

## **LUCRAREA NR. 4**

### **CIRCUITE LOGICE COMBINAȚIONALE**

#### **1. Scopul lucrării**

Se prezintă circuitele logice combinaționale fundamentale. Se realizează circuite combinaționale pentru implementarea unor funcții booleene în formă canonică și minimală. Se studiază și se verifică funcționarea unor circuite combinaționale: circuit care realizează incrementarea cu 1 a codului BCD, comparator de numere pe 2 biți, sumator de numere pe 2 biți.

#### **2. Considerații teoretice**

Circuitele logice combinaționale constituie clasa dispozitivelor numerice fără memorie, circuite ale căror ieșiri, la un moment dat, sunt complet determinate de intrări. Ele sunt alcătuite din arbori de porți logice elementare. Pentru realizarea lor sunt suficiente unul sau mai multe tipuri de porți logice elementare.

##### **2.1 Logică pozitivă și logică negativă**

La implementarea funcțiilor logice cu dispozitive electronice, acestea operează cu tensiuni și nu cu nivele logice. Există întotdeauna două interpretări ale oricărui tabel de adevăr care descrie funcționalitatea unei porți, bazate pe logica pozitivă și respectiv pe logica negativă. Valorile de tensiune pot fi interpretate ca nivele logice numai prin prisma acestor convenții. Tensiunile de la ieșire sunt fizic aceleași, numai interpretarea logică diferă.

Până acum am presupus că "1" logic este reprezentat prin nivelul de tensiune mai mare decât "0" logic. Această convenție se numește *logică activă pe 1* (*active high*) sau *logică pozitivă*. Când dorim activarea unui anumit semnal (de exemplu "deschide ușa"), aplicăm un nivel de tensiune mai mare (High) pe acea linie de semnal și acesta este interpretat ca "1"

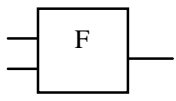


logic. Convenția opusă este însă uneori preferabilă, mai ales atunci când folosim porți ȘI-NU sau SAU-NU pentru implementarea logicii care inițiază evenimentul (*logică de validare*) sau îi împiedică manifestarea sau apariția (*logică de invalidare*). Această convenție se numește *logică activă pe 0* sau *logică negativă*. În acest caz, se folosește nivelul de tensiune coborât (Low) pentru a indica faptul că semnalul este activat, în timp ce nivelul de tensiune mai mare (High) indică faptul că semnalul este inactiv.

În figura 4.1 se prezintă un tabel de adevăr exprimat în termenii a două valori de tensiune relative, High și Low. În interpretarea logicii pozitive, tabelul de adevăr descrie o funcție ȘI, dar în interpretarea logicii negative, obținem funcția SAU.

Fiind dată o funcție în logică pozitivă, putem afla funcția sa echivalentă în logică negativă aplicând teoremele lui De Morgan (4.1):

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

$$\overline{A} + \overline{B} = \overline{A \bullet B} \quad (4.1)$$

								
Tabel de adevăr al tensiunilor			Logică pozitivă			Logică negativă		
A	B	F	A	B	F	A	B	F
low	low	low	0	0	0	1	1	1
low	high	low	0	1	0	1	0	1
high	low	low	1	0	0	0	1	1
high	high	high	1	1	1	0	0	0

**Figura 4.1** Interpretările tabelului de adevăr în logică pozitivă și negativă

Din cauza realelor posibilități de confuzie, este de preferat să se evite folosirea amestecată a logicii pozitive și a celei negative într-un același proiect. Însă acest fapt nu este întotdeauna posibil, de aceea trebuie verificată întotdeauna cu mare atenție convenția folosită pentru fiecare semnal în parte, pentru a evita situații de genul conectării unui semnal de ieșire activ pe "1" la un semnal de intrare activ pe "0".

## 2.2 Funcții incomplet specificate

Până acum am presupus că trebuie să definim o funcție de  $n$  variabile pentru toate cele  $2^n$  combinații posibile ale variabilelor de intrare. În realitate, lucrurile nu stau întotdeauna astfel.

Să considerăm o funcție care are drept intrări un semi-octet în codul BCD. Reamintim că numerele BCD sunt cifre zecimale din intervalul [0-9] care sunt reprezentate de numere binare pe patru biți, folosind combinațiile  $0000_2$  (0) până la  $1001_2$  (9). Celelalte combinații, de la  $1010_2$  (10) până la  $1111_2$  (15) nu vor fi niciodată întâlnite. Putem simplifica expresiile booleene presupunând că funcția are în aceste cazuri un comportament *indiferent* (sau *don't care*).

