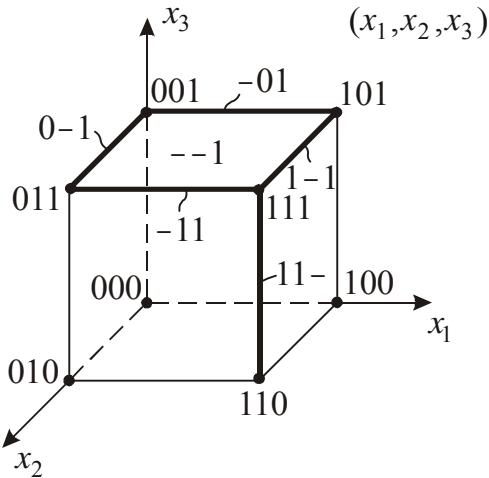


Fig.A.1 Reprezentarea pe hipercub a funcției booleene de trei variabile.

dimensionale incluse. Pe pozițiile variabilelor care diferă în cele 2^r subcuburi 0-dimensionale se scriu simboluri “–“.

Pentru funcția f din figura A.1 există mai multe mulțimi de subcuburi prin care se poate reprezenta. Una dintre acestea este mulțimea subcuburilor 0-dimensionale:

$$f(x_1, x_2, x_3) = \left\{ \begin{array}{l} 001 \\ 011 \\ 101 \\ 110 \\ 111 \end{array} \right\}, \quad (\text{A.2})$$

corespunzătoare FCD a funcției.

Alte două mulțimi de subcuburi care reprezintă funcția f sunt mulțimea de subcuburi 1-dimensionale:

$$f(x_1, x_2, x_3) = \left\{ \begin{array}{l} 0-1 \\ -01 \\ 1-1 \\ -11 \\ 11- \end{array} \right\}, \quad (\text{A.3})$$

și mulțimea de subcuburi de diferite dimensiuni:

$$f(x_1, x_2, x_3) = \left\{ \begin{array}{l} --1 \\ 11- \end{array} \right\}. \quad (\text{A.4})$$

Există mai multe mulțimi de subcuburi care pot reprezenta o funcție. Importantă pentru reprezentarea funcției este mulțimea subcuburilor 0-dimensionale și mulțimea cu număr minim de elemente.

Orice mulțime de subcuburi care cuprinde toate vârfurile n -hipercubului corespunzătoare n -uplelor unei funcții de n variabile aplicate în 1 este o acoperire a funcției. Pentru a se obține o acoperire cu număr cât mai mic de elemente trebuie luate subcuburi de dimensiuni cât mai mari.

Un subcub al funcției care nu este inclus într-un alt subcub de dimensiune mai mare este un implicant prim al funcției. Mulțimea tuturor implicanților primi ai unei funcții formează o acoperire a funcției deoarece include toate n -uplele aplicate de funcție în 1. Elementele mulțimii minime se aleg dintre implicanții primi ai funcției.

De exemplu, cei doi implicanți primi ai funcției f reprezentată pe cubul din figura A.1, subcuburile $--1$ și $11-$, formează o acoperire a funcției.

Un implicant prim care conține un subcub 0-dimensional care nu mai este cuprins în nici un alt implicant prim al funcției devine implicant prim esențial.

Urmărind figura A.1 se vede că atât subcubul $--1$ cât și subcubul $11-$ reprezintă implicanții primi esențiali ai funcției.

Orice acoperire formată numai din implicanții primi esențiali este o acoperire cu număr minim de elemente deoarece înălțarea oricărui din elementele sale ar face să nu mai fie o acoperire, s-ar elimina cel puțin un subcub 0-dimensional al funcției.

ANEXA B

STRUCTURI TTL/CMOS DE IEŞIRE

Proiectanții de circuite integrate au adus numeroase modificări circuitului TTL respectiv CMOS de bază, pentru a realiza porți logice dedicate anumitor aplicații. În continuare, se vor descrie doar două dintre cele mai cunoscute variante de structuri TTL/CMOS de ieșire [2,23,27,30].

Ieșiri cu colectorul/drena în gol

Porțile logice TTL la care etajul de ieșire este de tip *totem pole* (*în contratimp*) prezintă un neajuns important și anume acela că ieșirile acestor porți nu pot fi cuplate în paralel. Această caracteristică constituie unul dintre dezavantajele porții TTL, încrucișat multe dintre sistemele numerice actuale sunt organizate pe magistrale de informație, la care se leagă ieșirile unui număr mare de circuite.

În figura B.1 se prezintă schema electrică a porții fundamentale TTL (poarta řI-NU cu două intrări). Aceste porți nu sunt adaptate pentru o funcționare ca cea descrisă mai sus, datorită impedanțelor mici de ieșire ale circuitului. Se consideră două asemenea porți ale căror ieșiri sunt cuplate în paralel (v. fig. B.2). Dacă ambele ieșiri sunt simultan în starea Low (0 logic), sau simultan în starea High (1 logic), circuitul funcționează corect. În cazul însă în care o ieșire se află în „0” și alta în „1”, se realizează punerea practic la masă a sursei (scurtcircuitarea acesteia prin intermediul tranzistoarelor aflate în conducție, unul aparținând porții aflate în starea High, celălalt aparținând porții aflate în starea Low și prin linia comună de magistrală la care sunt legate ieșirile celor două porți); acest lucru are drept consecință parcurgerea tranzistorului T_4'' de un curent mare, limitat numai de diodă (D'') și de rezistența R_4'' (de valoare scăzută); un asemenea regim se menține până când nivelele logice se schimbă. Această situație atrage după sine creșterea puterii absorbite de la sursa de alimentare și, ceea ce este mai important, disiparea unei puteri excesiv de mari în tranzistoarele T_3' , T_4'' și în rezistența R_4'' . Ca urmare, parametrii porții se modifică (dacă tranzistoarele nu se distrug prin ambalare termică) pierzându-se imunitatea la zgromot (se alterează nivelele logice 0 și respectiv 1).

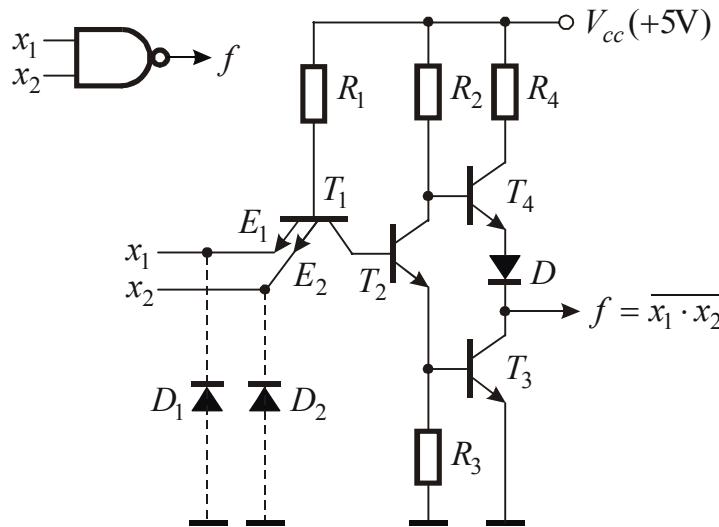


Fig.B.1 Poarta fundamentală TTL (ŞI-NU).

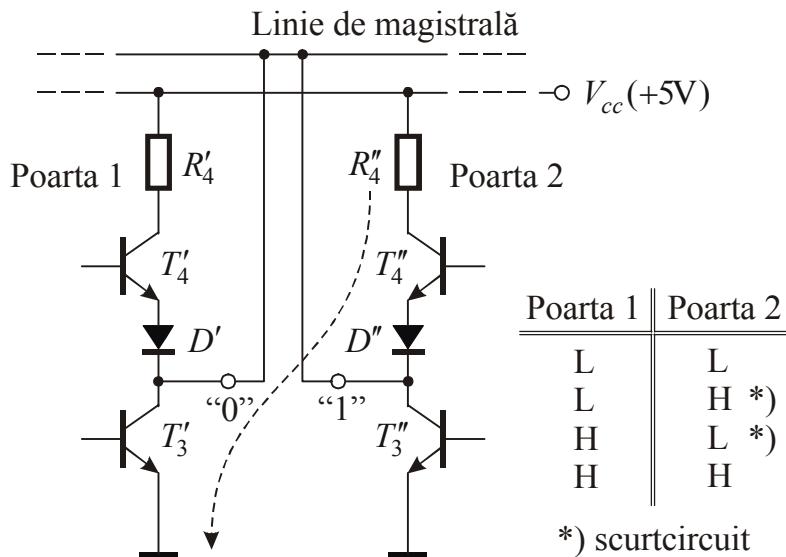


Fig.B.2 Etajele de ieşire a două porți ŞI-NU TTL cuplate în paralel.

Pentru a se cupla, totuși, în paralel mai multe porți se utilizează circuite de tip cu *colector în gol/deschis, OC (Open Collector)*. La aceste porți nu mai apar, în schema etajului de ieșire, tranzistorul T_4 , dioda D și rezistența R_4 (lipsește repotorul pe emitor din etajul în contratimp – v. fig. B.3). Colectorul tranzistorului T_3 este conectat la sursa de alimentare, singur sau împreună cu colectoarele altor porți TTL OC, prin intermediul unei rezistențe de sarcină R_c (v. fig. B.4) (circuitele TTL OC, generează corect potențialul coborât la ieșire (0 logic), dar necesită o rezistență conectată la sursa de alimentare (*pull-up resistor* – rezistor de forțare în starea High) pentru a genera nivelul logic 1). La aceeași magistrală se leagă și intrările porților comandate (de exemplu, două sau mai multe porți ŞI-NU TTL). Aceste intrări sunt legate la emitoarele tranzistorului multi-emitor T_1 , tranzistor ce realizează funcția ŞI-NU în schema fiecărei porți ŞI-NU (v. fig. B.1).

Potențialul pe o linie a magistralei va fi High (1 logic) numai atunci când toate ieșirile porților conectate (cablate) la linia respectivă de magistrală sunt în starea High (1 logic). În schimb, potențialul pe linia magistralei va fi Low (0 logic) când cel puțin una

dintre ieșirile portilor conectate (cablate) la linia de magistrală este în starea Low (0 logic).

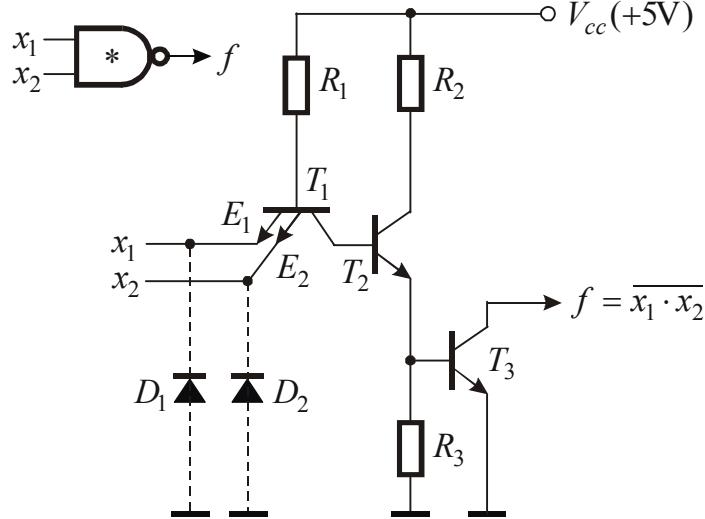


Fig.B.3 Poartă řI-NU TTL cu colectorul în gol.

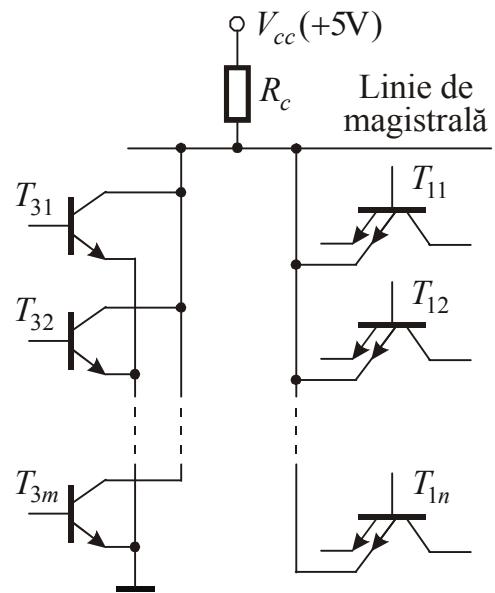


Fig.B.4 Cablarea portilor cu colectorul în gol.

Această conexiune (cablată) realizează funcția řI, de unde și denumirea de řI-cablat (wired – AND). În cazul în care semnalele se consideră active pe 0 logic (deci în logică negativă), conexiunea realizează funcția SAU-cablat (wired – OR). Deoarece se realizează funcțiile logice řI respectiv SAU fără a exista fizic o poartă řI respectiv SAU, în literatură se mai folosește și denumirea de řI-virtual respectiv SAU-virtual.

Calculul valorii rezistenței de sarcină R_c se face acoperitor, ținând cont de variația tensiunii de alimentare (V_{cc}), de asigurarea marginilor de zgomot (M_H și M_L) și de asigurarea nivelelor de tensiune permise la intrarea portilor, atât pentru starea High (V_{IH}) cât și pentru starea Low (V_{IL}). În acest mod rezultă domeniul pentru alegerea valorii rezistenței R_c , care trebuie să fie cuprinsă în intervalul:

$$R_{c\min} \leq R_c \leq R_{c\max}. \quad (\text{B.1})$$

Aceste valori limite se obțin ținând cont de curentii debitați de portile conectate în paralel și de curentii absorbiți de portile comandate și sunt:

$$R_{c\max} \leq \frac{V_{cc\min} - V_{OH\min}}{m \cdot I_{OH\max} + n \cdot I_{IH\max}} \quad (\text{B.2})$$

și

$$R_{c\min} \geq \frac{V_{cc\max} - V_{OL\max}}{I_{OL\max} - n \cdot I_{IL\max}}, \quad (\text{B.3})$$

unde:

V_{cc} tensiunea sursei de alimentare (tensiunea de colector);

$V_{OH\min}$ valoarea minimă garantată de catalog a tensiunii de ieșire, în starea H;

$V_{OL\max}$ valoarea maximă garantată de catalog a tensiunii de ieșire, în starea L;

$I_{OH\max}$	currentul maxim rezidual prin tranzistorul blocat al etajului de ieșire al unei porți aflate în starea H;
$I_{OL\max}$	currentul maxim admis absorbit de etajul de ieșire al unei porți, în starea L;
$I_{IH\max}$	currentul maxim absorbit de intrarea unei porți comandate, în starea H;
$I_{IL\max}$	currentul maxim generat de intrarea unei porți comandate, în starea L;
n	numărul intrărilor de porți comandate legate la magistrală;
m	numărul ieșirilor de porți de comandă cablate legate la magistrală.

