## *CAPITOLUL 4*

# *Circuite logice combinaționale*

Circuitele logice combinaționale (c.l.c.) sunt circuite fără memorie (independente de propriile stări anterioare), caracterizate prin faptul că semnalele de ieşire sunt combinații logice ale semnalelor de intrare, existând numai atâta timp cât acestea din urmă există.

Schema bloc a unui circuit logic combinațional este dată în fig. 4.1, iar funcțiile de ieşire ale acestuia pot fi scrise sub forma:

$$y_k = y_k (x_1, x_2, \dots , x_n), \qquad (4.1)$$

cu $k = 1, 2, \dots , m$.

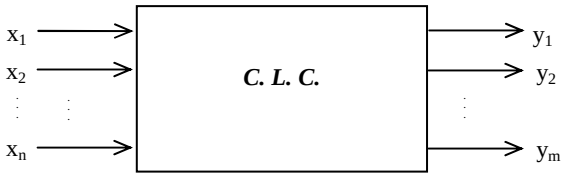



Fig. 4.1. Schema bloc a unui c.l.c.

Independența față de timp a relațiilor 4.1 ar putea fi interpretată ca un răspuns instantaneu şi simultan al ieşirilor circuitului logic combinațional la o modificare simultană a intrărilor acestuia.

În realitate, situația este puțin mai complicată.

Ținând seama de faptul că un c.l.c. reprezintă un ansamblu de porți logice elementare interconectate între ele în diverse moduri, astfel încât informațiile prezente la intrări parcurg, de regulă, în drumul lor către ieşiri, un număr variabil de porți logice elementare, rezultă că efectul modificării valorilor logice ale intrărilor c.l.c. se propagă către ieşiri în intervale de timp diferite, întotdeauna multipli de $t_{pd}$.

Presupunând că cea mai scurtă *cale* intrare-ieşire parcurge *a* porți, iar cea mai lungă – *b* porți, înseamnă că vectorul ieşirilor va începe să varieze la un interval de timp $a \cdot t_{pd}$ după modificarea vectorului de intrare şi se va stabiliza abia după un interval de timp $b \cdot t_{pd}$ de la momentul respectiv.

Prin urmare, în intervalul $(b-a) \cdot t_{pd}$, vectorul de ieşire înregistrează variații neconforme cu relația 4.1, cunoscute sub denumirea de *hazard combinațional* sau *hazard logic*.

Eliminarea inconvenientelor pe care le implică hazardul logic poate fi realizată numai printr-o proiectare riguroasă care constă fie în asigurarea unor întârzieri

egale pe toate *căile* intrare-ieşire, fie prin citirea informaţiilor de la ieşirea circuitului numai după terminarea intervalului $(b\text{-}a)\cdot t_{pd}$, corespunzător procesului tranzitoriu.

## *4.1. Analiza şi sinteza circuitelor logice combinaționale*

În legătură cu circuitele logice combinaționale, se pun de regulă două probleme importante şi anume: analiza şi sinteza c.l.c.

### *4.1.1. Analiza circuitelor logice combinaționale*

Analiza c.l.c. porneşte de la schema logică cunoscută a circuitului şi urmăreşte stabilirea modului de funcţionare a acestuia, fie prin construirea tabelului de funcționare, fie prin scrierea formei analitice a funcției de ieşire.

Spre exemplu, pornind de la schema logică a unui c.l.c. simplu, fig. 4.2, deducem din aproape în aproape, urmărind transformările semnalelor de intrare, expresia analitică a funcției de ieşire:

$$Y = \overline{A}B + A\overline{B} \tag{4.2}$$

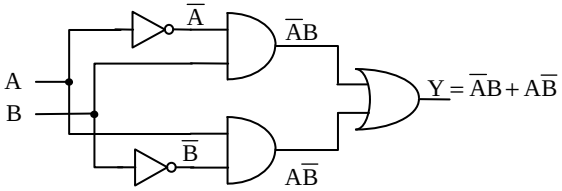



Fig. 4.2. Schema logică a unui XOR

Construirea tabelului de funcționare este acum extrem de simplă şi urmează paşii prezentați în coloanele tabelului 4.1.

Tab. 4.1. Tabelul de funcționare al c.l.c. din fig. 4.2

| $B$ | $A$ | $\overline{B}$ | $\overline{A}$ | $\overline{A}B$ | $A\overline{B}$ | $Y = \overline{A}B + A\overline{B}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Recunoaştem funcția de ieşire şi tabelul de funcționare al circuitului SAU-EXCLUSIV (XOR).

### *4.1.2. Sinteza circuitelor logice combinaționale*

Sinteza c.l.c. porneşte de la funcția pe care trebuie să o îndeplinească circuitul şi îşi propune obținerea unei variante (minimale) a structurii acestuia.

Etapele sintezei sunt: definirea funcției (funcțiilor) de ieşire, minimizarea şi, în final, desenarea schemei circuitului.

După modul în care este scrisă funcția, implementarea se poate face în diverse variante dintre care menționăm:

a) cu orice combinație de circuite logice elementare;

b) numai cu circuite NAND;

c) numai cu circuite NOR.

Spre exemplu, considerând funcția:

$$Y = A \oplus B \tag{4.3}$$

şi tabelul ei de funcționare, tab. 4.2, ne propunem să realizăm sinteza circuitului corespunzător în mai multe variante.

Tab. 4.2. Tabelul de adevăr al funcției XOR

| ***B*** | ***A*** | ***Y*** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

a) Sinteza utilizând mai multe tipuri de circuite logice elementare

Pornind de la tab. 4.2, observăm că forma canonică disjunctivă (FCD) a funcției este cea exprimată de relația 4.2. Fiind o formă deja minimală, implementarea ei conduce la circuitul din fig. 4.2.

Procedând similar, dar utilizând forma canonică conjunctivă (FCC), obținem:

$$Y = (A + B) \cdot (\overline{A} + \overline{B}), \tag{4.4}$$

care în urma implementării conduce la circuitul din fig. 4.3.