Tabelul 4.1 reprezintă tabelul de adevăr al unui circuit care realizează *cod BCD incrementat cu 1*. Fiecare număr BCD este reprezentat cu patru variabile booleene, A, B, C și D. Ieșirea circuitului de incrementare este reprezentată de funcții booleene de patru variabile: W, X, Y și Z.

**Tabelul 4.1** Tabelul de adevăr al circuitului cod BCD incrementat cu 1

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	Ø	Ø	Ø	Ø
1	0	1	1	Ø	Ø	Ø	Ø
1	1	0	0	Ø	Ø	Ø	Ø
1	1	0	1	Ø	Ø	Ø	Ø
1	1	1	0	Ø	Ø	Ø	Ø
1	1	1	1	Ø	Ø	Ø	Ø

Valorile funcțiilor sunt "Ø" (*indiferente* sau *don't care*) pentru toate combinațiile variabilelor de intrare care nu apar niciodată. A nu se confunda această valoare "Ø" cu valoarea "0" sau "X" raportată de multe simulatoare logice, unde ea reprezintă o valoare *nedefinită* (sau *don't know*). Orice implementare practică a circuitului va genera totuși o anumită ieșire pentru

cazurile indiferente. Folosind într-un tabel de adevăr valoarea "X" sau "Ø" înseamnă că avem posibilitatea de a alege între a atribui valoarea 0 sau 1 logic respectivei ieșiri din tabelul de adevăr. În general urmărim să alegem acea valoare care va duce la cea mai simplă implementare fizică.

### 2.3 Simplificarea circuitului

Simplificarea circuitului este operația de găsire a unui circuit care este funcțional echivalent cu circuitul dat, dar care este mai simplu într-un anumit sens.

Putem oricând aplica legile algebrei booleene pentru a simplifica o expresie, dar atunci apar mai multe probleme. În primul rând, nu există algoritm care să determine dacă soluția obținută este optimă - atunci înseamnă că nu știm când putem să încetăm să mai căutăm soluții simplificatoare. În al doilea rând, de multe ori este necesar să complicăm expresiile înainte de a le putea simplifica. Este împotriva firii umane de a căuta un "minim local" în speranța găsirii unei soluții globale mai bune, dar este exact ceea ce suntem nevoiți să facem. În fine, este mult prea riscant să manipulăm manual expresii booleene, mai ales de mari dimensiuni.

Deoarece există suficient de multe unelte software de simplificare a expresiilor booleene, de ce trebuie să învățăm metode manuale, mai ales atunci când acestea sunt inaplicabile pentru probleme cu multe variabile (mai mult de șase)? Este totuși necesară cunoașterea principiilor fundamentale utilizate în simplificare. Pe măsură ce instrumentele CAD devin tot mai sofisticate, trebuie să avem o cunoaștere mai profundă a algoritmilor pe care ele le aplică pentru a le putea utiliza efectiv. Și să nu uităm că instrumentele CAD au fost scrise tot de oameni și că ele nu funcționează întotdeauna fără greșală! Trebuie să fim în măsură să verificăm rezultatul generat de aceste instrumente.

Criteriul tradițional al simplificării îl constituie numărul de porți, iar în vederea satisfacerii lui au fost descoperite mai multe metode și algoritmi. Însă odată cu apariția noilor tehnologii, adeseori nu numărul de porți este cel care contează atât de mult, ci numărul sau lungimea firelor de cablaj. Acest aspect schimbă radical procesul de simplificare. Nu vom insista aici asupra acestor criterii; vom prezenta în continuare câteva reguli simple de simplificare, care sunt cel mai des folosite și care sunt adeseori suficiente pentru proiecte mai mici.

a) *introducerea de variabile auxiliare*

Această metodă se mai numește și partajarea semnalelor. Să luăm de exemplu două definiții:

$$\begin{aligned} a &= x \bullet y + z \bullet w \\ b &= \overline{x \bullet y} + \overline{z \bullet w} \quad (4.2) \end{aligned}$$

Vom introduce variabilele auxiliare  $u$  și  $v$  definite astfel:

$$\begin{aligned} u &= x \bullet y \\ v &= z \bullet w \quad (4.3) \end{aligned}$$