Se constată că valoarea minimă a rezistenței de sarcină nu depinde de numărul porților cuplate în paralel.

Pentru simplificarea proiectării schemelor logice care folosesc funcția řI-cablat, în cataloage se dă tabele de dimensionare a rezistenței R_c , în funcție de numărul porților cu colector în gol și numărul porților TTL comandate.

Tabelul de mai jos prezintă valorile acestor rezistențe de sarcină, calculate în următoarele condiții: $V_{cc} = 5V$, $I_{OH\max} = 250 \mu A$, $I_{OL\max} = 16mA$, $I_{IL\max} = 1,6mA$, $I_{IH\max} = 40\mu A$, $V_{OL\max} = 0,4V$, $V_{OH\min} = 2,4V$.

Tab.B.1 Tabel de dimensionare a rezistenței de sarcină R_c .

Numărul de porți comandate (n)	Numărul porților cu colectorul în gol (m)						
	Valoarea rezistenței de sarcină $R_c [\Omega]$						
	$R_{c\max} [\Omega]$						$R_{c\min} [\Omega]$
	1	2	3	4	5	6	7
1	8965	4814	3291	2500	2015	1688	1452
2	7878	4482	3132	2407	1954	1654	1420
3	7027	4193	2988	2321	1897	1604	1390
4	6341	3939	2857	2241	1843	1566	1361
5	5777	3714	2736	2166	1793	1529	1333
6	5306	3513	2626	2096	1744	1494	1306
7	4905	3333	2524	2031	1699	1460	1280
8	4561	3170	2429	1969	1656	x	x
9	4262	3023	x	x	x	x	x
10	4000*	x	x	x	x	x	4000*

x – caz nerecomandat sau imposibil;

* – valoarea teoretică este ∞ .

Implementarea funcțiilor logice cu porți OC conduce, în general, la scheme logice mai avantajoase din punctul de vedere al numărului de nivele logice și al timpului de propagare.

În figura B.5 se prezintă trei variante de implementare ale funcției:

$$f^{\text{FMC}}(x_1, \dots, x_7) = (x_1 + \bar{x}_2)(x_3 + \bar{x}_4 + x_5)(x_6 + x_7) = \\ = \overline{\bar{x}_1 x_2 + \bar{x}_3 x_4 \bar{x}_5 + \bar{x}_6 \bar{x}_7} = \overline{\bar{x}_1 x_2} \cdot \overline{\bar{x}_3 x_4 \bar{x}_5} \cdot \overline{\bar{x}_6 \bar{x}_7}. \quad (\text{B.4})$$

Varianta de implementare din figura B.5,a necesită o logică pe trei nivele (se obține o întârziere mare). Aceeași funcție se implementează în figura B.5,b cu porți logice řI-NU OC. Circuitul realizează funcția řI între ieșirile porților řI-NU. Se constată că circuitul OC realizează funcția řI-SAU-NU pentru grupurile de variabile de la intrările por-

ților ȘI-NU. Aceasta explică denumirea impropriu de SAU-cablat care nu este corectă deoarece adevarata funcție realizată este ȘI-cablat. Sinteză schemei logice se poate face și plecând de la forma minimă disjunctivă a negaiei funcției.

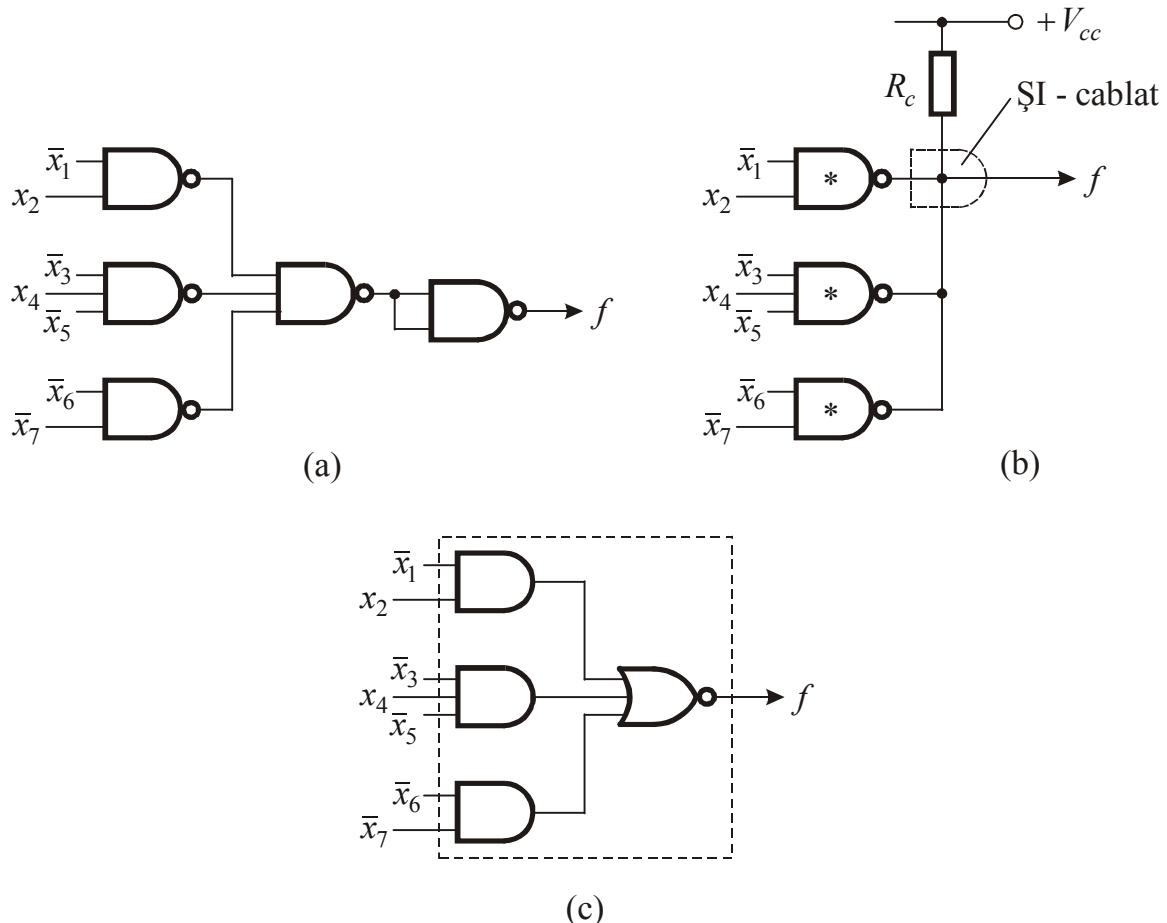


Fig.B.5 Variante de implementare a funcției din exemplu: (a) varianta de implementare cu porți logice ȘI-NU „clasice”; (b) realizarea funcției cu porți logice ȘI-NU cu colectorul în gol; (c) schema echivalentă cu o poartă logică ȘI-SAU-NU.

Circuitul cu colectorul în gol are următoarele dezavantaje:

- impedanță de ieșire mare în starea High (rezistență montată în colector față de cea de ieșire a unui repetor pe emitor la structura în contratimp);
- fronturi și tempi de propagare mari, mai ales la comutarea $0 \rightarrow 1$, motiv pentru care funcționează la o viteză mai mică decât viteza maximă de lucru a circuitelor TTL cu ieșire în contratimp;
- imunitate scăzută la zgomot și necesitatea unei rezistențe suplimentare calculabile de fiecare dată, în funcție de condițiile de lucru.

O configurație similară, numită *ieșire cu drena în gol*, se regăsește la familiile de circuite logice CMOS. Unele porți CMOS se fabrică și în varianta cu *drena în gol*, *OD* (Open Drain). Aplicațiile porților CMOS cu drena în gol și calculele necesare sunt similare celor prezentate pentru porțile TTL cu colectorul în gol.

Ieșiri cu trei stări

Ieșirile logice au două stări normale, Low (L) și High (H), corespunzătoare valorilor logice 0 și 1. Unele ieșiri prezintă însă și o a treia stare, care nu este de fapt o stare logică

numită *stare de înaltă impedanță*, HZ (High Z) sau *flotantă*. Într-o asemenea stare, ieșirea se comportă de parcă nici n-ar fi conectată cu restul circuitului, cu excepția prezenței unui curent rezidual slab, care poate circula către sau dinspre borna de ieșire. Prin urmare, o ieșire se poate afla într-una dintre cele trei stări: 0 logic, 1 logic și HZ, numindu-se în acest caz *ieșire cu trei stări logice* (!), TSL (Tri/Three State Logic).

Dispozitivul de bază cu trei stări este *circuitul tampon cu trei stări*, numit și *circuit de comandă cu trei stări* (driver). În figura B.6 sunt prezentate simbolurile logice pentru patru variante constructive ale circuitului tampon cu trei stări. Simbolul de bază este cel de circuit tampon neinversor (v. fig. B.6,a,b) sau inversor (v. fig. B.6,c,d). Semnalul suplimentar, din partea superioară a simbolului reprezintă o *intrare de activare a celor trei stări*, care poate fi activă în High (v. fig. B.6,a,c) sau în Low (v. fig. B.6,b,d). Când intrarea de activare este „confirmată”, dispozitivul se comportă ca un circuit tampon sau ca un inversor obișnuit. Când intrarea de activare este negată, ieșirea dispozitivului este „flotantă”; cu alte cuvinte ieșirea trece în starea de înaltă impedanță, echivalentă cu deconectarea, iar funcțional pare că nici nu există.

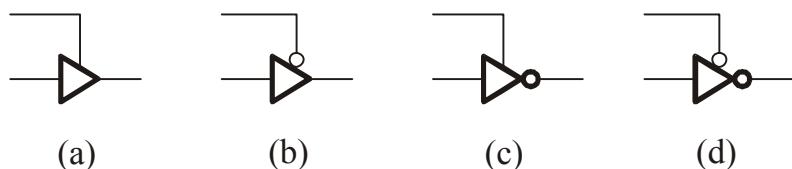


Fig.B.6 Diverse tipuri de circuite tampon cu trei stări: (a) neinversor, cu intrarea de activare cu nivel activ High; (b) neinversor, cu intrarea de activare cu nivel activ Low; (c) inversor, cu intrarea de activare cu nivel activ High; (d) inversor, cu intrarea de activare cu nivel activ Low.

În figura B.7,a se prezintă schema electrică a unui inversor cu trei stări TTL, împreună cu simbolul logic (v. fig. B.7,b) și tabelul de adevăr (v. fig. B.7,c). Se reamintește că în circuitul de ieșire în contratimp al unei porți TTL (v. fig. B.1), în permanență unul dintre tranzistoare (T_3 sau T_4) conduce.

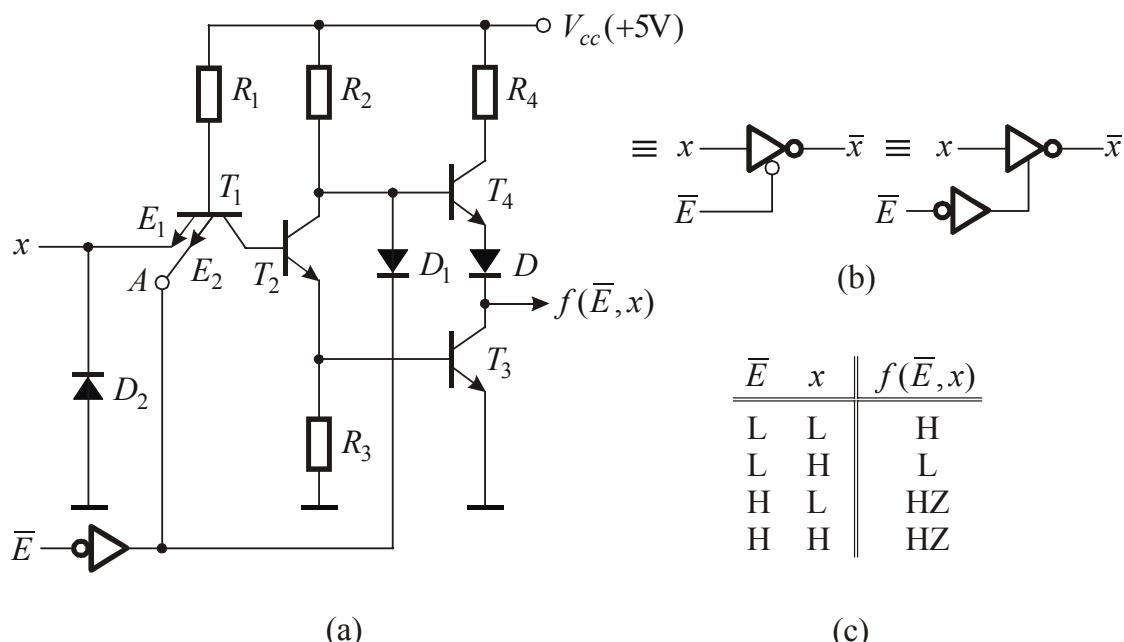


Fig.B.7 Inversor cu trei stări TTL: (a) schema electrică; (b) simbolul logic; (c) tabelul de adevăr.

Dacă la un moment dat s-ar putea realiza blocarea ambelor tranzistoare, atunci cir-

cuitul de ieșire ar fi izolat și, văzută dinspre ieșire, poarta TTL s-ar prezenta ca o impedanță de valoare ridicată care nu consumă decât un inevitabil curent rezidual de valoare foarte mică. Circuitul ar dispune în acest caz de trei stări (dar va funcționa în permanență în logică binară): starea 0, starea 1 și starea izolată, de înaltă impedanță, care lasă ieșirea flotantă când cele două tranzistoare sunt blocate.