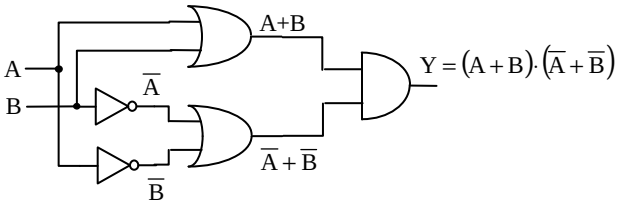



Fig. 4.3. O altă variantă de implementare a XOR-ului

b) Sinteza numai cu porţi NAND

Aplicând De Morgan asupra FCD, rel. 4.2, obţinem:

$$\overline{\overline{Y}} = \overline{\overline{A}B + A\overline{B}} = \overline{\left(\overline{\overline{A}B}\right)\cdot\left(\overline{A\overline{B}}\right)}, \qquad (4.5)$$

a cărei implementare poate fi realizată numai cu NAND-uri şi conduce la circuitul din fig. 4.4.

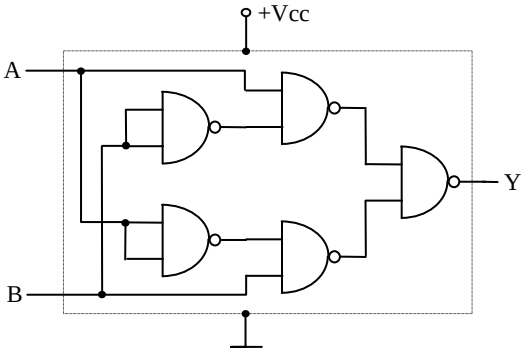



Fig. 4.4. Implementarea XOR-ului numai cu NAND-uri

c) Sinteza numai cu porţi NOR

Aplicând De Morgan asupra FCC, rel. 4.4, obţinem:

$$\overline{\overline{Y}} = \overline{(A+B)\cdot(\overline{A}+\overline{B})} = \overline{(\overline{A+B})} + \overline{(\overline{\overline{A}+\overline{B}})}, \qquad (4.6)$$

a cărei implementare poate fi făcută numai cu NOR-uri şi conduce la circuitul din fig. 4.5.

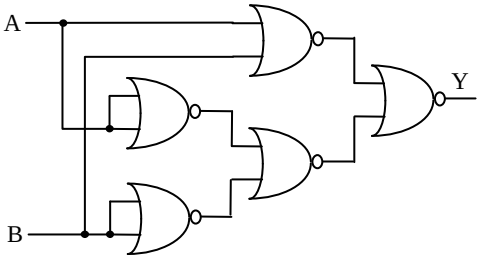


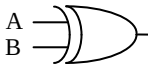

Fig. 4.5. Implementarea XOR-ului numai cu NOR-uri

În cele ce urmează, ne propunem prezentarea sintezei celor mai importante circuite logice combinaţionale utilizate în electronica digitală.

## *4.2. Detectorul de paritate*

Detectorul de paritate este un circuit logic combinațional care are rolul de a determina *paritatea* sau *imparitatea* numărului de variabile de intrare egale cu 1 logic. El are la bază unele proprietăți ale funcției SAU-EXCLUSIV (XOR).

Din motive legate de simplitatea expunerii, vom considera pentru început poarta XOR cu două intrări, fig. 4.6.




$$Y = A \oplus B = \overline{A}B + A\overline{B}$$

Fig. 4.6. Poarta logică XOR

După cum se poate observa din tabelul de adevăr al funcției XOR de 2 variabile, tab. 4.2, la ieşirea circuitului din fig. 4.6 se obține 1 logic când intrările sunt diferite (01 sau 10, deci un număr impar de intrări este 1 logic) şi 0 logic - când intrările coincid (00 sau 11, deci un număr par de intrări este1 logic).

Spunem că circuitul XOR cu două intrări este un detector de paritate impară, sau, mai simplu, un detector de imparitate.

Prezentăm în continuare câteva proprietăți ale funcției logice XOR, care urmează a fi folosite la sinteza detectorului de imparitate cu mai multe intrări.

Proprietatea nr. 1 (asociativitatea funcției XOR):

$$Y = (A \oplus B) \oplus C = A \oplus (B \oplus C). \qquad (4.7)$$

Demonstrație:

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

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

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

Proprietatea nr. 2 (oricare ar fi numărul de intrări al unei porți XOR, ieşirea Y=1/0 dacă un număr impar/par de variabile de intrare este egal cu 1):

$$\underbrace{1 \oplus 1 \oplus ... \oplus 1}_{\text{nr. par de "1"}} \oplus \underbrace{0 \oplus 0 \oplus ... \oplus 0}_{\text{nr. oarecare de "0"}} = 0; \qquad (4.8)$$

$$\underbrace{1 \oplus 1 \oplus ... \oplus 1}_{\text{nr. impar de "1"}} \oplus \underbrace{0 \oplus 0 \oplus ... \oplus 0}_{\text{nr. oarecare de "0"}} = 1. \qquad (4.9)$$

Demonstrația se bazează pe tabelul de adevăr al funcției XOR, tab. 4.2.

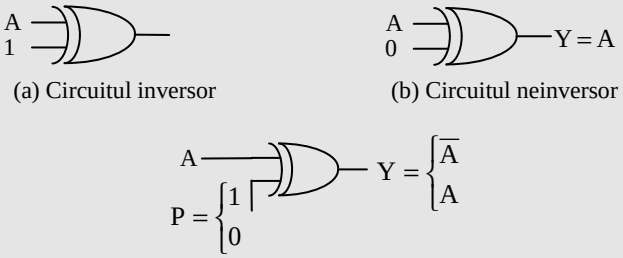
Pentru o mai bună edificare asupra acestei proprietăți, pot fi construite tabeluri de adevăr ale funcției XOR cu 3 şi 4 variabile (v. tab. 4.3).

Proprietatea nr. 3 (utilizarea XOR-ului ca circuit inversor/neinversor comandat):

$$A \oplus 1 = \overline{A} \cdot 1 + A \cdot \overline{1} = \overline{A}; \qquad (4.10)$$

$$A \oplus 0 = \overline{A} \cdot 0 + A \cdot 0 = A. \qquad (4.11)$$

Ilustrarea relațiilor 4.10 şi 4.11 este prezentată în fig. 4.7 a şi b, care cumulate, conduc la schema circuitului inversor / neinversor comandat din fig. 4.7 c.