Atunci:

$$\begin{aligned} a &= u + v \\ b &= \overline{u} + \overline{v} \quad (4.4) \end{aligned}$$

b) *aplicarea teoremelor algebrei booleene*

Practic, prin aplicarea teoremelor se obține un circuit echivalent (nu neapărat mai simplu). Trebuie însă ținut cont de faptul că în termenii electronicii, porțile ȘI-NU sunt întrucâtva mai simple decât porțile ȘI și SAU, astfel încât uneori aplicarea teoremelor conduce într-adevăr la un circuit mai simplu.

c) *eliminarea termenilor redundanți*

Rezultă în urma aplicării următoarelor echivalențe:

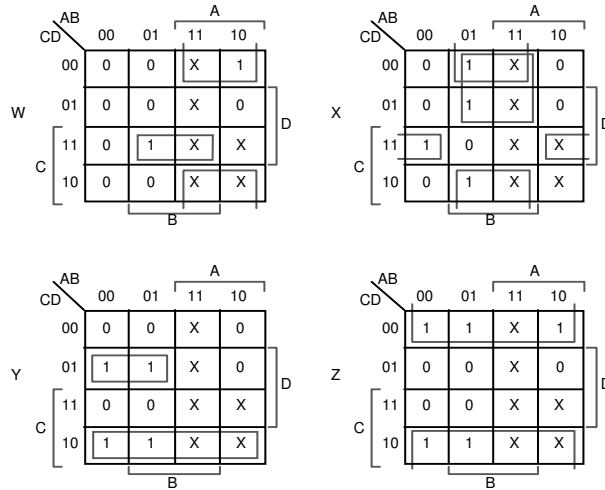
$$\begin{aligned} \overline{\overline{x}} &= x & x + \overline{x} \bullet y &= x + y \\ x \bullet y + \overline{x} \bullet y &= y & x \bullet (\overline{x} + y) &= x \bullet y \quad (4.5) \end{aligned}$$

Există mai multe metode utilizate pentru simplificarea expresiilor booleene, dintre care cele mai cunoscute sunt metoda lui M. Karnaugh (construirea diagramelor Karnaugh) și metoda iterativă Quine-McCluskey (larg răspândită în programele de simplificare automată a expresiilor booleene).

## 2.4 Aplicații: implementarea funcțiilor logice

### 2.4.1 Funcția multidimensională cod BCD incrementat cu 1

Am prezentat în secțiunea 2.3 funcția *cod BCD incrementat cu 1* ca un exemplu de funcție cu combinații indiferente. Pe baza tabelului de adevăr 4.1 se generează diagrame Karnaugh de 4 variabile (figura 4.2):



**Figura 4.2** *Diagramele Karnaugh ale funcției cod BCD incrementat cu 1*

Pentru simplificarea funcțiilor urmărim să realizăm cele mai mari grupări posibile de celule adiacente, profitând de prezența locațiilor indifferente (notate cu x în diagrame) pentru a mări suprafața sub-cuburilor (grupărilor). Obținem următoarele expresii pentru fiecare funcție de ieșire unidimensională (W, X, Y și Z) din componența circuitului:

$$\begin{aligned} W &= B \cdot C \cdot D + A \cdot \bar{D}; & X &= B \cdot \bar{D} + B \cdot \bar{C} + \bar{B} \cdot C \cdot D; \\ Y &= \bar{A} \cdot \bar{C} \cdot D + C \cdot \bar{D}; & Z &= \bar{D}. \end{aligned} \quad (4.6)$$

#### 2.4.2 Comparator de numere pe doi biți

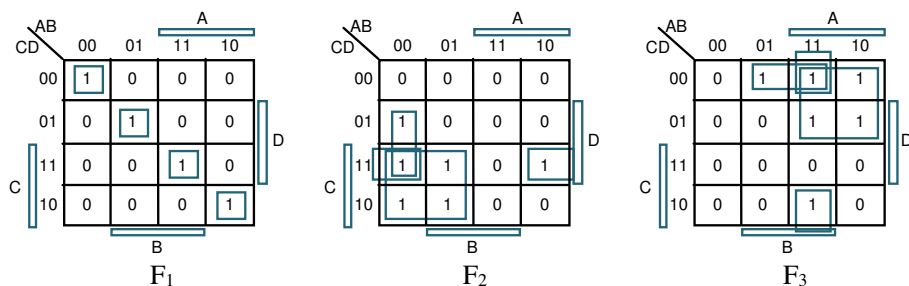
Se cere proiectarea unui circuit care primește la intrare două numere pe doi biți,  $N_1$  și  $N_2$ , și generează trei ieșiri:  $F_1$  dacă  $N_1 = N_2$ ,  $F_2$  dacă  $N_1 < N_2$  și  $F_3$  dacă  $N_1 > N_2$ . Vom nota biții constituenți ai numerelor  $N_1$  și  $N_2$  prin A, B și respectiv C, D.