Așa cum s-a precizat mai sus, dispozitivele prevăzute cu ieșiri TSL au o intrare suplimentară numită, de obicei, *activarea (validarea) ieșirii*, $\overline{OE}/\overline{E}$ (*Output Enable/Enable*) sau *dezactivarea (invalidarea) ieșirii*, OD (*Output Disable*), pentru trecerea ieșirii (ieșirilor) dispozitivului în starea de înaltă impedanță. Circuitul din figura B.7,a este prevăzut cu o bornă de autorizare (validare) a funcționării (\overline{E}). Pentru $\overline{E} = 0$ (intrare prioritată), $A = 1$, dioda D_1 este blocată și circuitul se comportă ca un inversor pentru intrarea x . Dacă $\overline{E} = 1$, $A = 0$, D_1 se deschide, T_1 este saturat, T_2 și T_3 sunt blocate, iar T_4 este blocat și el deoarece prin dioda D_1 deschisă, potențialul bazei sale coboară la o valoare scăzută. Între ieșire și masă se află o rezistență mare și circuitul se prezintă, la ieșire, ca o impedanță înaltă (HZ). Dacă structurii din figura B.7,a î se adaugă una sau mai multe intrări, se obține poarta fundamentală **SI-NU** a acestei subfamilii, cu ajutorul căreia se poate realiza o întreagă familie de circuite logice.

Asemenea porților TTL, porțile CMOS pot fi prevăzute cu ieșiri cu trei stări. Aplicațiile porților CMOS TSL sunt similare aplicațiilor porților TTL TSL.

Avantajele utilizării porților TSL în raport cu utilizarea celor OC/OD sunt următoarele:

- oferă o impedanță mică la ieșire în cele două stări (L și H), ca și la poarta standard;
- nu necesită rezistență adițională;
- în starea de înaltă impedanță (HZ) nu încarcă circuitele cu care sunt cuplate la ieșire;
- permite cuplarea în paralel a ieșirilor, fără dezavantajele întâlnite la montajul cu colectorul în gol.

Subfamilia de circuite integrate cu trei stări (TSL), reprezentă o alternativă bună la cuplarea mai multor circuite în paralel pentru transmiterea, de exemplu, a informației pe o magistrală comună de date. O magistrală cu trei stări se realizează prin cablarea împreună a mai multor ieșiri TSL. Circuitele de comandă pentru activările ieșirilor trebuie să lucreze astfel încât la un moment dat să fie activă cel mult o ieșire (nu în starea sa HZ!). Singurul dispozitiv activat poate transmite pe magistrală nivelele logice (H și L).

Ca și în cazul porților logice, într-un singur circuit integrat SSI pot fi încapsulate câteva circuite tampon cu trei stări, independente. Majoritatea aplicațiilor ce utilizează linii comune se realizează cu o magistrală cu mai mult de un bit de date. De aceea, pentru a reduce dimensiunile capsulei în cazul aplicațiilor cu magistrale foarte largi, cele mai utilizate componente MSI conțin mai multe circuite tampon cu trei stări cu intrări de activare comune.

Un *transceiver*¹ de magistrală conține perechi de circuite tampon cu trei stări conectate în sensuri opuse la fiecare pereche de pini, astfel ca datele să poată fi transferate în orice direcție.

¹ Cuvânt format prin contragerea cuvintelor *transmitter (emitter)* și *receiver (receptor)*.

ANEXA C

SISTEME DE NUMERAȚIE. CODURI

Informația este o noțiune de mare generalitate, ca și noțiunea de mulțime, despre care se poate afirma că reprezintă, pentru un sistem oarecare (biologic, tehnic, social, etc.) un mesaj despre evenimentele care au avut loc, au loc, sau vor avea loc în interiorul sau exteriorul sistemului.

Baza celor mai multe schimburi de informație o constituie reprezentările simbolice pentru idei, cantități, modele, etc. Reprezentarea simbolică se realizează cu ajutorul unei mulțimi de simboluri și reguli care guvernează utilizarea acestor simboluri.

Reprezentarea simbolică a informației într-un sistem numeric poate fi analizată prin prisma sistemelor de numerație și a codurilor utilizate.

Sistemul de numerație este format din totalitatea regulilor de reprezentare a numerelor cu ajutorul unor simboluri numite *cifre*. Sistemele de numerație sunt de două feluri: *poziționale* și *nepozitionale*. Un exemplu de sistem pozițional este sistemul zecimal, iar de sistem nepozitional sistemul roman. În sistemele de calcul se folosesc în special sistemele de numerație poziționale, datorită simplității de reprezentare și de efectuare a calculelor aritmetice.

Sisteme de numerație poziționale

Un sistem de numerație pozițional este caracterizat prin *bază*; aceasta reprezintă numărul total de simboluri (cifre) ale sistemului. În tabelul C.1 sunt prezentate câteva baze, împreună cu mulțimile de simboluri corespunzătoare.

Într-un sistem de numerație de bază b , un număr N , format din parte întreagă și parte fracționară, se poate reprezenta într-una din următoarele trei forme:

$$N = \begin{cases} a_{n-1}a_{n-2}\dots a_1a_0a_{-1}a_{-2}\dots a_{-m}, \\ a_{n-1}b^{n-1} + \dots + a_1b^1 + a_0b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \dots + a_{-m}b^{-m}, \\ \sum_{i=-m}^{n-1} a_i b^i, \end{cases} \quad (\text{C.1})$$

unde, b este baza sistemului, a_i sunt cifrele, n numărul de cifre ale părții întregi, m numărul de cifre ale părții fracționare, a_{n-1} cifra cea mai semnificativă (c.m.s.), a_{-m} cifra cea mai puțin semnificativă (c.m.p.s.).

Tab.C.1 Sisteme de numerație.

Baza	Denumirea sistemului	Simboluri utilizate
2	Binar	0,1
3	Ternar	0,1,2
4	Cuaternar	0,1,2,3
8	Octal	0,1,2,3,4,5,6,7
10	Zecimal	0,1,2,3,4,5,6,7,8,9
16	Hexazecimal	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Deoarece numărul de cifre utilizate în sistem este egal cu baza, rezultă că:

$$0 \leq a_i \leq b-1, \text{ unde } -m \leq i \leq n-1. \quad (\text{C.2})$$

Dacă $n=0$, atunci numărul N este subunitar, iar dacă $m=0$, atunci numărul N este întreg.

Relația (C.1) arată de ce astfel de sisteme sunt denumite pozitionale: fiecare cifră a_i (din rangul i) intră în valoarea numărului respectiv cu o pondere dată de puterea i a bazei b . Fiecare număr se obține din numărul anterior prin adăugarea unei unități la ultima cifră.

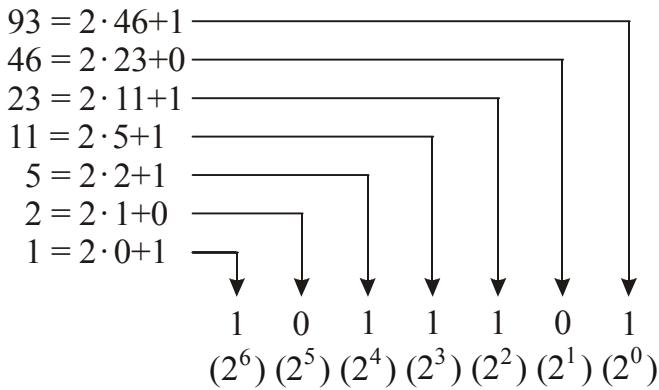
În general, conversia între două baze de numerație nu se poate efectua doar printr-o simplă substituire; sunt necesare unele operații aritmetice, existând în acest sens algoritmi de conversie. Algoritmul de conversie zecimal-binar, $N_{(10)} \rightarrow N_{(2)}$, pentru un număr întreg, se deduce pornind de la expresia (C.1), scrisă sub forma:

$$N = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \dots + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0, \quad (\text{C.3})$$

prin împărțiri succesive:

$$\begin{aligned} N &= 2 \underbrace{(a_{n-1} \cdot 2^{n-2} + a_{n-2} \cdot 2^{n-3} + \dots + a_3 \cdot 2^2 + a_2 \cdot 2^1 + a_1)}_{N_1} + a_0 \text{ (rest)}, \\ N_1 &= 2 \underbrace{(a_{n-1} \cdot 2^{n-3} + a_{n-2} \cdot 2^{n-4} + \dots + a_3 \cdot 2^1 + a_2)}_{N_2} + a_1 \text{ (rest)}, \\ &\dots \\ N_k &= 2(a_{n-1} \cdot 2^{n-k-2} + \dots + a_{n-k-1}) + a_k \text{ (rest)}, \end{aligned} \quad (\text{C.4})$$

se obțin astfel resturile care sunt tocmai bițiile numărului N exprimat în binar; de exemplu $93_{(10)} = ?_{(2)}$:



$$93_{(10)} = 1011101_{(2)},$$

$$N = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 93_{(10)}.$$

Algoritmul de conversie zecimal-binar, $N_{(10)} \rightarrow N_{(2)}$, pentru un număr fracționar (subunitar) se deduce pornind de la expresia:

$$N = a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + a_{-3} \cdot 2^{-3} + \dots + a_{-(m-1)} \cdot 2^{-(m-1)} + a_{-m} \cdot 2^{-m}, \quad (\text{C.5})$$

prin înmulțiri succesive cu 2:

$$\begin{aligned} N \cdot 2 &= a_{-1} + \underbrace{a_{-2} \cdot 2^{-1} + a_{-3} \cdot 2^{-2} + \dots + a_{-(m-1)} \cdot 2^{-(m-2)}}_{N_1} + a_{-m} \cdot 2^{-(m-1)}, \\ N_1 \cdot 2 &= a_{-2} + \underbrace{a_{-3} \cdot 2^{-1} + \dots + a_{-(m-1)} \cdot 2^{-(m-3)}}_{N_2} + a_{-m} \cdot 2^{-(m-2)}, \\ &\dots \\ N_{k-1} \cdot 2 &= a_{-k} + \underbrace{a_{-k-1} \cdot 2^{-1} + \dots + a_{-m} \cdot 2^{-(m-k)}}_{N_k}, \end{aligned} \quad (\text{C.6})$$

se obțin părți întregi (zero sau unu) care sunt tocmai biții numărului N , exprimat în binar (operația de înmulțire se oprește atunci când partea fracționară devine zero sau când se consideră satisfăcător un număr de cifre binare pentru precizia stabilită); de exemplu $0.48932_{(10)} = ?_{(2)}$:

$$\begin{aligned} 0.48932 \times 2 &= 0 + 0.97864 \rightarrow a_{-1} = 0, \\ 0.97864 \times 2 &= 1 + 0.95728 \rightarrow a_{-2} = 1, \\ 0.95728 \times 2 &= 1 + 0.91456 \rightarrow a_{-3} = 1, \\ 0.91456 \times 2 &= 1 + 0.82912 \rightarrow a_{-4} = 1, \\ 0.82912 \times 2 &= 1 + 0.75824 \rightarrow a_{-5} = 1, \\ 0.75824 \times 2 &= 1 + 0.51648 \rightarrow a_{-6} = 1, \\ 0.51648 \times 2 &= 1 + 0.03296 \rightarrow a_{-7} = 1, \end{aligned}$$

$$0.48932_{(10)} = 0.011111\dots_{(2)},$$

$$N = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} + 1 \cdot 2^{-6} + 1 \cdot 2^{-7} + \dots = 0.48932_{(10)}.$$

Pentru conversia zecimal-binară a unui număr ce prezintă atât parte întreagă, cât și parte fracționară, se combină cele doi algoritmi. Din exemplele de mai sus s-a putut vedea că pentru conversia binar-zecimală, $N_{(2)} \rightarrow N_{(10)}$, se adună puterile succesive ale lui 2, fiecare fiind înmulțită cu bitul din poziția respectivă a numărului binar. Algoritmii deduși mai sus ((C.4) și (C.6)) pot fi aplicati și în cazul conversiei zecimal-octal, zecimal-hexazecimal, etc.

Conversiile directe octal-binar, hexazecimal-binar, cât și conversiile inverse binar-octal, binar-hexazecimal, se realizează mult mai simplu datorită faptului că 8 și 16 sunt puteri ale lui 2. Un număr (mai mic decât baza) în octal și în hexazecimal poate fi exprimat prin trei cifre binare (triadă), respectiv prin patru cifre binare (tetradă). Deci, conversiile în ambele sensuri se bazează pe corespondența dintre triade, în numărarea binară și numerele până la șapte la numărarea în bază 8, respectiv dintre tetrade în numărarea binară și numerele până la 15 în numărarea în baza 16.

Exemple.

$$\frac{\underline{001}}{1} \cdot \frac{\underline{101}}{5} \cdot \frac{\underline{111}}{7} \cdot \frac{\underline{010}}{2}_{(2)} = 15.72_{(8)},$$

$$\frac{\underline{2}}{010} \cdot \frac{\underline{0}}{000} \cdot \frac{\underline{4}}{100}_{(8)} = 10.0001_{(2)},$$

$$\frac{\underline{1001}}{9} \cdot \frac{\underline{0000}}{0} \cdot \frac{\underline{1111}}{F} \cdot \frac{\underline{0010}}{2} \cdot \frac{\underline{1010}}{A}_{(2)} = 90F.2A_{(16)},$$

$$1D.C_{(16)} = \frac{\underline{0001}}{1} \cdot \frac{\underline{1101}}{D} \cdot \frac{\underline{1100}}{C}_{(2)}.$$

Conversiile inverse binar-hexazecimal, binar-octal se realizează delimitând cuvântul binar de la dreapta spre stânga (sau, pornind de la punctul părții fracționare în ambele sensuri) în tetrade sau triade și înlocuindu-le cu cifrele corespunzătoare din sistemul hexazecimal, respectiv din sistemul octal.

Conversiile directe hexazecimal-binar, octal-binar se realizează simplu prin substituția cifrelor hexazecimale sau octale prin tetrade sau triade corespunzătoare.

Pentru modul de realizare a operațiilor aritmetice în sistemele de numerație cu baza o putere a lui 2 (binar, octal, hexazecimal) se poate consulta [11].

Sistemul de numerație utilizat în sistemele numerice de calcul este *sistemul binar*. Reprezentarea numerelor în acest sistem prezintă mai multe forme în funcție de soluția aleasă pentru a se indica poziția virgulei și semnul numărului. Poziția fixă sau variabilă a virgulei determină reprezentarea numită în *virgulă fixă* sau, respectiv în *virgulă mobilă*.