(a) Circuitul inversor (b) Circuitul neinversor

(c) Circuitul inversor / neinversor comandat

Fig. 4.7. Ilustrativă pentru proprietăţile funcţiei XOR

### *4.2.1. Detectorul de paritate impară cu 4 variabile de intrare*

Pornind de la tabelul de adevăr, tab. 4.3, în care valorile logice din coloanele Y au fost obţinute ţinând seama de proprietăţile (1) şi (2) ale XOR-ului, rezultă pentru circuit două variante de implementare.

Varianta prezentată în fig. 4.9 prezintă avantajul unor întârzieri egale cu $2 \cdot t_{pd}$ pe toate căile intrare-ieşire, fapt care face să dispară pericolul hazardului logic.

Tab. 4.3. Tabelul de adevăr al detectorului de imparitate

| ***Var. intrare*** | | | | ***Y=[(A ⊕ B) ⊕ C] ⊕ D*** | | | ***Y=(A ⊕ B) ⊕ (C ⊕ D)*** | | |
|---|---|---|---|---|---|---|---|---|---|
| ***D*** | ***C*** | ***B*** | ***A*** | $Y_{AB}= A \oplus B$ | $Y_{ABC}= Y_{AB} \oplus C$ | $Y= Y_{ABC} \oplus D$ | $Y_{AB}= A \oplus B$ | $Y_{CD}= C \oplus D$ | $Y= Y_{AB} \oplus Y_{CD}$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

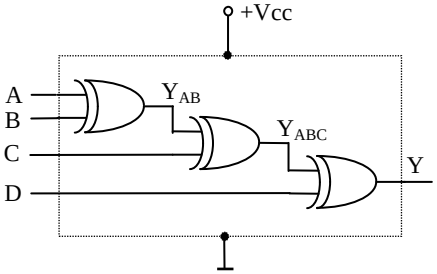



Fig. 4.8. Schema detectorului de paritate impară - varianta 1

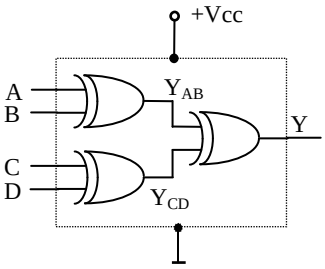



Fig. 4.9. Schema detectorului de paritate impară - varianta 2

### *4.2.2. Detectorul de paritate comandat*

În sinteza detectorului este necesar să se țină seama de următoarele condiții:

1) Transformarea detectorului de paritate impară, fig. 4.9, în detector de paritate pară, trebuie realizată prin schimbarea valorii logice a unei singure "bare" de comandă, în maniera prezentată în fig. 4.7.

2) Indiferent de regimul de "imparitate" sau "paritate" în care lucrează detectorul, ieşirea acestuia trebuie să fie "1" logic în momentul detecției. Rezultă pentru regimul de "imparitate", $Y' = Y$, iar pentru regimul de "paritate", $Y' = \overline{Y}$ (v. tab. 4.4). Este deci necesară utilizarea proprietății (3) de maniera din fig. 4.10.

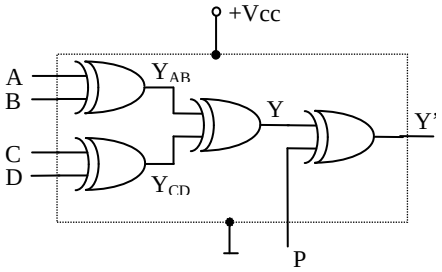



Fig. 4.10. Schema detectorului de paritate comandat

Într-adevăr,

$$Y' = Y \oplus P = \begin{cases} Y, \text{ pentru } P = 0 \text{ (detector de imparitate);} \\ \overline{Y}, \text{ pentru } P = 1 \text{ (detector de paritate).} \end{cases} \qquad (4.12)$$

Pentru confirmarea acestor rezultate, prezentăm tabelul de adevăr 4.4 al detectorului de paritate comandat. Acest tabel reia practic de două ori primele 4 coloane şi ultima din tab. 4.3: o dată pentru P=0 şi a doua oară pentru P=1. În final, este adăugată coloana Y', obținută prin aplicarea relației 4.12.

Tab. 4.4. Tabelul de adevăr al detectorului de paritate comandat

| ***D*** | ***C*** | ***B*** | ***A*** | ***Y*** | ***P*** | ***Y'*** |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |

*O aplicație importantă* a detectorului de paritate o constituie *controlul de paritate al transmisiunilor de date*, capabil să detecteze erorile de transmisie şi să declanşeze o procedură de corecție a acestora.

Astfel, considerând că informația care se transmite prin magistrala de date se compune din cuvinte a câte 4 biți, fig. 4.10, fiecărui cuvânt i se adaugă la emisie (E)

un al 5-lea bit de control la paritate furnizat de către un detector de paritate cu 4 intrări, DP-I. În acest mod, pe cele 4+1 linii de transmitere a informației vom avea în fiecare moment câte un cuvânt de cod format din 5 biți, în componența cuvântului respectiv existând întotdeauna un număr par de biți egali cu 1 logic.

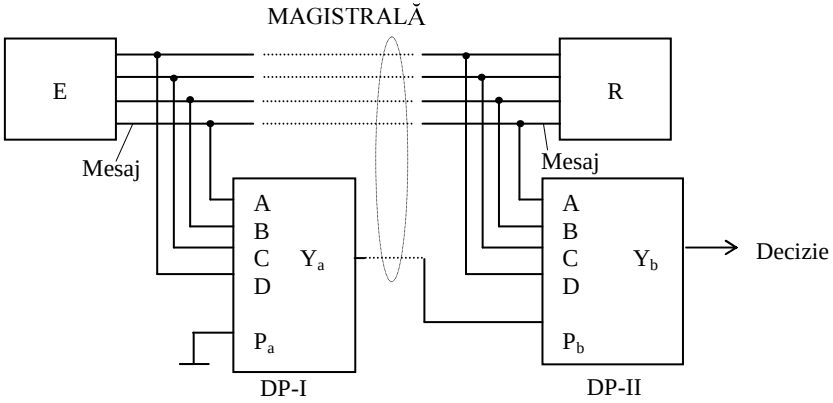



Fig. 4.10. Detectarea erorilor de transmisie a informațiilor binare

La receptorul R există un alt detector de paritate cu 5 intrări, DP-II, la ieşirea căruia se va obține 1 logic în cazul în care transmisia de date a fost corectă (număr par de 1 logic pe cele 5 linii) şi 0 logic dacă aceasta a fost perturbată. Evident, în acest din urmă caz, se ia decizia blocării execuției şi a corecției erorii apărute prin metode specifice, cum ar fi transmiterea repetată a informației.