Primul pas în abordarea problemei este de a înțelege foarte clar funcționalitatea circuitului. Vom construi așadar o schemă bloc a circuitului și vom determina tabelele de adevăr ale funcțiilor (figura 4.3):

	A	B	C	D	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>
	0	0	0	0	1	0	0
	0	0	0	1	0	1	0
	0	0	1	0	0	1	0
	0	0	1	1	0	1	0
	0	1	0	0	0	0	1
	0	1	0	1	1	0	0
	0	1	1	0	0	1	0
	0	1	1	1	0	1	0
	1	0	0	0	0	0	1
	1	0	0	1	0	0	1
	1	0	1	0	1	0	0
	1	0	1	1	0	1	0
	1	1	0	0	0	0	1
	1	1	0	1	0	0	1
	1	1	1	0	0	0	1
	1	1	1	1	1	0	0

**Figura 4.3** Schema bloc și tabelul de adevăr al comparatorului pe 2 biți

În continuare, pe baza tabelului de adevăr, vom obține următoarele diagrame Karnaugh (figura 4.4) pentru ieșiri:



**Figura 4.4** Diagramele Karnaugh pentru comparatorul pe 2 biți

Vom obține următoarele expresii pentru funcții:

$$F_1 = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D$$

$$F_1 = (\overline{A \oplus C}) \cdot (\overline{B \oplus D})$$

$$F_2 = \overline{A} \cdot \overline{B} \cdot D + \overline{B} \cdot C \cdot D + \overline{A} \cdot C$$

$$F_3 = A \cdot \overline{C} + A \cdot B \cdot \overline{D} + B \cdot \overline{C} \cdot \overline{D}$$

(4.7)

unde am simplificat funcția  $F_1$  și mai mult utilizând algebra booleană.

### 2.4.3 Sumator de numere pe doi biți

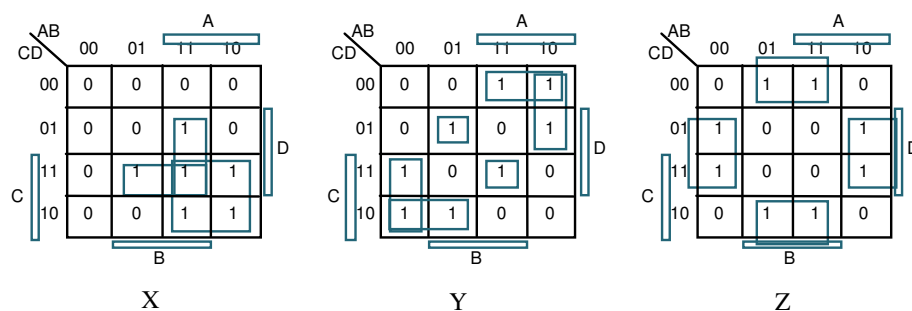
Se cere proiectarea unui circuit care primește la intrare două numere pe doi biți,  $N_1$  și  $N_2$ , și generează la ieșire un număr binar pe 3 biți,  $N_3$ . Și aici, numărul  $N_1$  este reprezentat de biții A și B,  $N_2$  prin C și D, iar  $N_3$  prin funcțiile booleene X, Y și Z, unde X reprezintă bitul de transport (*carry*), iar Y și Z sunt biții propriu-ziși constituenți ai rezultatului.