Reprezentarea numerelor în virgulă fixă

Blocurile aritmetice ale sistemelor de calcul care lucrează în *virgulă fixă*, consideră vir-

gula plasată în fața celei mai semnificative cifre a numărului. Deci, numerele cu care se operează sunt *subunitare*. Pentru a se indica semnul numărului există mai multe tehnici, fiecare dintre acestea determinând un mod de reprezentare: prin *mărime și semn (cod direct)*, prin *complement față de 2 (cod complementar)* și prin *complement față de 1 (cod invers)*.

Înainte de a ne referi la fiecare dintre aceste reprezentări utilizate, să definim noțiunea de complement al unui număr N scris într-o bază b .

Definiție.

a) Se numește complement față de baza b al numărului $N_{(b)}$, numărul $[N]_{comp}$ definit prin relația:

$$[N]_{comp} = b^n - N_{(b)}. \quad (\text{C.7})$$

b) Se numește complement față de $b-1$ al numărului $N_{(b)}$ numărul $[N]_{inv}$ definit de relația:

$$[N]_{inv} = b^n - N_{(b)} - b^{-m}, \quad (\text{C.8})$$

unde, n reprezintă numărul de cifre ale părții întregi a numărului N , iar m reprezintă numărul de cifre ale părții fracționare a numărului N .

Exemplu. În sistemul de numerație binar:

$$[N]_{comp} = 2^n - N_{(2)} \quad \text{și} \quad [N]_{inv} = 2^n - N_{(2)} - 2^{-m}. \quad (\text{C.9})$$

Dacă $N_{(2)} = 1011.11$, atunci:

$$[N]_{comp} = 2^4 - 1011.11 = 0100.01,$$

$$[N]_{inv} = 2^4 - 1011.11 - 2^{-2} = 0100.00.$$

Cele trei moduri de reprezentare ale numerelor, menționate anterior, sunt identice dacă numărul N este pozitiv, iar dacă numărul este negativ, în reprezentarea prin mărime și semn cifrele numărului sunt chiar cifrele numărului considerat, în timp ce în celelalte două moduri de reprezentare, cifrele numărului reprezintă complementul față de 2 respectiv față de 1.

Reprezentarea prin mărime și semn (cod direct)

Reprezentarea prin mărime și semn a unui număr algebric N este dată de relația:

$$[N]_{dir} = a_n 2^n + \sum_{i=-m}^{n-1} a_i 2^i, \quad (\text{C.10})$$

unde, a_n reprezintă bitul de semn și ia valoarea 0 dacă $N \geq 0$ și valoarea 1 dacă $N \leq 0$, iar a_i ($-m \leq i \leq n-1$) sunt biții numărului N .

Exemple.

$$[+9]_{dir} = 0 \begin{array}{l} 1001 \\ \swarrow \quad \searrow \\ \text{valoarea} \\ \text{numărului} \\ \rightarrow \text{bitul de semn} \end{array}$$

$$[-9]_{dir} = 1 \begin{array}{l} 1001 \\ \swarrow \quad \searrow \\ \text{valoarea} \\ \text{numărului} \\ \rightarrow \text{bitul de semn} \end{array}$$

$$[-0.6875]_{dir} = 1 \begin{array}{l} 1011 \\ \swarrow \quad \searrow \\ \text{valoarea} \\ \text{numărului} \\ \rightarrow \text{bitul de semn} \end{array}$$

Acest sistem de reprezentare are avantajul de a fi foarte asemănător cu scrierea „normală” (semnul se indică printr-un bit special), dar din punctul de vedere al realizării calculelor prezintă unele dezavantaje. Realizarea unei operații de adunare sau scădere a două numere nu depinde numai de funcția de executat ci și de semnul numerelor respective; în aceste condiții este necesară examinarea, înainte de executarea operației respective a bitului de semn. Pe de altă parte, blocul aritmetic al unui sistem numeric se poate simplifica mult dacă în loc să conțină dispozitive de adunare și scădere, conține numai dispozitive de adunare. Acest lucru poate fi realizat prin alegerea convenabilă a sistemului de reprezentare a numerelor și anume prin reprezentarea în *cod complementar*.

Reprezentarea prin complementul față de 2 (cod complementar)

Un număr algebric N se reprezintă prin complementul față de 2, în forma indicată de relația:

$$[N]_{comp} = \begin{cases} 0 \times 2^n + \sum_{i=-m}^{n-1} a_i 2^i, & \text{pentru } N \geq 0, \\ 1 \times 2^n + \sum_{i=-m}^{n-1} \bar{a}_i 2^i + 2^{-m}, & \text{pentru } N \leq 0, \end{cases} \quad (\text{C.11})$$

unde, $\bar{a}_i = 1 - a_i$, reprezintă complementul față de 1 al cifrei a_i ($-m \leq i \leq n-1$).

Exemple.

$$[+9]_{comp} = 01001;$$

$$[-9]_{comp} = ?$$

Deoarece $-9 = -1001_{(2)}$,

– se trece numărul în cod complementar:

$$1001 \rightarrow 0111,$$

– reprezentarea numărului -9 în cod complementar va fi:

$$[-9]_{comp} = 10111.$$

Se poate demonstra că reprezentarea numărului prin complement față de 2 corespunde chiar valorii reale a numărului [11].

Reprezentarea prin complementul față de 1 (cod invers)

Un număr algebric N se reprezintă prin complementul față de 1, în forma indicată de relația:

$$[N]_{inv} = \begin{cases} 0 \times 2^n + \sum_{i=-m}^{n-1} a_i 2^i, & \text{pentru } N \geq 0, \\ 1 \times 2^n + \sum_{i=-m}^{n-1} \bar{a}_i 2^i, & \text{pentru } N \leq 0, \end{cases} \quad (\text{C.12})$$

unde, $\bar{a}_i = 1 - a_i$, reprezintă complementul față de 1 al cifrei a_i ($-m \leq i \leq n-1$).

Exemplu.

$$[+9]_{inv} = 01001;$$

$$[-9]_{inv} = ?$$

Deoarece $-9 = -1001_{(2)}$,

– se trece numărul în cod invers:

$$1001 \rightarrow 0110,$$

– reprezentarea numărului -9 în cod invers va fi:

$$[-9]_{inv} = 10110.$$

Urmărind exemplele prezentate mai sus, se poate observa că, aşa cum s-a mai menționat, cele trei forme de reprezentare coincid în cazul numerelor pozitive și diferă în cazul numerelor negative.

Reprezentarea numerelor în virgulă mobilă

Necesitatea reprezentării în sistemele numerice de calcul a numerelor foarte mari sau foarte mici, cu un grad de precizie ridicat a condus la modul de reprezentare în virgulă mobilă. Acest mod de reprezentare este binecunoscut în sistemul zecimal. Un număr N reprezentat în virgulă mobilă (flotantă) are două componente. Prima E , denumită *exponent*, dă indicații asupra ordinului de mărime al numărului, iar a doua, denumită *mantisă* M , asupra mărimii exacte a numărului într-un anumit domeniu:

$$N = M \times b^E, \quad (\text{C.13})$$

în care b este baza sistemului de numerație (în cazul studiat 2).

Reprezentarea numerelor algebrice în virgulă mobilă este ilustrată în figura C.1. Se observă că numărul este reprezentat pe 32 de poziții binare din care: una este utilizată pentru semnul exponentului, una pentru semnul mantisei, şase pentru exponent și 24 pentru mantisă. În figura C.2 s-au reprezentat numerele $-2^7 \times 0.1011$, $+2^{-4} \times 0.11011$, $-2^{-15} \times 0.111$, $+2^{29} \times 0.10111$. Pentru a se indica semnul exponentului și al mantisei s-a utilizat convenția: 0 pentru semnul „+” și 1 pentru semnul „-”.

SE								SM						...	
0	1	2	3	4	5	6	7							31	

Fig.C.1 Reprezentarea în virgulă mobilă: SE – semnul exponentului; biții 1÷6 – exponentul; SM – semnul mantisei; biții 8÷31 – mantisa.

- (a)

0	0	0	0	1	1	1	1	0	1	1	0	...	0
0	1	2	3	4	5	6	7	8					31

 $N = -2^7 \times 0.1011$
- (b)

1	0	0	0	1	0	0	0	1	1	0	1	1	0	...	0
0					6	7	8						31		

 $N = 2^4 \times 0.11011$
- (c)

1	0	0	1	1	1	1	1	1	1	1	0	...	0
0					6	7	8						31

 $N = -2^{-15} \times 0.111$
- (d)

0	0	1	1	1	0	1	0	1	0	1	1	1	0	...	0
0					6	7	8						31		

 $N = 2^{29} \times 0.10111$

Fig.C.2 Exemple de reprezentare a numerelor în virgulă mobilă.

Coduri

Datorită principiilor de structură și funcționalitate pe baza cărora a fost construit, un calculator numeric nu poate recunoaște decât cifre binare (biți). Datele vehiculate la nivelul extern al sistemului informatic apar însă codificate prin cifre zecimale sau litere ale alfabetului latin. Ca urmare, apare necesitatea folosirii unei noi codificări prin care fiecare simbol alfanumeric să-și găsească exprimarea sub forma unei secvențe binare. Această codificare acționează în procesul transpunerii datelor pe suporti tehnici, astfel încât stocarea, prelucrarea și transferul informației în cadrul sistemului de calcul se face sub forma unor siruri de biți, indiferent de forma externă și de semnificația datelor respective.

Se consideră două mulțimi de elemente, A și B . A codifica elementele mulțimii A prin elementele mulțimii B , înseamnă a face să corespundă fiecarui element $a \in A$ o secvență de elemente $b \in B$.

Dacă se notează cu I mulțimea sirurilor de elemente din B , atunci aplicația care asociază fiecarui element din A un element din I se numește *cod* și se notează cu:

$$f : A \rightarrow I. \quad (\text{C.14})$$

În mod curent, în tehnica de calcul simbolurile mulțimii A (numite și *alfabet*) se numesc *caractere*, ele cuprinzând litere, cifre, semne de punctuație, etc.

La ieșirea din sistem, se aplică un proces de decodificare prin care se revine la formatul extern al datelor.

Codurile în care sunt reprezentate numai numere se numesc *coduri numerice*, iar cele care cuprind și literele și celealte semne se numesc *coduri alfanumerice*.

Având în vedere că într-un calculator numeric se utilizează sistemul binar, în continuare se vor prezenta o serie de coduri care codifică litere sau cifre prin elementele mulțimii $B = \{0,1\}$.

Prima problemă care se pune este: dacă mulțimea A are un număr N de caractere, cât de lungă trebuie să fie secvența $i \in I$ de elemente din B pentru a se putea codifica

toate cele N caractere?

Dacă n este lungimea secvenței i , atunci combinațiile ce se pot face cu două elemente (0 și 1) luate câte n vor fi în număr de 2^n . Deci n trebuie astfel ales încât să fie îndeplinită condiția:

$$N \leq 2^n. \quad (\text{C.15})$$

Coduri numerice

Codurile numerice sau codurile *binar-zecimale*, *BCD* (*Binary Coded Decimal*), asociază fiecarei cifre zecimale o secvență de cifre binare (biți). Aceste coduri se împart în două categorii: *ponderate* și *neponderate*. Un cod binar-zecimal este ponderat dacă unei cifre zecimale îi corespunde o secvență binară în care fiecare rang i are asociată o anumită pondere P_i . Aceasta înseamnă că un număr zecimal:

$$N = Z_m Z_{m-1} \dots Z_k \dots Z_1, \quad (\text{C.16})$$

unde, $Z_k \in \{0,1,2,\dots,9\}$, va avea fiecare cifră reprezentată printr-o secvență binară care satisface relația:

$$Z_k = a_n P_n + a_{n-1} P_{n-1} + \dots + a_i P_i + \dots + a_2 P_2 + a_1 P_1, \quad (\text{C.17})$$

sau

$$Z_k = \sum_{i=1}^n a_i P_i, \quad (\text{C.18})$$

unde, $a_i \in \{0,1\}$, P_i reprezintă ponderea corespunzătoare rangului i , iar n este numărul de biți din secvență binară asociată prin cod cifrei zecimale. Ponderea P_i poate lua una din valorile: $0, \pm 1, \pm 2, \dots, \pm 9$.

În tabelul C.2 se prezintă o serie de coduri binar-zecimale ponderate mai frecvent utilizate.

Codul 8421 se mai numește și *codul binar-zecimal natural (NBCD – Natural Binary Coded Decimal)*¹, deoarece având ca ponderi puterile lui 2, fiecare tetradă de biți reprezintă de fapt exprimarea cifrei zecimale respective în sistemul de numerație binar.

Codul 2421 se numește și *cod Aiken* după numele celui care l-a imaginat și l-a folosit la primele sisteme de calcul automat. Este vorba de profesorul Howard Aiken, de la Universitatea Harvard, care în colaborare cu firma IBM (International Business Machines) a realizat, între anii 1939-1944 o mașină automată de calculat, denumită MARK I, care este precursorul primelor calculatoare electronice ce au apărut în anii imediat următori.

Secvențele de cod Aiken au pentru primele cinci cifre zecimale aceeași exprimare ca și în codul 8421. În continuare, secvența binară corespunzătoare cifrei 5 se obține din cea a cifrei 4 schimbând 0 în 1 și 1 în 0. Aceeași regulă se aplică pentru obținerea secvenței lui 6 din cea a lui 3, 7 din 2, 8 din 1 și 9 din 0. Acest lucru înseamnă în termeni matematici că fiecare complement față de 9 al unei cifre zecimale se exprimă printr-o

¹ În terminologia curentă este denumit (impropriu) doar codul BCD.

secvență ce rezultă complementând față de 1 biții din secvența cifrei zecimale respective.

Codurile care se bucură de această proprietate se numesc *autocomplementare*. Un astfel de cod prezintă avantaje în efectuarea operațiilor aritmetice și simplifică schema constructivă a blocului aritmetic din calculatoarele numerice.

Codul 4221 are aceleași proprietăți ca și codul 2421 (Aiken) prezentat mai sus: utilizează ponderea 2 în două poziții distincte ale tetradei, iar tetradele care reprezintă cifre zecimale a căror sumă este egală cu 9 se complementează reciproc; primele cinci cifre zecimale au pe prima poziție 0, iar ultimele cinci 1.