## *4.3. Multiplexoare*

Multiplexoarele (MUX-urile) sunt circuite logice combinaționale care permit trecerea datelor de la una din cele *n* intrări spre ieşirea unică, fig. 4.11.

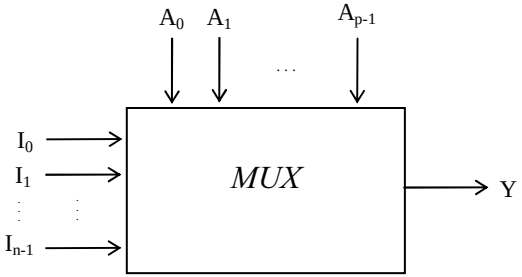



Fig. 4.11. Schema bloc generală a unui multiplexor

Selecția intrării care urmează a avea acces la ieşire se face printr-un cuvânt de cod (adresă) având *p* biți.

Se observă că $n=2^p$, adică numărul de intrări este egal cu numărul combinațiilor logice de adresă a căror apariție urmează să autorizeze accesul succesiv al intrărilor către ieşire.

### *4.3.1. Circuitul de multiplexare cu 4 intrări*

În cazul MUX-ului cu n=4 intrări ($I_0$, $I_1$, $I_2$, $I_3$), numărul barelor de adresă este p=2 ($A_0$, $A_1$).

Pornind de la definiția multiplexorului, construim tabelul de funcționare al unui MUX cu 4 intrări, tab. 4.5, scriem forma canonică disjunctivă, rel. 4.13, şi o implementăm în fig. 4.12.

Tab. 4.5. Tabelul de funcționare al unui MUX cu 4 intrări

| $\overline{E}$ | $A_1$ | $A_0$ | $I_0$ | $I_1$ | $I_2$ | $I_3$ | $Y$ |
|---|---|---|---|---|---|---|---|
| 1 | x | x | x | x | x | x | 0 |
| 0 | 0 | 0 | $I_0$ | x | x | x | $I_0$ |
| 0 | 0 | 1 | x | $I_1$ | x | x | $I_1$ |
| 0 | 1 | 0 | x | x | $I_2$ | x | $I_2$ |
| 0 | 1 | 1 | x | x | x | $I_3$ | $I_3$ |

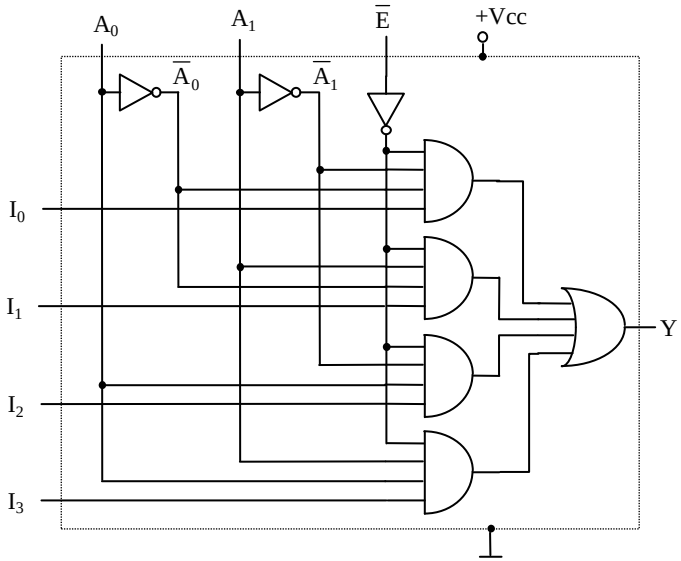



Fig. 4.12. MUX-ul cu 4 intrări

$$Y = E(\underbrace{\overline{A}_1\overline{A}_0}_{P_0} I_0 + \underbrace{\overline{A}_1 A_0}_{P_1} I_1 + \underbrace{A_1\overline{A}_0}_{P_2} I_2 + \underbrace{A_1 A_0}_{P_3} I_3). \quad (4.13)$$

Observăm că schema este prevăzută şi cu o intrare de autorizare $\overline{E}$ ($\overline{ENABLE}$), activă în starea "L". Pentru $\overline{E} = 1$, indiferent de stările logice ale intrărilor şi barelor de adresă, ieşirea se fixează în 0 logic şi MUX-ul este inactivat.

## *4.4. Demultiplexoare*

Circuitele de demultiplexare (DMUX-urile) sunt c.l.c. care permit transmiterea datelor de la o intrare unică, la una din cele m ieşiri selectate printr-un cuvânt de cod (adresă).

Schema bloc a unui DMUX cu m ieşiri şi p bare de adresă ($m=2^p$) este prezentată în fig. 4.13.

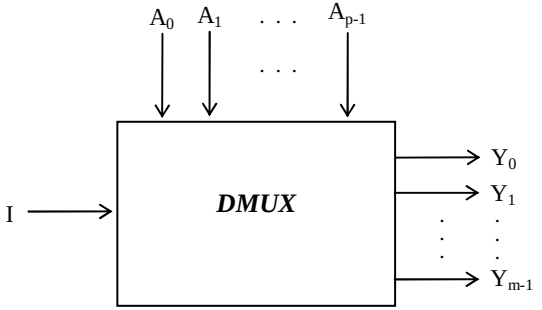



Fig. 4.13. Schema bloc generală a unui DMUX

### *4.4.1. Circuitul de demultiplexare cu 4 ieşiri*

Circuitul de demultiplexare cu m=4 ieşiri ($Y_0$,$Y_1$, $Y_2$, $Y_3$), are p=2 bare de adresă ($A_0$,$A_1$).