Vom construi de asemenea o schemă bloc a circuitului și vom determina tabelele de adevăr ale funcțiilor (figura 4.5):

A	B	C	D	X	Y	Z
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

**Figura 4.5** Schemă bloc și tabel de adevăr pentru sumatorul pe 2 biți

În continuare, pe baza tabelului de adevăr, vom obține reprezentările funcțiilor prin următoarele diagrame Karnaugh (figura 4.6) pentru ieșiri:



**Figura 4.6** Diagramele Karnaugh pentru sumatorul pe 2 biți



În urma simplificării vom obține următoarele expresii pentru funcțiile de ieșire:

$$X = A \cdot C + B \cdot C \cdot D + A \cdot B \cdot D$$

$$Z = B \cdot \bar{D} + \bar{B} \cdot D = B \oplus D$$

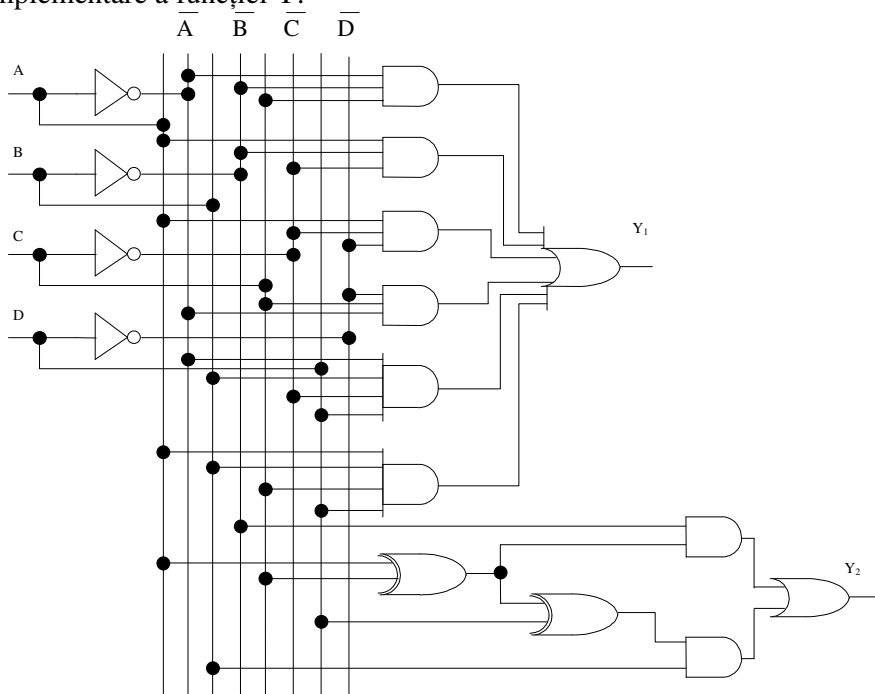
$$Y = \bar{B} \cdot (A \oplus C) + B \cdot (A \oplus C \oplus D)$$

(4.8)

$$\text{sau: } Y = \bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot C \cdot \bar{D} + \bar{A} \cdot B \cdot \bar{C} \cdot D + A \cdot B \cdot C \cdot D$$

Observăm că putem reduce și mai mult expresiile finale dacă utilizăm operatorul SAU-EXCLUSIV. Acest lucru se vede foarte clar dacă examinăm expresia lui Y minimizată cu SAU-EXCLUSIV și apoi minimizată doar în formă de sumă de produse.

În figura 4.7 sunt prezentate cele două variante posibile de implementare a funcției Y:



**Figura 4.7** Două variante de implementare pentru funcția Y

### **3. Desfășurarea lucrării**

1. Se vor realiza și se vor verifica schemele logice care implementează funcțiile W, X, Y și Z ale circuitului cod BCD incrementat cu 1.
2. Se vor implementa funcțiile  $F_1$ ,  $F_2$  și  $F_3$  constituate ale comparatorului de numere binare pe 2 biți, potrivit expresiilor simplificate obținute după minimizare. Se va verifica funcționarea comparatorului. Pot să fie active mai multe semnale de ieșire simultan?
3. Se vor implementa funcțiile X și Z constituate ale sumatorului de numere binare pe 2 biți, conform expresiilor obținute după minimizare. Se va verifica funcționarea sumatorului.
4. Se va implementa funcția Y din cadrul aceluiași sumator potrivit celor două scheme din figura 4.7. Care variantă este mai avantajoasă?
5. Se va construi un convertor de cod din 8421 (cod BCD) în 2421, parcurgându-se toți pașii succesivi prezentați la comparator și la sumator. Se vor implementa funcțiile constituate ale convertorului și se vor verifica din punct de vedere funcțional.
6. Pentru problemele enunțate anterior, se va realiza implementarea numai cu porți ȘI-NU; cu porți SAU-NU; cu porți ȘI-SAU-NU. Comparați implementările respective. Care este implementarea cea mai avantajoasă din punctul de vedere al prețului, al numărului de porți și a suprafeței ocupate de circuitele integrate?