Codul 5421 are caracteristic faptul că cifrele $5 \div 9$ se deosebesc de cifrele $0 \div 4$ numai prin primul bit.

Codul 7421 utilizează ponderile 7,4,2,1; în aceste condiții se observă că cifra 7 poate fi reprezentată în două moduri (0111 sau 1000) și pentru a se înlătura ambiguitatea s-a introdus o restricție suplimentară și anume: se utilizează din toate combinațiile posibile aceea care corespunde numărului maxim de biți semnificativi.

În tabelul C.2 sunt prezentate, de asemenea, câteva coduri neponderate, create pe baza unor considerente impuse de modul lor de utilizare.

Codul Exces 3 se obține din codul 8421, prin adunarea la fiecare tetradă a cifrei 3 (0011) în binar. Rezultă astfel un cod cu proprietatea de autocomplementare și din care s-a eliminat combinația 0000².

Tab.C.2 Coduri binar-zecimale.

Numere în zecimal	Coduri binar-zecimale							
	Coduri ponderate					Coduri neponderate		
	8421	2421	4221	5421	7421	Exces 3	Gray	2 din 5 74210
0	0000	0000	0000	0000	0000	0011	0000	00011
1	0001	0001	0001	0001	0001	0100	0001	00101
2	0010	0010	0010	0010	0010	0101	0011	00110
3	0011	0011	0011	0011	0011	0110	0010	01001
4	0100	0100	0110	0100	0100	0111	0110	01010
5	0101	1011	1001	1000	0101	1000	0111	01100
6	0110	1100	1100	1001	0110	1001	0101	10001
7	0111	1101	1101	1010	0111	1010	0100	10010
8	1000	1110	1110	1011	1001	1011	1100	10100
9	1001	1111	1111	1100	1010	1100	1101	11000

Codul Gray poartă și el numele celui care l-a imaginat și se caracterizează prin aceea că trecerea de la o cifră zecimală la următoarea se face prin modificarea unui singur rang binar al tetradei. Secvențele codului Gray pot fi deduse din cele ale codului 8421, pe baza următoarelor relații:

$$b_1 = a_1 \oplus a_2, \quad b_2 = a_2 \oplus a_4, \quad b_3 = a_4 \oplus a_8, \quad b_4 = a_8, \quad (C.19)$$

unde, b_1, b_2, b_3 și b_4 sunt pozițiile unei secvențe Gray scrise de la dreapta spre stânga, iar a_1, a_2, a_4 și a_8 sunt pozițiile codului 8421 scrise în ordinea ponderilor.

² Pentru a se putea face distincție între valoarea 0 a unui semnal și absența lui, în circuitele electronice este importantă excluderea combinației 0000.

Codul „2 din 5” este un cod pseudoponderat deoarece pentru cifrele zecimale 1,2,...,9 se pot asocia bițiilor din secvență ponderile 74210; în schimb secvența asociată cifrei zero nu mai respectă această regulă a ponderilor. Caracteristica principală a codului „2 din 5” constă în faptul că toate secvențele binare asociate cifrelor zecimale au același număr de biți semnificativi (1), anume câte 2 (din cele 32 de combinații binare care se pot forma cu 5 biți, numai zece satisfac această condiție), fapt ce a determinat și atribuirea denumirii. Această proprietate oferă un criteriu de depistare a erorilor sau, altfel spus cre-ează posibilitatea controlului asupra transmisiei informației codificate în acest mod.

În codurile BCD, fiecare cifră zecimală este reprezentată, aşa cum s-a menționat, de un cod binar cu patru respectiv cinci biți. Codurile cu patru/cinci biți ale fiecărei cifre zecimale sunt pur și simplu concatenate (șir de grupe în conexiune). De exemplu, numărul $2743_{(10)}$ va avea corespondentul 0010 0111 0100 0011 în codul 8421 (BCD).

Operațiile aritmetice cu numerele codate în această reprezentare se fac ca și cum numerele ar fi binare, însă cu o serie de modificări. Presupunând că ar trebui să adunăm două numere zecimale reprezentate, de exemplu, în codul 8421 (BCD), se pot întâlni următoarele situații:

–dacă fiecare cifră a rezultatului este mai mică decât 10, se adună codurile ca și când acestea ar fi fost binare;

–dacă o cifră a rezultatului este mai mare sau egală cu 10, rezultatul trebuie corectat prin adăugarea unui aşa numit „factor de corecție” de valoare 6 pentru a „sări” peste cele şase valori neutilizate în codul BCD, reprezentate de numerele 10, 11, 12, 13, 14, 15 și anume 1010, 1011, 1100, 1101, 1110 și 1111;

–când rezultatul este 16, 17 sau 18, va fi generat un transport, care va afecta următoarea cifră BCD.

Scăderile pot fi făcute prin scăderea factorilor de corecție când este necesar.

Și în codurile BCD se pot reprezenta numere cu semn, dar în acest caz este necesar, în cuvântul de cod, un bit în plus pentru reprezentarea semnului.

Coduri alfanumerice

Codurile alfanumerice realizează reprezentarea tuturor caracterelor unui alfabet cu ajutorul secvențelor de cifre binare. Alegerea numărului n de poziții binare necesare este o problemă destul de dificilă, acesta trebuind să îndeplinească, pe lângă condiția (C.15) și o serie de condiții legate de asigurarea protecției la transmiterea informației. Există la ora actuală în uz coduri de 6,7 sau 8 biți.

Un exemplu de cod alfanumeric de 8 biți este codul *EBCDIC* (*Extended Binary Coded Decimal Information Interchange Code*) care în traducere înseamnă „Cod binar-zecimal extins, pentru schimburi de informații”, prezentat în tabelul C.4.

Deși codul EBCDIC cunoaște o largă răspândire, acesta nu este standardizat, spre deosebire de codurile *ASCII* (*American Standard Code for Information Interchange*) în traducere „Cod standard U.S.A. pentru schimburi de informații”. Se cunosc două variante de coduri ASCII, ASCII-7 și ASCII-8, având șapte și respectiv opt biți pe caracter. Codul ASCII-7, reprezentat în tabelul C.3, este recomandat și de Organizația Internațională a Standardelor (*International Standards Organisation*), motiv pentru care poartă și denumirea de cod *ISO*.

Codurile EBCDIC și ASCII prezintă un număr suficient de secvențe pentru codificarea literelor, în ambele variante. În plus, mai rămân destul de multe secvențe neasignate

unor caractere alfanumerice. O parte dintre acestea reprezintă codurile unor semnale de comandă și control în cadrul protocolelor de transmitere a datelor (început și sfârșit de mesaj, de transmisie, etc.) sau pentru transmiterea unor comenzi specifice la periferice.

Tab.C.3 Codul ASCII-7.

		765 4321	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	`	p	
0001	1	SOH	DC1	!	1	A	Q	a	q	
0010	2	STX	DC2	“	2	B	R	b	r	
0011	3	ETX	DC3	#	3	C	S	c	s	
0100	4	EOT	DC4	\$	4	D	T	d	t	
0101	5	ENQ	NAK	%	5	E	U	e	u	
0110	6	ACK	SYN	&	6	F	V	f	v	
0111	7	BEL	ETB	,	7	G	W	g	w	
1000	8	BS	CAN	(8	H	X	h	x	
1001	9	HT	EM)	9	I	Y	i	y	
1010	A	LF	SUB	*	:	J	Z	j	z	
1011	B	VT	ESC	+	;	K	[k	{	
1100	C	FF	FS	,	<	L	\	l		
1101	D	CR	GS	-	=	M]	m	}	
1110	E	SO	RS	.	>	N	^	n	~	
1111	F	SI	US	/	?	O	-	o	DEL	

Coduri de comandă

NUL	Nul (Null)	DC4	Comanda 4 a dispozitivului (Device control 4)
SOH	Începutul referinței (Start of heading)	NAK	Confirmare de eroare (Negative acknowledgement)
STX	Începutul textului (Start of text)	SYN	Sincronizare (Synchronization)
ETX	Sfârșitul textului (End of text)	ETB	Sfârșitul blocului transmis (End of text block)
EOT	Sfârșitul transmisiei (End of transmission)	CAN	Anulare (Cancel)
ENQ	Căutare (Enquiry)	EM	Terminarea mijlocului de comunicație (End of medium)
ACK	Confirmare (Acknowledge)	SUB	Înlocuitor (Substitute)
BEL	Avertizare sonoră (Bell)	ESC	Ieșire (Escape)
BS	Spațiu spre stânga (Backspace)	FS	Separator de fișiere (File separator)
HT	Tabulare orizontală (Horizontal tab)	SO	Deplasare către exterior (Shift out)
LF	Salt la linie nouă (Line feed)	SI	Deplasare către interior (Shift in)
VT	Tabulare verticală (Vertical tab)	SP	Spațiu (Space)
FF	Salt la pagină nouă (Form feed)	GS	Separator de grupe (Group separator)
DLE	Ieșire din legătură (Data link escape)	RS	Separator de înregistrări (Record separator)
CR	Început de rând (Carriage return)	US	Separator de unități (Unit separator)
DC1	Comanda 1 a dispozitivului (Device control 1)	DEL	Ștergere (Delete)
DC2	Comanda 2 a dispozitivului (Device control 2)		
DC3	Comanda 3 a dispozitivului (Device control 3)		

Tab.C.4 Codul EBCDIC.

8765 4321	← 00 →				← 01 →				← 10 →				← 11 →			
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000	NUL	DLE	DS		SP	&	-	/	a	j	~		{	}	\	0
0001	SOH	DC1	SOS						b	k	s		A	J		1
0010	STX	DC2	FS	SYN					c	l	t		B	K	S	2
0011	ETX	TM							d	m	u		C	L	T	3
0100	PF	RES	BYP	PN					e	n	v		D	M	U	4
0101	HT	NL	LF	RS					f	o	w		E	N	V	5
0110	LC	BS	ETB	UC					g	p	x		F	O	W	6
0111	DEL	IL	ESC	EOT					h	q	y		G	P	X	7
1000		CAN							i	r	z		H	Q	Y	8
1001		EM											L	R	Z	9
1010	SMM	CC	SM		¢	!		:								
1011	VT	CU1	CU2	CU3	.	\$,	#								
1100	FF	IFS		DC4	<	*	%	@								
1101	CR	IGS	ENQ	NAK	()	-	,								
1110	SO	IRS	ACK		+	;	>	=								
1111	SI	IUS	BEL	SUB		--	?	"								

Coduri detectoare și corectoare de erori

În timpul prelucrării sau transmisiei datelor în sistemul de calcul, pot apărea erori; în scopul detectării și eventual corectării acestor erori se utilizează coduri cu proprietăți speciale. În general, aceste coduri, conțin pe lângă biții care reprezintă informația respectivă și o serie de *biți de control*, cu ajutorul lor identificându-se și corectându-se eventualele erori apărute.

Cel mai simplu mod de detectare a erorilor de transmisie este utilizarea *bitului de paritate*. Prin aceasta se înțelege atașarea la secvența de cod a fiecărui caracter (de exemplu, la codul 8421 sau EBCDIC) a unui bit suplimentar, a cărui valoare se calculează conform convenției: suma tuturor biților din secvență să fie un număr par (respectiv impar).

Dacă un caracter se exprimă printr-o secvență de cod de patru biți în convenție de paritate impară, atunci calculul bitului de paritate se face conform relației:

$$a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus p = 1, \quad (\text{C.20})$$

unde, a_i reprezintă biții informaționali, iar p reprezintă bitul de paritate.

La punctul de emisie al informației se calculează bitul de paritate, iar la punctul de recepție se calculează suma biților care va trebui să fie un număr par sau impar conform convenției. În caz de nesatisfacere a acestei condiții, secvența respectivă este eronată.

Trebuie remarcat că prin această metodă nu se poate identifica poziția binară unde a avut loc eroarea și deci nu poate fi corectată. De asemenea, dacă numărul de biți afectat de erori este par, metoda nu detectează secvența de cod ca eronată.

Codul Hamming. Hamming a dezvoltat un cod capabil să detecteze și să corecteze o

singură eroare. În acest cod un caracter se exprimă prin șapte biți din care patru sunt *biți informationali* iar trei *biți de control*:

$$c = p_1 p_2 a_3 p_4 a_5 a_6 a_7, \quad (\text{C.21})$$

unde p_i cu $i \in \{1,2,4\}$ sunt biți de control, iar a_i cu $i \in \{3,5,6,7\}$ – biți de informație. Biții de control p_i se calculează astfel:

$$\begin{aligned} s_1 &= p_1 \oplus a_3 \oplus a_5 \oplus a_7 = 0, \\ s_2 &= p_2 \oplus a_3 \oplus a_6 \oplus a_7 = 0, \\ s_3 &= p_4 \oplus a_5 \oplus a_6 \oplus a_7 = 0. \end{aligned} \quad (\text{C.22})$$

La punctul de emisie se calculează valorile biților p_i , iar la punctul de recepție se verifică sumele s_i ($i = 1,2,3$). Dacă sumele s_i sunt toate 0, atunci secvența de cod s-a recepționat corect. În caz contrar numărul zecimal codificat binar prin biții s_i de ponderi $2^2, 2^1, 2^0$, indică poziția eronată din secvența de cod transmisă.

Există o mare varietate de coduri cu proprietăți de detecție a erorilor; alegerea unui anumit cod se face în funcție de numărul erorilor care trebuie detectate și numărul biților de control care sunt necesari în acest scop, încrucișând costul transmisiei crește odată cu lungimea secvenței de cod.

ANEXA D

CIRCUITE INTEGRATE DIGITALE DIN SERIA 74 – TEXAS INSTRUMENTS

Compania Texas Instruments, este una dintre primii și cei mai cunoscuți producători de circuite integrate digitale.

Toate dispozitivele TTL/CMOS pe care le vom enumera mai jos (menționăm că nu prezentăm o listă completă a produselor firmei Texas Instruments) sunt desemnate prin coduri de forma „74FAM n ”, „FAM” reprezentând denumirea – formată din litere – a familiei de circuite logice, iar n – un indicativ numeric al funcției realizate (codurile din seria 7400 (realizată în tehnologia TTL *standard*) introduse, pentru porți și alte componente TTL, de către compania Texas Instruments, au ajuns rapid să constituie un *standard industrial*). Dispozitivele aparținând unor familii diferite, dar cu aceeași valoare n realizează aceeași funcție (de exemplu, 74LS30, 74S30, ..., 74HC30, 74HCT30, etc., sunt porți ȘI-NU (NAND) cu 8 intrări). Când un cod este însotit de prefixul „54”, înseamnă că dispozitivul respectiv este conceput pentru a lucra într-o gamă mai largă de temperaturi sau tensiuni de alimentare, pentru aplicații militare (în prezenta anexă ne vom referi doar la varianta comercială a dispozitivelor).