Tab. 4.6. Tabelul de funcționare al unui DMUX cu 4 ieşiri

| $A_1$ | $A_0$ | $I$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | I | I | 0 | 0 | 0 |
| 0 | 1 | I | 0 | I | 0 | 0 |
| 1 | 0 | I | 0 | 0 | I | 0 |
| 1 | 1 | I | 0 | 0 | 0 | I |

Pornind de la tabelul de funcţionare al unui astfel de circuit, tab. 4.6, se scriu funcţiile de ieşire:

$$Y_0 = I \cdot \overline{A_1}\,\overline{A_0}, \quad Y_1 = I \cdot \overline{A_1} A_0, \quad Y_2 = I \cdot A_1 \overline{A_0}, \quad Y_3 = I \cdot A_1 A_0, \qquad (4.14)$$

şi se obţine varianta de implementare din fig. 4.14.

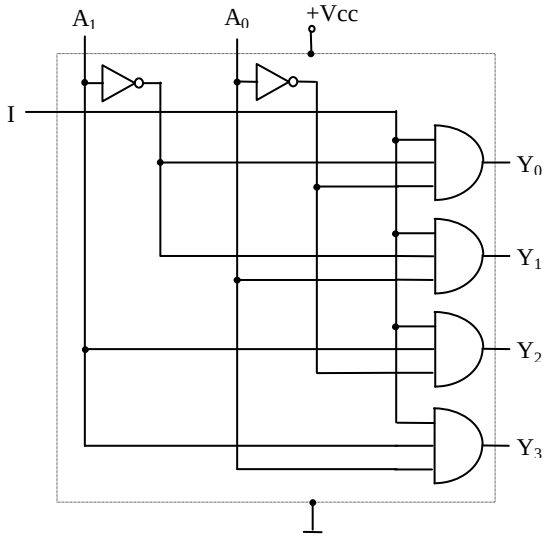



Fig. 4.14. DMUX-ul cu 4 ieşiri

## *4.5. Comparatoare numerice*

Comparatoarele numerice sunt c.l.c. care permit determinarea valorii relative a două numere exprimate în cod binar.

Schema bloc a unui comparator de n biţi este prezentată în fig. 4.15.

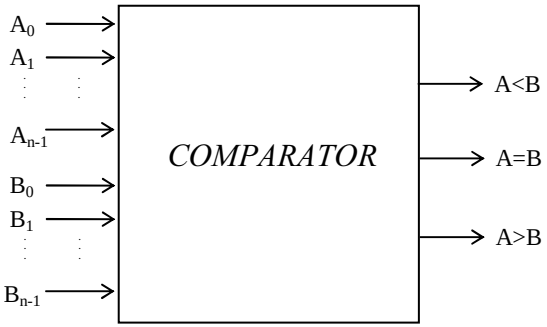



Fig. 4.15. Schema bloc a unui comparator de n biţi

### *4.5.1. Comparatorul numeric de un bit*

Comparatorul numeric de un bit prezintă schema bloc din fig. 4.16.

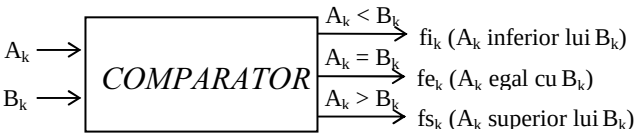



Fig. 4.16. Schema bloc a comparatorului de 1 bit

Compararea celor două numere de câte un bit fiecare, permite definirea următoarelor funcții, v. tab. 4.7:

- funcția de inferioritate, $f_{i_k} = \overline{A_k} B_k$, care ia valoarea logică 1 numai când $A_k<B_k$, adică atunci când $A_k=0$ şi $B_k=1$;
- funcția de egalitate, $f_{e_k} = \overline{A_k \oplus B_k}$, care ia valoarea logică 1 numai când $A_k=B_k$, adică fie $A_k=B_k=0$, fie $A_k=B_k=1$ logic;
- funcția de superioritate, $f_{s_k} = A_k \overline{B_k}$, care ia valoarea logică 1 numai când $A_k>B_k$.

Sintetic, putem scrie:
$$\begin{cases} \overline{A_k} B_k = 1 & \text{pentru } A_k < B_k; \\ \overline{A_k \oplus B_k} = 1 & \text{pentru } A_k = B_k; \\ A_k \overline{B_k} = 1 & \text{pentru } A_k > B_k, \end{cases} \qquad (4.15)$$

relații care ne ajută să construim tabelul de funcționare al comparatorului de 1 bit, tab. 4.7.

Tab. 4.7. Tabelul de funcționare al comparatorului de 1 bit

| | | $fi_k$ | $fe_k$ | $fs_k$ |
|---|---|---|---|---|
| $A_k$ | $B_k$ | $\overline{A_k} B_k$ | $\overline{A_k \oplus B_k}$ | $A_k \overline{B_k}$ |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| | | $A_k<B_k$ | $A_k=B_k$ | $A_k>B_k$ |

Pornind de la tabelul de funcționare, tab. 4.7, în care coloanele 3, 4 şi 5 reprezintă ieşirile comparatorului de 1 bit pentru cele 3 situații posibile rezultate în urma comparării, se obține varianta de implementare din fig. 4.17.

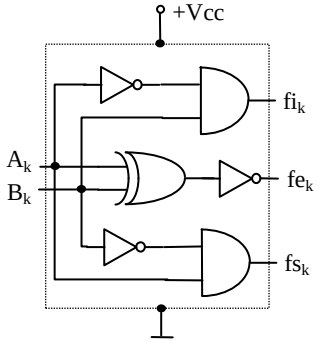



Fig. 4.17. Schema logică a comparatorului de 1 bit

### *4.5.2. Comparatorul numeric de 4 biţi*

Se poate obţine prin interconectarea a patru comparatoare de un bit.