Semnificația notațiilor utilizate în tabele este următoarea:

LS	TTL Schottky de mică putere (Low – power Schottky TTL),
S	TTL Schottky,
ALS	TTL Schottky avansată, de mică putere (Advanced Low – power Schottky TTL),
F	TTL rapidă (Fast TTL),
AS	TTL Schottky avansată (Advanced Schottky TTL),
HC	CMOS de mare viteză (High – speed CMOS),
HCT	CMOS de mare viteză, compatibile cu TTL (High – speed CMOS, TTL compatible),
TP	Totem Pole,
OC	Open Collector,
TSL	Tri/Three State Logic,
XCVR	Transceiver.

PORȚI LOGICE

Număr dispozitiv	Functia	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
00	4 NAND cu 2 intrări	x	x	x	x	x	x	x	x	TP 14	
01	4 NAND cu 2 intrări / OC	x	x		x			x		OC 14	
02	4 NOR cu 2 intrări	x	x	x	x	x	x	x	x	TP 14	
03	4 NAND cu 2 intrări / OC	x	x	x	x			x	x	OC 14	
04	6 porți inversoare	x	x	x	x	x	x	x	x	TP 14	
05	6 porți inversoare / OC	x	x	x	x			x		OC 14	
06	6 porți inversoare / OC/ 30 V	x	x							OC 14	
07	6 porți neinversoare / OC	x	x							OC 14	
08	4 porți AND cu 2 intrări	x	x	x	x	x	x	x	x	TP 14	
09	4 porți AND cu 2 intrări / OC	x	x	x	x	x		x		OC 14	
10	3 porți NAND cu 3 intrări	x	x	x	x	x	x	x	x	TP 14	
11	3 porți AND cu 3 intrări		x	x	x	x	x	x	x	TP 14	
12	3 porți NAND cu 3 intrări / OC	x	x		x					OC 14	
13	2 porți NAND cu 4 intrări Trigger-Schmitt	x	x							TP 14	
14	6 porți inversoare cu intrări Trigger-Schmitt	x	x					x	x	TP 14	
15	3 porți AND cu 3 intrări / OC	x	x	x						OC 14	
16	6 porți inversoare / OC/ 15 V	x								OC 14	
17	6 porți neinversoare / OC	x								OC 14	
18	2 porți NAND cu 4 intrări Trigger-Schmitt		x							TP 14	
19	6 porți inversoare cu intrări Trigger-Schmitt		x							TP 14	
20	2 porți NAND cu 4 intrări	x	x	x	x	x	x	x	x	TP 14	
21	2 porți AND cu 4 intrări		x		x	x	x	x	x	TP 14	
22	2 porți NAND cu 4 intrări / OC	x	x	x	x					OC 14	

0	1	2	3	4	5	6	7	8	9	10
23	2 porți NOR cu 4 intrări, cu STROB	x								TP 16
24	4 porți NAND cu 2 intrări Trigger-Schmitt		x							TP 14
25	2 porți NOR cu 4 intrări, cu STROB	x								TP 14
26	4 porți NAND cu 2 intrări / OC / 15 V	x	x							OC 14
27	3 porți NOR cu 3 intrări	x	x		x	x	x	x	x	TP 14
28	4 porți NOR cu 2 intrări	x	x		x					TP 14
30	Poartă NAND cu 8 intrări	x	x	x	x	x	x	x	x	TP 14
32	4 porți OR cu 2 intrări	x	x	x	x	x	x	x	x	TP 14
33	4 porți OR cu 2 intrări / OC	x	x		x					OC 14
34	6 porți neinversoare				x		x			TP 14
35	6 porți neinversoare / OC				x					OC 14
36	4 porți NOR cu 2 intrări					x		x		TP 14
37	4 porți NAND cu 2 intrări	x	x	x	x	x				TP 14
38	4 porți NAND cu 2 intrări / OC	x	x	x	x	x				OC 14
39	4 porți NAND cu 2 intrări / OC	x								OC 14
40	2 porți NAND cu 4 intrări	x	x	x	x	x				TP 14
50	2 porți AND/NOR cu 2x2 intrări expandabile	x								TP 14
51	2 porți AND/NOR cu 2x2 intrări	x	x	x		x		x		TP 14
53	Poartă AND/NOR cu 4x2 intrări expandabile	x								TP 14
54	Poartă AND/NOR cu 3+2+3+2 intrări	x	x							TP 14
55	Poartă AND/NOR cu 2x4 intrări		x							TP 14
60	2 porți AND/NOR cu 2x2 intrări	x								TP 14
64	Poartă AND/NOR cu 4+2+3+2 intrări			x		x				TP 14
65	Poartă AND/NOR cu 4+2+3+2 intrări / OC			x						OC 14
86	4 porți XOR cu 2 intrări	x	x	x	x	x	x	x	x	TP 14
128	4 porți NOR cu 2 intrări	x								TP 14
132	4 porți NAND cu 2 intrări Trigger-Schmitt	x	x	x				x	x	TP14
133	Poartă NAND cu 13 intrări			x	x			x		TP 16

0	1	2	3	4	5	6	7	8	9	10
134	Poartă NAND cu 12 intrări			x						TSL 16
135	4 porți XOR / NOR			x						TP 16
136	4 porți XOR cu 2 intrări / OC	x	x		x		x			OC 14
140	2 porți NAND cu 4 intrări ($50\ \Omega$)			x						TP14
260	2 porți NOR cu 5 intrări			x						TP 16
266	4 porți NXOR cu 2 intrări / OC		x					x		OC 14
386	4 porți XOR cu 2 intrări		x					x		TP 14
425	4 porți neinversoare	x								TP 14
426	4 porți neinversoare	x								TP 14
618	3 porți NAND cu 4 intrări Trigger-Schmitt		x							TP 14
619	8 porți inversoare cu intrări Trigger-Schmitt		x							TP 14
804	6 porți NAND cu 2 intrări			x		x	x			TP 20
805	6 porți NOR cu 2 intrări			x		x	x	x		TP 20
808	6 porți AND cu 2 intrări			x		x	x	x		TP 20
810	4 porți NXOR cu 2 intrări			x		x				TP 14
811	4 porți NXOR cu 2 intrări / OC			x		x				OC 14
832	6 porți OR cu 2 intrări			x		x	x			TP 20
4049	6 porți inversoare							x		TP 14
4050	6 porți neinversoare							x		TP 14
4078	Poartă AND/NOR cu 4x2 intrări							x		TP 14

MULTIPLEXOARE/ SELECTOARE DE DATE

0	1	2	3	4	5	6	7	8	9	10
151	Multiplexor cu 8 intrări (8/1)	x	x	x	x	x	x	x	x	TP 16
152	Multiplexor cu 8 intrări (8/1)							x		TP 14
153	2 multiplexoare cu 4 intrări ($2 \times 4/1$)	x	x	x	x	x	x	x	x	TSL 16
157	4 multiplexoare cu 2 intrări ($4 \times 2/1$)	x	x	x	x	x	x	x	x	TP 16
158	4 multiplexoare cu 2 intrări ($4 \times 2/1$), 157 inversor	x	x	x	x	x	x	x	x	TP 16
250	Multiplexor cu 16 intrări (16/1)						x			TSL 24
251	Multiplexor cu 8 intrări (8/1)	x	x	x	x	x	x	x	x	TSL 16
253	2 multiplexoare cu 4 intrări ($2 \times 4/1$)		x		x	x	x	x	x	TSL 16
257	4 multiplexoare cu 2 intrări ($4 \times 2/1$)		x	x	x	x	x	x	x	TSL 16
258	4 multiplexoare cu 2 intrări ($4 \times 2/1$), 257 inversor		x	x	x	x	x	x	x	TSL 16
352	2 multiplexoare cu 4 intrări ($2 \times 4/1$), 153 inversor		x		x	x	x	x		TP 16
353	2 multiplexoare cu 4 intrări ($2 \times 4/1$), 253 inversor		x		x	x	x	x		TSL 16
354	Multiplexor cu 8 intrări (8/1)		x					x	x	TSL 20
355	Multiplexor cu 8 intrări (8/1) / OC		x							OC 20
356	Multiplexor cu 8 intrări (8/1)		x					x	x	TSL 20
357	Multiplexor cu 8 intrări (8/1) / OC		x							OC 20
399	4 multiplexoare cu 2 intrări ($4 \times 2/1$) / registru		x							TP 16
604	8 multiplexoare cu 2 intrări ($8 \times 2/1$) / registru		x					x		TSL 28
605	8 multiplexoare cu 2 intrări ($8 \times 2/1$) / registru / OC		x							OC 28
606	8 multiplexoare cu 2 intrări ($8 \times 2/1$) / registru		x							TSL 28
607	8 multiplexoare cu 2 intrări ($8 \times 2/1$) / registru / OC		x							OC 28
850	Multiplexor cu 16 intrări (16/1)					x				TSL 24
851	Multiplexor cu 16 intrări (16/1)					x				TSL 24
857	6 multiplexoare cu 2 intrări ($6 \times 2/1$) / registru				x	x				TSL 24
4051	Multiplexor cu 8 intrări (8/1)							x	x	TSL 16
4052	2 multiplexoare cu 4 intrări ($2 \times 4/1$)							x	x	TSL 16
4053	4 multiplexoare cu 2 intrări ($4 \times 2/1$) / registru							x	x	TSL 16
4067	Multiplexor cu 16 intrări (16/1)							x		TP 24

0	1	2	3	4	5	6	7	8	9	10
4351	Multiplexor cu 8 intrări (8/1)							x	x	TSL 16
4352	2 multiplexoare cu 4 intrări (2×4/1)							x		TSL 16

CONVERTOARE BCD-BINAR

Număr dispozitiv	Functia	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
184	Convertor BCD-binar / OC	x								OC 16	
185	Convertor binar-BCD / OC	x								OC 16	

DECODIFICATOARE/DEMULITIPLEXOARE

Număr dispozitiv	Functia	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
42	Decodificator BCD-zecimal (4/10)	x	x					x	x	TP 16	
43	Decodificator BCD-zecimal (4/10)	x								TP 16	
44	Decodificator BCD-zecimal (4/10)	x								TP 16	
131	Decodificator de 3 biți (3/8) / latch adresabil				x		x			TP 16	
137	Decodificator de 3 biți (3/8) / latch adresabil		x		x		x	x	x	TP 16	
138	Decodificator de 3 biți		x	x	x	x	x	x	x	TP 16	
139	2 decodificatoare de 2 biți (2×2/4)		x	x	x	x	x	x	x	TP 16	
154	Decodificator de 4 biți (4/16)	x						x	x	TSL 24	

0	1	2	3	4	5	6	7	8	9	10
155	2 decodificatoare de 2 biți ($2 \times 2/4$)	x	x							TP 16
156	2 decodificatoare de 2 biți ($2 \times 2/4$) / OC	x	x		x					OC 16
159	Decodificator de 4 biți (4/16) / OC	x								OC 24
237	Decodificator de 3 biți (3/8) / latch adresabil						x	x		TP 16
238	Decodificator de 3 biți (3/8)						x	x		TP 16
239	2 decodificatoare de 2 biți ($2 \times 2/4$)						x			TP 16
4514	Decodificator de 4 biți (4/16) / latch adresabil						x	x		TP 24
4515	Decodificator de 4 biți (4/16) / latch adresabil						x	x		TP 24

DECODIFICATOARE PENTRU AFIȘARE

Număr dispozitiv	Functia	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
45	Decodificator BCD-zecimal / OC / 30 V	x								OC 16	
46	Decodificator BCD-7 segmente / OC / 30 V	x								OC 16	
47	Decodificator BCD-7 segmente / OC / 15 V	x	x							OC 16	
48	Decodificator BCD-7 segmente / 5.5 V	x	x							TP 16	
49	Decodificator BCD-7 segmente / OC / 5.5 V		x							OC 14	
141	Decodificator BCD-zecimal / 60 V	x								OC 16	
145	Decodificator BCD-zecimal / OC / 15 V	x	x							OC 16	
246	Decodificator BCD-7 segmente / OC / 30V	x								OC 16	
247	Decodificator BCD-7 segmente / OC / 15V	x	x							OC 16	
248	Decodificator BCD-7 segmente / 5.5 V		x							TP 16	
249	Decodificator BCD-7 segmente / OC / 5.5 V		x							OC 16	
445	Decodificator BCD-zecimal / OC / 7 V		x							OC 16	

CODIFICATOARE PRIORITARE

Număr dispozitiv	Funcția	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
147	Codificator prioritар cu 10 intrări	x	x					x	x	TP 16	
148	Codificator prioritар cu 8 intrări	x	x			x		x		TP 16	
348	Codificator prioritар cu 8 intrări		x							TSL 16	

COMPARATOARE

Număr dispozitiv	Funcția	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
85	Comparator de 4 biți	x	x	x			x	x	x	TP 16	
518	Comparator de 8 biți / OC				x	x				OC 20	
519	Comparator de 8 biți / OC				x	x				OC 20	
520	Comparator de 8 biți				x	x				TP 20	
521	Comparator de 8 biți				x	x				TP 20	
522	Comparator de 8 biți / OC				x					OC 20	
682	Comparator de 8 biți		x					x		TP 20	
683	Comparator de 8 biți / OC		x							OC 20	
684	Comparator de 8 biți		x					x		TP 20	
685	Comparator de 8 biți / OC		x							OC 20	
686	Comparator de 8 biți		x							TP 24	
687	Comparator de 8 biți / OC		x							OC 24	

0	1	2	3	4	5	6	7	8	9	10
688	Comparator de 8 biți		x		x			x	x	TP 20
689	Comparator de 8 biți / OC		x		x					OC 20
866	Comparator de 8 biți						x			TP 20
885	Comparator de 8 biți cu intrare latch						x			TP 24

DETECTOARE/GENERATOARE DE PARITATE

Număr dispozitiv	Funcția	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
180	Detector/generator de paritate de 8 biți	x						x		TP 14	
280	Detector/generator de paritate de 9 biți		x	x	x	x	x	x	x	TP 14	
286	Detector/generator paritate de 9 biți cu port I/O					x	x			TP 14	