Cele două numere de câte 4 biţi fiecare se pot scrie astfel:

$$A = 2^3A_3+2^2A_2+2^1A_1+2^0A_0;$$

$$B = 2^3B_3+2^2B_2+2^1B_1+2^0B_0.$$

Procesul comparării începe cu biţii cei mai semnificativi. Astfel, pentru a avea A<B este necesar ca:

sau $A_3 < B_3$,

sau $A_3 = B_3$ şi $A_2 < B_2$,

sau $A_3 = B_3$ şi $A_2 = B_2$ şi $A_1 < B_1$,

sau $A_3 = B_3$ şi $A_2 = B_2$ şi $A_1 = B_1$ şi $A_0 < B_0$.

Rezultă funcţia:

$$Fi = fi_3 +fe_3fi_2+fe_3fe_2fi_1+fe_3fe_2fe_1fi_0. \quad (4.16)$$

Pentru A = B ete necesar ca:

$A_3 = B_3$ şi $A_2 = B_2$ şi $A_1 = B_1$ şi $A_0 = B_0$.

Rezultă funcţia:

$$Fe = fe_3fe_2fe_1fe_0. \quad (4.17)$$

Pentru A > B este necesar ca:

sau $A_3 > B_3$,

sau $A_3 = B_3$ şi $A_2 > B_2$,

sau $A_3 = B_3$ şi $A_2 = B_2$ şi $A_1 > B_1$,

sau $A_3 = B_3$ şi $A_2 = B_2$ şi $A_1 = B_1$ şi $A_0 > B_0$.

Rezultă funcţia:

$$Fs = fs_3+fe_3fs_2+fe_3fe_2fs_1+fe_3fe_2fe_1fs_0. \quad (4.18)$$

Întrucât relaţiile 4.16, 4.17 şi 4.18 nu pot fi adevărate simultan, se poate scrie că oricare din cele 3 relaţii este adevărată dacă celelalte două sunt false:

$$Fi = \overline{Fe} \cdot \overline{Fs}; \quad (4.19)$$

$$Fe = \overline{Fi} \cdot \overline{Fs}; \quad (4.20)$$

$$Fs = \overline{Fi} \cdot \overline{Fe}. \quad (4.21)$$

Prin urmare, teoretic este suficientă obținerea a două din relațiile 4.16, 4.17 şi 4.18, a treia rezultând (cu numai două invesoare şi o poartă ŞI) dintr-una din relațiile 4.19, 4.20 sau 4.21. Practic, se implementează toate relațiile 4.16, 4.17 şi 4.18, pentru a nu apărea diferențe de timpi de propagare.

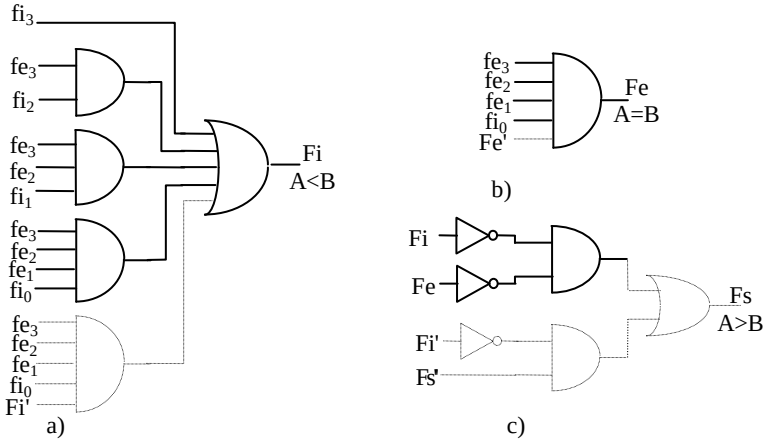



Fig. 4.18. Schemele logice simplificate ale funcțiilor de ieşire ale comparatorului de 4 biți

În fig. 4.18 este prezentată implementarea funcțiilor Fi, fig. 4.18 a, şi Fe, fig. 4.18 b, cu observația că circuitul corespunzător lui Fs poate fi realizat de maniera din fig. 4.18a (evident cu alte mărimi de intrare) sau de maniera din fig. 4.18 c (v. relația 4.21).

Fi', Fe' şi Fs' sunt intrări de extensie la care se conectează ieşirile comparatorului de 4 biți de rang inferior.

Varianta integrată a comparatorului numeric de 4 biți este circuitul integrat SN 7485, fig. 4.19.

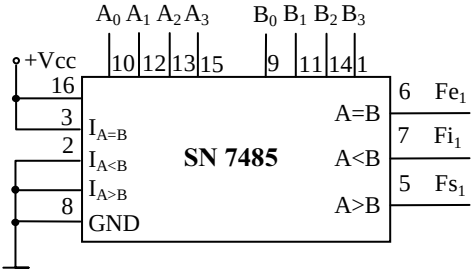



Fig. 4.19. Schema comparatorului integrat de 4 biți

### *4.5.3. Comparatorul numeric de 8 biţi*

Conectând în cascadă două comparatoare SN 7485, obţinem un comparator numeric de 8 biţi, fig. 4.20.

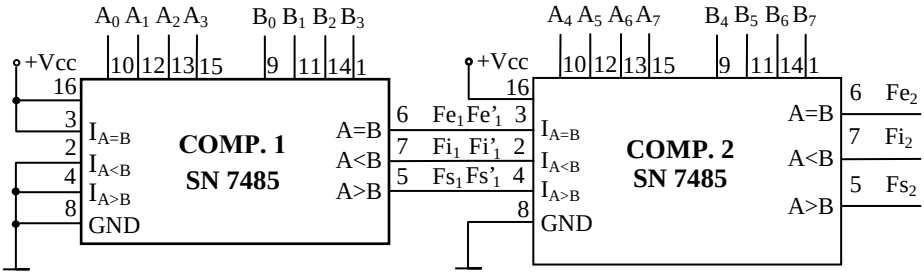



Fig. 4.20. Schema unui comparator de 8 biţi sintetizat cu 2 x SN 7485

În fig. 4.19 şi 4.20 putem observa modul în care sunt conectate intrările care provin de la rangul inferior al comparatorului numeric integrat SN 7485.

Astfel, intrarea corespunzătoare funcţiei de egalitate, A=B, se conectează la $+V_{CC}$ (1 logic), simulându-se astfel egalitatea biţilor de rang inferior care de fapt nu există (v. tab. 4.6).

Similar, intrările corespunzătoare funcţiilor de inferioritate (A<B) şi superioritate (A>B) sunt conectate la masă, simulând absenţa oricărei inegalităţi provenite de la rangul inferior.

## *4.6. Sumatoare*

Sumatoarele sunt subsisteme logice combinaţionale care asigură - direct sau indirect - efectuarea tuturor operaţiilor aritmetice dintr-un sistem de calcul.

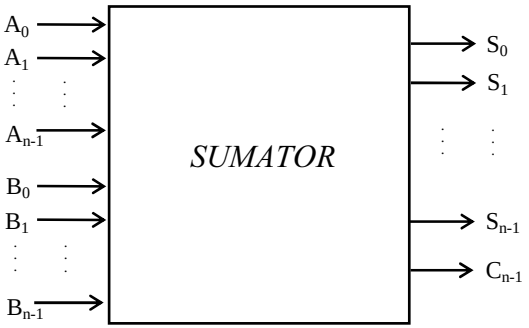



Fig. 4.21. Schema bloc generală a unui sumator

Schema bloc a unui sumator de 2 numere binare a câte n biți este prezentată în fig. 4.21, unde s-au notat cu $S_i$ , i=0,1, ..., n-1, biții corespunzători sumei, iar cu $C_i$ transportul către rangul următor.

### *4.6.1. Semisumatorul*

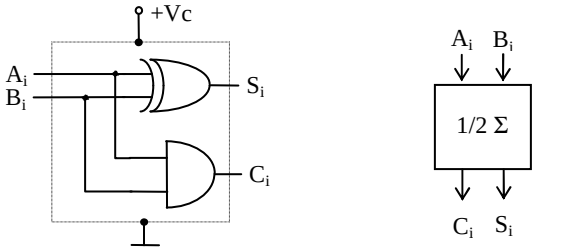
Semisumatorul realizează suma a două numere binare de câte 1 bit, fără a ține seama de transportul de la bitul imediat inferior ca semnificație.

Pornind de la tabelul de adevăr al unui semisumator de 1 bit, tab. 4.8, se obțin relațiile de calcul 4.22 şi 4.23 a căror implementare conduce la schema din fig. 4.22 a, sau, la nivel de schemă bloc, fig. 4.22 b.

Tab. 4.8. Tabelul de adevăr al semisumatorului de 1 bit

| $A_i$ | $B_i$ | *Rezultatul adunării* | *Suma* ($S_i$) | *Transport* ($C_i$) |
|---|---|---|---|---|
| 0 | 0 | 00 | 0 | 0 |
| 0 | 1 | 01 | 1 | 0 |
| 1 | 0 | 01 | 1 | 0 |
| 1 | 1 | 10 | 0 | 1 |

$$\begin{cases} S_i = A_i \oplus B_i; & (4.22) \\ C_i = A_i \cdot B_i. & (4.23) \end{cases}$$




a) schema logică b) schema bloc

Fig. 4.22. Semisumatorul de 1 bit

### *4.6.2. Sumatorul complet de 1 bit*

Spre deosebire de semisumator, sumatorul complet de 1 bit ia în considerație şi transportul $C_{i-1}$ de la bitul imediat inferior, conform schemei bloc din fig. 4.23.

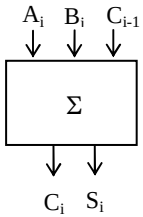



Fig. 4.23. Schema bloc a sumatorului complet de 1 bit

Tabelul de funcţionare al sumatorului complet de 1 bit este tab. 4.9.

Tab. 4.9. Tabelul de funcţionare al sumatorului complet de 1 bit

| ***Intrări*** | | | ***Suma*** | ***Ieşiri*** | |
|---|---|---|---|---|---|
| $A_i$ | $B_i$ | $C_{i-1}$ | | $S_i$ | $C_i$ |
| 0 | 0 | 0 | 00 | 0 | 0 |
| 0 | 0 | 1 | 01 | 1 | 0 |
| 0 | 1 | 0 | 01 | 1 | 0 |
| 0 | 1 | 1 | 10 | 0 | 1 |
| 1 | 0 | 0 | 01 | 1 | 0 |
| 1 | 0 | 1 | 10 | 0 | 1 |
| 1 | 1 | 0 | 10 | 0 | 1 |
| 1 | 1 | 1 | 11 | 1 | 1 |

Ca şi în cazul semisumatorului, ieşirea $S_i$ este suma modulo 2 a celor 3 intrări:

$$S_i = A_i \oplus B_i \oplus C_{i-1} = \tag{4.24}$$

$$= \overline{A}_i \overline{B}_i C_{i-1} + \overline{A}_i B_i \overline{C}_{i-1} + A_i \overline{B}_i \overline{C}_{i-1} + A_i B_i C_{i-1},$$

relaţie care se poate obţine şi direct din tab. 4.8, scriind $S_{iFCD}$.

Din acelaşi tabel se poate deduce şi $C_i$:

$$C_i = \overline{A}_i B_i C_{i-1} + A_i \overline{B}_i C_{i-1} + A_i B_i \overline{C}_{i-1} + A_i B_i C_{i-1}. \tag{4.25}$$

Grupând succesiv fiecare din primii trei termeni ai relaţiei (4.25) cu ultimul, se obţine:

$$C_i = B_i C_{i-1} + A_i C_{i-1} + A_i B_i, \tag{4.26}$$

iar după negarea relaţiei 4.26 şi aplicarea lui De Morgan, vom avea:

$$\overline{C}_i = \overline{A}_i \overline{B}_i + \overline{A}_i \overline{C}_{i-1} + \overline{B}_i \overline{C}_{i-1}. \tag{4.27}$$

Notând primii trei termeni din $S_i$ cu $D_i$:

$$D_i = \overline{A}_i \overline{B}_i C_{i-1} + \overline{A}_i B_i \overline{C}_{i-1} + A_i \overline{B}_i \overline{C}_{i-1}, \tag{4.28}$$

observăm că aceştia se pot obţine din produsul logic al lui $\overline{C}_i$ cu $(A_i + B_i + C_{i-1})$:

$$Di = (A_i + B_i + C_{i-1})\ \overline{C}_i\ . \qquad (4.29)$$

Într-adevăr, introducând $\overline{C}_i$ din relația 4.27 în 4.29 şi efectuând operațiile, se obține expresia 4.28.