UNITĂȚI ARITMETICE ȘI LOGICE (UAL), GENERATOARE DE FUNCȚII

Număr dispozitiv	Funcția	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
181	UAL / Generator de funcții de 4 biți	x	x	x			x			TP24	
182	Generator de transport anticipat	x		x			x			TP 16	
264	Generator de transport anticipat						x			TP 16	
282	Generator de transport anticipat							x		TP 16	
381	UAL / Generator de funcții de 4 biți		x	x		x				TP 20	

0	1	2	3	4	5	6	7	8	9	10
382	UAL de 4 biți, cu transport serie		x			x				TP 20
881	UAL / Generator de funcții de 4 biți					x				TP 24
882	Generator de transport anticipat de 32 biți					x				TP 24

SUMATOARE

Număr dispozitiv	Funcția	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
82	Sumator complet de 2 biți	x								TP 14	
83	Sumator complet de 4 biți	x	x							TP 16	
183	2 sumatoare complete		x							TP 14	
283	Sumator complet de 4 biți	x	x	x		x		x	x	TP 16	
385	4 sumatoare/scăzătoare seriale		x							TP 20	

MULTIPLICATOARE

Număr dispozitiv	Funcția	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
261	Multiplicator binar paralel 2x4 biți		x							TP 14	
284	Multiplicator binar paralel 4x4 biți	x								TP 16	
285	Multiplicator binar paralel 4x4 biți / OC	x								OC 16	
384	Multiplicator 2 complement		x							TP 16	

DRIVERE, BUS TRANSCEIVERE

Număr dispozitiv	Funcția	Tehnologie								Tipul ieșirii și numărul pinilor	
		bipolară						CMOS			
		TTL	LS	S	ALS	F	AS	HC	HCT		
0	1	2	3	4	5	6	7	8	9	10	
125	4 buffere/drivere neinversoare	x	x			x		x	x	TSL 14	
126	4 buffere/drivere neinversoare	x	x			x		x	x	TSL 14	
230	8 BUS drivere inversoare/neinversoare				x		x			TSL 20	
231	8 BUS drivere inversoare				x		x			TSL 20	
240	8 BUS drivere inversoare		x	x	x	x	x	x	x	TSL 20	
241	8 BUS drivere neinversoare		x	x	x	x	x	x	x	TSL 20	
242	4 BUS XCVR inversoare		x		x	x	x	x	x	TSL 14	
243	4 BUS XCVR neinversoare		x		x	x	x	x	x	TSL 14	
245	8 BUS XCVR neinversoare		x		x	x	x	x	x	TSL 20	
365	6 buffere/drivere neinversoare	x	x					x	x	TSL 16	
366	6 BUS drivere inversoare	x	x					x		TSL 16	
367	6 buffere/drivere neinversoare	x	x					x	x	TSL 16	
368	6 BUS drivere inversoare	x	x					x	x	TSL 16	
440	4 XCVR tridirectionale neinversoare		x							TSL 20	
441	4 XCVR tridirectionale neinversoare / OC		x							OC 20	
442	4 XCVR tridirectionale neinversoare		x							TSL 20	
446	4 BUS XCVR inversoare		x							TSL 16	
448	4 XCVR neinversoare / OC		x							OC 16	
449	4 BUS XCVR neinversoare		x							TSL 16	
540	8 BUS drivere inversoare		x		x	x		x	x	TSL 20	
541	8 BUS drivere neinversoare		x		x	x		x	x	TSL 20	
620	8 BUS XCVR inversoare		x		x	x	x	x	x	TSL 20	
621	8 BUS XCVR neinversoare / OC		x		x	x	x			OC 20	

0	1	2	3	4	5	6	7	8	9	10
622	8 BUS XCVR inversoare / OC		x		x	x	x			OC 20
623	8 BUS XCVR neinversoare		x		x	x	x	x	x	TSL 20
638	8 BUS XCVR inversoare / TSL / OC		x		x		x			TSL 20
639	8 BUS XCVR / neinversoare / TSL / OC		x		x		x			TSL 20
640	8 BUS XCVR inversoare		x		x		x	x	x	TSL 20
641	8 BUS XCVR neinversoare / OC		x		x		x			OC 20
642	8 BUS XCVR inversoare / OC		x		x		x			OC 20
643	8 BUS XCVR neinversoare / inversoare		x		x		x	x	x	TSL 20
644	8 BUS XCVR neinversoare / inversoare / OC		x		x		x			OC 20
645	8 BUS XCVR neinversoare		x		x		x	x	x	TSL 20
756	8 BUS drivere inversoare / OC				x		x			OC 20
757	8 BUS drivere neinversoare / OC						x			OC 20
758	8 BUS XCVR neinversoare / inversoare / OC				x		x			OC 14
759	8 BUS XCVR neinversoare / inversoare / OC						x			OC 14
762	8 BUS drivere neinversoare / inversoare / OC						x			OC 20
763	8 BUS drivere inversoare / OC						x			OC 20

Notă

Conform cu manualul “*Logic Reference Guide-Bipolar, BiCMOS, and CMOS Logic Technology*” al firmei Texas Instruments din 2003 (cf. fișier *scyb04.pdf* descărcat din site-ul Texas Instruments) circuitele din familiile ALS, F, HC și HCT sunt încadrate în zona celor ajunse la maturitate, în timp ce circuitele din familiile TTL, LS, S, AS sunt încadrate în zona de declin dar nescoase încă din fabricație.

BIBLIOGRAFIE

- [1] Angheloiu, I., *Teoria codurilor*, Ed. Militară, Bucureşti, 1972.
- [2] Blakeslee, T.R., *Proiectarea cu circuite logice MSI și LSI standard*, Ed. Tehnică, Bucureşti, 1988.
- [3] Creangă, I., ş.a., *Introducere algebrică în informatică. Teoria automatelor*, Ed. Junimea, Iaşi, 1973.
- [4] Daclin, E., Blanchard, M., *Synthèse des systèmes logiques*, Cepadues – Editions, Toulouse, France, 1976.
- [5] DePalma, R., *Algebra binară a lui Boole și aplicațiile ei în informatică*, Ed. Tehnică, Bucureşti, 1976.
- [6] Floyd, T.L., *Digital Fundamentals*, Merrills, London, 3rd Edition, 1986.
- [7] Hayes, J.P., *Digital System Design and Microprocessors*, McGraw – Hill, New York, 1984.
- [8] Hentea, T., *Automatizări industriale discrete*, Institutul Politehnic “Traian Vuia”, Timişoara, 1981.
- [9] Hill, F.J., Peterson, G.R., *Digital Systems: Hardware Organization and Design*, Wiley, New York, 2nd Edition, 1978.
- [10] Huțanu, C., *Circuite logice și comenzi secvențiale*, Ed. Junimea, Iaşi, 1983.
- [11] Ionescu, D., Nisipeanu, Livia, Stoica, E., *Baze matematice pentru calculatoare numerice*, Ed. Tehnică, Bucureşti, 1978.
- [12] Ionescu, D., *Codificare și coduri*, Ed. Tehnică, Bucureşti, 1981.
- [13] Ivănescu, M., Cauțil, I., *Automate industriale*, Ed. Scrisul Românesc, Craiova, 1984.
- [14] Ivănescu, M., *Roboți industriali*, Ed. Universitară, Craiova, 1995.
- [15] Maican, S., *Sisteme numerice cu circuite integrate. Culegere de probleme*, Ed. Tehnică, Bucureşti, 1980.
- [16] Maican, S., *Circuite integrate digitale*, Ed. Printech, Bucureşti, 1999.
- [17] Mange, D., *Analyse et synthèse des systèmes logiques*, Editions Georgi, Saint-Saphorin, Suisse, 1979.
- [18] Mano, M.M., *Digital Design*, Englewoods Cliff, NJ: Prentice Hall Co., 1991.
- [19] Morris, R.L., Miller, J.R., *Proiectarea cu circuite integrate TTL*, Ed. Tehnică,

- Bucureşti, 1974.
- [20] Muntean, I., *Sinteza automatelor finite*, Ed. Tehnică, Bucureşti, 1977.
- [21] Obermann, R.M.M., *Numărătoare electronice*, Ed. Tehnică, Bucureşti, 1978.
- [22] Pop, V., Popovici, V., *Circuite de comutare aplicate în calculatoarele electronice*, Ed. Facla, Timişoara, 1976.
- [23] Pop, V., *Analiza și sinteza dispozitivelor numerice*, Institutul Politehnic "Traian Vuia", Timişoara, 1986.
- [24] Sprînceană, N., Dobrescu, R., Borangiu, Th., *Automatizări discrete în industrie. Culegere de probleme*, Ed. Tehnică, Bucureşti, 1978.
- [25] Stonham, T.J., *Digital Logic Techniques*, Van Nostrand, Wokingham, 2nd Edition, 1987.
- [26] Sztojanov, I., ş.a., *De la poarta TTL la microprocesor*, Ed. Tehnică, Bucureşti, 1987, vol. I și II.
- [27] Ştefan, Gh. M., ş.a., *Circuite integrate digitale*, Ed. Didactică și Pedagogică, Bucureşti, 1984.
- [28] Ştefan, Gh. M., *Circuite și sisteme digitale*, Ed. Tehnică, Bucureşti, 2000.
- [29] Ştefan, Gh. M., Bistriceanu, V., *Circuite integrate digitale. Probleme, proiectare*, Ed. Albastră, Cluj-Napoca, 2000.
- [30] Toacşe, Gh., Nicula, D., *Electronică digitală*, Ed. Teora, Bucureşti, 1996.
- [31] Zahnd, J., *Machines séquentielles*, Presses Polytechniques Romandes, Lausanne, Suisse, 1987.
- [32] Wiatrowski, C.A., House, C.H., *Logic Circuits and Microcomputer Systems*, Mc Graw – Hill Book Company, New York, 1980.
- [33] Wilkinson, B., *Electronică digitală. Bazele proiectării*, Ed. Teora, Bucureşti, 2002.
- [34] *** Texas Instruments, *Digital Logic Circuits. Catalog*, 2000; <http://focus.ti.com/docs/logic/catalog/>.
- [35] *** Philips, *Fast TTL Logic Series – Data Handbook. Philips Components*, 1997; www.philipslogic.com/products/logic/.
- [36] *** Philips, *Programmable Logic Devices (PLD) – Data Handbook*, Philips Semiconductor, Inc., Sunnyvale, CA, 1994.
- [37] *** Advanced Micro Devices, *Programmable Logic Devices – Handbook/Data Book*, AMD, Inc., Sunnyvale, CA, 1995; <http://www.amd.com>.
- [38] *** Signetics, *Field Programmable Devices. Design Manual*, 1982.
- [39] *** Xilinx, *The Programmable Logic Data Book*, Xilinx, Inc., San Jose, CA, 1998; www.xilinx.com.

INDEX

- ABEL (Advanced Boolean Expression Language), 126
Absorbție, 8
Activarea (validarea) ieșirii, E/OE, 109
Adevărat, 7
Adiacență, 17
Adresă, vezi Intrări de adresă (selecție)
Adunare binară, vezi Sumator binar
Advanced Micro Devices, Inc., 125
Aiken, vezi Cod binar-zecimal 2421
Algebra
 booleană (logică), 9
 logică (booleană), 9
Analiza circuitelor logice combinaționale, 28
Arbore boolean, 31
Arie programabilă, 119
ASCII (American Standard Code for Information Interchange), vezi Cod ASCII
Asociativitate, 8
Axiomele algebrei booleene, 8
- Bait, 109
Bază, vezi Sistem de numerație
BCD (Binary Coded Decimal), vezi Cod binar-zecimal (cod numeric)
Binar
 cifră (bit), 206
 cod, 212
 număr, 206
 sistem de numerație, 206
Bit, 206
 de control, 217
Boole, G., vezi Algebra booleană (logică)
- Canonică, vezi Forma canonică conjunctivă/disjunctivă

- Capacitatea memoriei, 109
Capsulă de circuit integrat, 73
CD, vezi Codificator
Celulă de memorie, 107
Cifra, 205
 cea mai puțin semnificativă (c.m.p.s.), 206
 cea mai semnificativă (c.m.s.), 206
Cip, 73
Circuit
 de anticoïncidență (sumator modulo doi), 166
 de comandă (tampon) cu trei stări, 202
 de comutare, 27
 integrat
 pe scară mare, LSI, 104
 pe scară medie, MSI, 83
 pe scară mică, SSI, 73
 iterativ combinațional, 166
 logic
 combinațional, CLC, 27
 secvențial, CLS, 119
 pentru generarea transportului anticipat, GTA, 170
 tampon (de comandă) cu trei stări, 202
CLC, vezi Circuit logic combinațional
CLS, vezi Circuit logic secvențial
CMOS (Complementary Metal Oxide Semiconductor), 73
Cod, 212
 alfanumeric, 215
 ASCII, 215
 EBCDIC, 215
 binar-zecimal (numeric), BCD, 213
 neponderat, 213
 exces 3, 214
 Gray (reflectat), 214
 reflectat (Gray), 214
 2 din 5 (74210), 215
 74210 (2 din 5), 215
 ponderat, 213
 Aiken (2421), 213
 autocomplementar, 214
 natural (8421), NBCD, 213
 2421 (Aiken), 213
 4221, 214
 5421, 214
 7421, 214
 8421 (natural), NBCD, 213
 complementar (complement față de 2), 210
 corector de erori, 217

- detector de erori, 217
- direct (mărime și semn), 209
- Hamming, 218
- invers (complement față de 1), 211
- numeric (binar-zecimal), BCD, 213
- Codificator, CD, 150
- Codificator prioritar, 153
- Colector
 - deschis (în gol), OC, 82
 - în gol (deschis), OC, 82
- Combinație indiferentă, 61
- Comisia Electrotehnică Internațională (CEI), vezi Norme
- Comparator numeric, 157
- Complement, 9
 - față de 1, 209
 - față de 2, 209
- Complementare (inversare, negație logică, NU), 10
- Comutativitate, 8
- Condensarea diagramelor Karnaugh (metoda diagramelor VID), 53
- Conexiune
 - fixă, 113
 - programabilă, 113
- Conjuncție (intersecție, produs logic, SI), 10
- Consens în raport cu o variabilă, 55
- Consensul
 - implicantilor, 54
 - termenilor, 54
- Constituent al unității (funcția caracteristică a unității), 17
- Constituentul lui zero (funcția caracteristică a lui zero), 18
- Convertor de cod, 142
 - BCD-7 segmente, 148
- Data I/O Corp., 126
- DCD, vezi Decodificator (decodor)
- Decodificator (decodor), DCD, 93
- Decodificator BCD-7 segmente, 148
- Decodor (decodificator), DCD, 93
- De Morgan, vezi Relațiile lui De Morgan
- Demultiplexor (distribuitor), DMUX, 96
- Detector de paritate, 162
- Dezactivarea (invalidarea) ieșirii, OD, 203
- Diagrama
 - de semnal, 111
 - de timp (temporală), 21
 - Karnaugh, 16
 - Veitch, 17
 - VID, 53

- DIN 40700, vezi Norme
Disjuncție (reuniune, SAU, sumă logică), 10
Dispozitiv logic programabil, PLD, 119
Distribuitor (demultiplexor), DMUX, 96
Distributivitate, 9
DMUX, vezi Demultiplexor (distribuitor)
Drenă în gol, OD, 201
Duale, vezi Expresii algebrice duale
- E (Enable), vezi Activarea (validarea) ieșirii
EAROM (Electrical Alterable ROM), vezi Memoria ROM
EBCDIC (Extended Binary Coded Decimal Information Interchange Code), vezi Cod EBCDIC
Echivalente, vezi Expresii algebrice echivalente
EEPROM/E²PROM (Electrically Eraseable Programmable Read Only Memory), vezi Memoria ROM
Element complementar, 9
EPROM (Eraseable Programmable Read Only Memory), vezi Memoria ROM
Esențial, vezi Implicant prim esențial
Exces 3, vezi Cod exces 3
Expresie
 caracteristică, 55
 sumă-de-produse, 119
Expresii algebrice
 duale, 20
 echivalente, 35
- Fals, 7
Familie de circuite
 integrate, 73
 logice programabile, 120
- Forma
 canonică
 conjunctivă, 20
 disjunctivă, 19
 disjunctivă prescurtată, 39
 minimă, 35
 conjunctivă, 45
 disjunctivă, 36
 normal
 conjunctivă, 36
 disjunctivă, 36
- FPAL (Field/Fuse Programmable Array Logic), vezi Matrice programabilă logic
FPGA (Field/Fuse Programmable Gates Array), vezi Matrice de porți programabilă
FPLA (Field/Fuse Programmable Logic Array), vezi Matrice logică programabilă
Funcția
 binară (booleană), 11

- booleană (binară), 11
 - incomplet (parțial) definită, 11
 - parțial (incomplet) definită, 11
 - caracteristică
 - a lui zero (constituentul lui zero), 18
 - a unității (constituent al unității), 17
 - echivalentă, 15
 - implicație, 15
 - inhibare (interdicție), 16
 - interdicție (inhibare), 16
 - logică, 11
 - conjuncție (ȘI), 14
 - constantă 0, 13
 - constantă 1, 13
 - disjuncție (SAU), 14
 - identitate, 13
 - negație, 14
 - SAU-cablat, 199
 - ȘI-cablat, 199
 - lui Pierce (SAU-NU, Webb), 14
 - lui Sheffer (ȘI-NU), 14
 - lui Webb (Pierce, SAU-NU), 14
 - suma modulo doi (SAU-EXCLUSIV), 15
- Functii booleene elementare, 13
- identice, 12
- GAL (Generic Logic Array), vezi Matrice/arie logică generică
- Generator
 - de funcții, 180
 - de paritate, 162
- Graf boolean, 30
- Gray, vezi Cod binar reflectat
- GTA, vezi Circuit pentru generarea transportului anticipat
- Hamming, vezi Cod Hamming
- Hartă de programare, 126
- Hazard combinațional, 133
 - dinamic, 135
 - static, 133
- Hipercub, 46
- HZ (High Z), vezi Stare de înaltă impedanță
- Idempotență, 8
- Identități booleene, 46
- Ieșire
 - cu colectorul în gol, OC, 198
 - cu drena în gol, OD, 201

- cu trei stări logice, TSL, 201
- în contratimp, 197
- Implementare, 69
- Implementarea funcțiilor booleene
 - cu circuite integrate, 71
 - LSI, 104
 - MSI, 83
 - SSI, 73
 - cu decodificatoare și demultiplexoare, 98
 - cu dispozitive logice programabile (PLD), 127
 - cu memorii ROM, 111
 - cu multiplexoare, 86
 - cu relee electromagnetice, 69
- Implicit, 37
 - prim, 37
 - esențial, 39
- Implicația consensului, 54
- Implicație, 36
- Implicații de minimalitate, 56
- Incomplet definită, vezi Funcția incomplet definită
- Intersecție (conjuncție, produs logic, ȘI), 10
- Intrare de strobare (autorizare, validare), 84
- Intrări
 - de adresă (selecție), 83
 - de date, 83
 - de selecție (adresă), 83
- Inversare (complementare, negație logică, NU), 10
- Iterativ, vezi Circuit iterativ combinațional

- Înmulțitor (multiplicator), 188
 - elementar, 190

- Karnaugh, M., vezi Diagrama Karnaugh

- Latice, 8
- Lattice Semiconductor Corp., 125
- Legile
 - lui 0, 9
 - lui 1, 9
- Livovschi, L., 69
- Logică
 - de conexiuni, 69
 - de nivele, 69
 - mixtă, 72
 - negativă, 72
 - pozitivă, 72
- Logigramă (rețea de comutare, schemă logică), 20

LSI (Large Scale Integration), vezi Circuit integrat pe scară mare

Margine (marjă) de zgomot, 72

Marjă (margine) de zgomot, 72

Matrice

de comandă, 95

de decodificare (de referință), 95

de memorie, 109

de porți programabilă, PGA/FPGA, 125

de referință (de decodificare), 95

logică programabilă, PLA/FPLA, 120

programabilă logic, PAL/FPAL, 125

/arie logică generică, GAL, 125

Maxterm (termen canonic disjunctiv, termen maximal), 20

McCluskey, E.J., vezi Metoda de minimizare Quine-McCluskey

Memorie

cu acces aleatoriu, RAM, 106

fixă, ROM, 106

programabilă, PROM, 106

Metoda

consensurilor (a lui Tison), 54

de minimizare bazată pe diagrama Karnaugh, 46

de minimizare Quine-McCluskey, 39

diagramelor VID, vezi Condensarea diagramelor Karnaugh

MIL-STD-806 B(MIL), vezi Norme

Minimizarea

funcțiilor booleene, 35

incomplet (parțial) definite, 60

pe baza axiomelor și teoremelor algebrei booleene, 38

prin metoda consensurilor, 54

prin metoda diagramelor Karnaugh, 46

prin metoda Quine-McCluskey, 38

sistemelor de funcții booleene, 65

Minterm (termen canonic conjunctiv, termen minimal), 19

Moisil, Gr. C., 69

MOS (Metal Oxide Semiconductor), 107

MSI (Medium Scale Integration), vezi Circuit integrat pe scară medie

Multiplexor (selector), MUX, 83

Multiplicator (înmulțitor), 188

elementar, 190

NBCD (Natural Binary Coded Decimal), vezi Cod 8421 (binar-zecimal natural)

Negație logică (complementare, inversare, NU), 10

Nivel logic

al rețelei, 30

0 (L-Low), 72

1 (H-High), 72

- Nivele de tensiune, 72
Norme
 ale Comisiei Electrotehnice Internaționale (CEI), 21
 DIN 40700, 21
 MIL-STD-806 B (MIL), 21
NU (complementare, inversare, negație logică), 10
Numărul de nivele logice al unei rețele de comutare, 30
- OC (Open Collector), vezi Colector în gol (deschis)
OD (Open Drain), vezi Drenă în gol
OD (Output Disable), vezi Dezactivarea (invalidarea) ieșirii
OE (Output Enable), vezi Activarea (validarea) ieșirii
Operator logic
 INVERSOR, 22
 SAU, 22
 SAU-cablat, 199
 SAU-EXCLUSIV, 22
 SAU-NU, 22
 ȘI, 22
 ȘI-cablat, 199
 ȘI-NU, 22
Operator n-ar, 20
- PAL (Programmable Array Logic), vezi Matrice programabilă logic
PALASM 2 (PAL Assembler), 126
Partial definită, vezi Funcția parțial definită
PGA (Programmable Gates Array), vezi Matrice de porți programabilă
Philips, 120
Pierce, vezi Funcția lui Pierce (SAU-NU, Webb)
PLA (Programmable Logic Array), vezi Matrice logică programabilă
PLD (Programmable Logic Device), vezi Dispozitiv logic programabil
Poartă logică
 INVERSOR (INVERTER), 73
 SAU (OR), 73
 SAU-EXCLUSIV (XOR), 73
 SAU-NU (NOR), 73
 ȘI (AND), 73
 ȘI-NU (NAND), 73
 ȘI-SAU-NU (AOI), 73
Pondere, 206
Principiul
 contradicției, 9
 dualității, 8
 dublei negații (involuției), 10
 involuției (dublei negații), 10
 terțului exclus, 9
Produs logic (conjuncție, intersecție, ȘI), 10

- PROM (Programmable Read Only Memory), vezi Memorie fixă programabilă
Proprietatea de minimalitate, 56
Proprietățile algebrei booleene, 8
- Quine, W.V., vezi Teorema lui Quine
- RAM (Random Access Memory), vezi Memorie cu acces aleatoriu
Rang, 206
Rangul termenului normal, 36
Reflectat, vezi Cod binar reflectat (cod Gray)
Relații de minimalitate, 56
Relațiile lui De Morgan, 10
Reprezentarea
 conjunctivă a funcției booleene, 18
 disjunctivă a funcției booleene, 18
 funcțiilor booleene (binare), 16
 analitică, 17
 prin diagramă de timp, 21
 prin diagramă Karnaugh, 16
 prin schemă logică (logigramă, rețea de comutare), 20
 prin tabel de adevăr (combinatorial), 16
numerelor
 în virgulă fixă, 208
 în virgulă mobilă (flotantă), 211
 prin complement față de 1 (cod invers), 211
 prin complement față de 2 (cod complementar), 210
 prin mărime și semn (cod direct), 209
Rețea de comutare (logigramă, schemă logică), 20
Reuniune (disjuncție, SAU, sumă logică), 10
Reziduu în raport cu o variabilă, 55
ROM (Read Only Memory), vezi Memorie fixă
- SAU (disjuncție, reuniune, sumă logică), 10
SAU-EXCLUSIV (suma modulo doi), 15
SAU-NU (funcția lui Pierce/Webb), 14
Scădere binară, 174
Schemă logică (logigramă, rețea de comutare), 20
Selector (multiplexor), MUX, 83
Semisumator, 166
Semnal
 de selecție a cipului, 109
 logic
 activ în 0, 72
 activ în 1, 72
Sheffer, vezi Funcția lui Sheffer (ȘI-NU)
Simboluri grafice, 22
Simplificare

- algebrică, 38
- grafică, 38
- simultană a funcțiilor multiple, 65
- Sinteza circuitelor logice combinaționale, 35
- Sistem
 - complet de funcții (sistem funcțional complet de bază), 23
 - complet de funcții minim, 23
 - de numerație, 205
 - nepozititional, 205
 - pozititional, 205
 - binar, 208
 - funcțional complet de bază (sistem complet de funcții), 23
- SSI (Small Scale Integration), vezi Circuit integrat pe scară mică
- Stare
 - de înaltă impedanță (flotantă), HZ, 109
 - flotantă (de înaltă impedanță), 109
 - logică 0, 84
 - logică 1, 84
- Subcub, 48
- Sumator
 - binar, 165
 - complet (elementar), 166
 - cu propagarea transportului
 - în paralel (cu transport anticipat, cu transport simultan), 168
 - în serie, 168
 - cu transport anticipat (cu propagarea transportului în paralel, cu transport simultan), 168
 - elementar (complet), 166
 - modulo doi (circuit de anticoincidență), 166
 - paralel (derivație), 165
 - serie, 165
- Sumă-de-produse, vezi Expresie sumă-de-produse
- Sumă logică (disjuncție, reuniune, SAU), 10
- ȘI (conjuncție, intersecție, produs logic), 10
- ȘI-NU (funcția lui Sheffer), 14
- Tabel de adevăr (combinațional), 16
- Tabelul
 - acoperirilor, 63
 - implicantilor primi, 39
 - neesențiali, 44
 - reziduurilor, 55
- Teorema lui Quine, 39
- Termen
 - canonic
 - conjunctiv (termen minimal, minterm), 19

- disjunctiv (termen maximal, maxterm), 20
- maximal (termen canonic disjunctiv, maxterm), 20
- minimal (termen canonic conjunctiv, minterm), 19
- normal conjunctiv, 36
- redundant, 49
- Terminal
 - drenă, tranzistorul MOS, 108
 - poartă, tranzistorul MOS, 108
 - sursă, tranzistorul MOS, 108
- Texas Instruments, 74
- Timp de propagare, 133
- Timpul de acces al memoriei, 107
- Tison, vezi Metoda consensurilor
- Transceiver de magistrală, 203
- Transport, 165
- Tranzistor MOS
 - de tip n, 108
 - de tip p, 108
- Trei stări logice, TSL, 82
- TSL (Tri/Three State Logic), vezi Trei stări logice
- TTL (Transistor Transistor Logic), 73
- Unitate aritmetică și logică, UAL, 181
- Variabilă
 - biformă, 55
 - binară (booleană), 10
 - booleană (binară), 10
 - monoformă, 54
 - redundantă, 40
- Veitch, E.W., vezi Diagrama Veitch
- VID (variabile incluse/înglobate în diagramă), vezi Condensarea diagramelor Karnaugh (metoda diagramelor VID)
- Virgulă
 - fixă, vezi Reprezentarea numerelor în virgulă fixă
 - mobilă (flotantă), vezi Reprezentarea numerelor în virgulă mobilă (flotantă)
- Webb, vezi Funcția lui Webb (Pierce, SAU-NU)
- Xilinx, Inc., 126
- Zecimal
 - cifră, 206
 - număr, 206
 - sistem de numerație, 206
- Zgomot, vezi Margine (marjă) de zgomot.