Rezultă că $S_i$ se poate scrie:

$$S_i = D_i + A_i B_i C_{i-1} =$$

$$= A_i \overline{C}_i + B_i \overline{C}_i + C_{i-1}\overline{C}_i + A_i B_i C_{i-1}\ . \qquad (4.30)$$

Implementarea relațiilor 4.26 şi 4.30 conduce la sinteza schemei sumatorului complet de 1 bit, fig. 4.24.

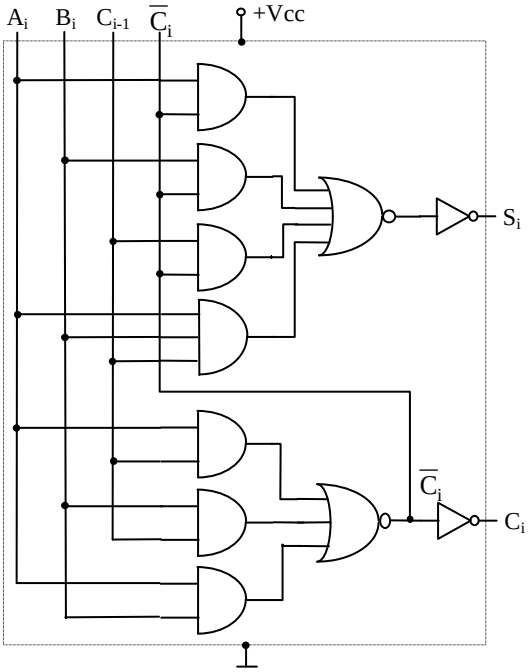



Fig. 4.24. Schema logică a sumatorului complet de 1 bit

### *4.6.3. Sumatorul complet de 4 biți*

Se obține prin interconectarea a 4 sumatoare complete de 1 bit, aşa cum este ilustrat în fig. 4.25. Întrucât implementarea unui astfel de sumator cu ajutorul circuitelor logice elementare este deosebit de laborioasă, vom utiliza pentru ilustrare sumatorul complet de 4 biți integrat CDB 483, a cărui schemă bloc este identică cu cea prezentată în fig. 4.25.

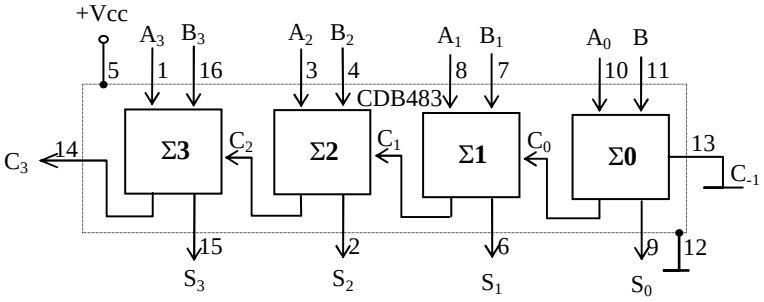



Fig. 4.25. Schema bloc a sumatorului complet de 4 biţi

## *4.7. Convertoare de cod*

Convertoarele de cod sunt circuite logice combinaţionale care permit transformarea unui cod binar în altul.

Schema bloc a unui convertor de n / m biţi este prezentată în fig. 4.26.


Fig. 4.26. Schema bloc generală a unui convertor de cod

### *4.7.1. Convertorul de cod "binar natural – Gray"*

Schema bloc a unui convertor pe 4 biţi din *cod binar natural* în *cod Gray* se obţine din fig. 4.26 pentru n = m = 4 şi este prezentată în fig. 4.27.


Fig. 4.27. Schema bloc a convertorului de cod "binar natural - Gray"

După cum rezultă şi din tabelul de adevăr, tab. 4.10, codul binar reflectat (Gray) se obţine din codul binar natural astfel:

$G_0$ - repetă primele 2 locaţii ale lui $B_0$, după care se reflectă din 2 în 2 locaţii;

$G_1$ - repetă primele 4 locații ale lui $B_1$, după care se reflectă din 4 în 4 locații;
$G_2$ - repetă primele 8 locații ale lui $B_2$, după care se reflectă din 8 în 8 locații;
$G_3$ - repetă $B_3$.

Tab. 4.10. Tabelul de adevăr al convertorului de cod "binar natural - Gray"

| ***Binar natural*** | | | | ***Gray*** | | | |
|---|---|---|---|---|---|---|---|
| $B_3$ | $B_2$ | $B_1$ | $B_0$ | $G_3$ | $G_2$ | $G_1$ | $G_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Pornind de la tab. 4.10, alcătuim diagramele VK pentru $G_3$, $G_2$, $G_1$ şi $G_0$, fig. 4.28.

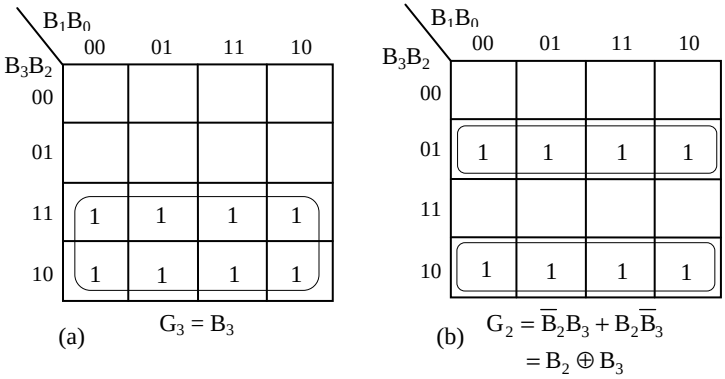



Fig. 4.28. Diagramele VK corespunzătoare funcțiilor de ieşire ale convertorului

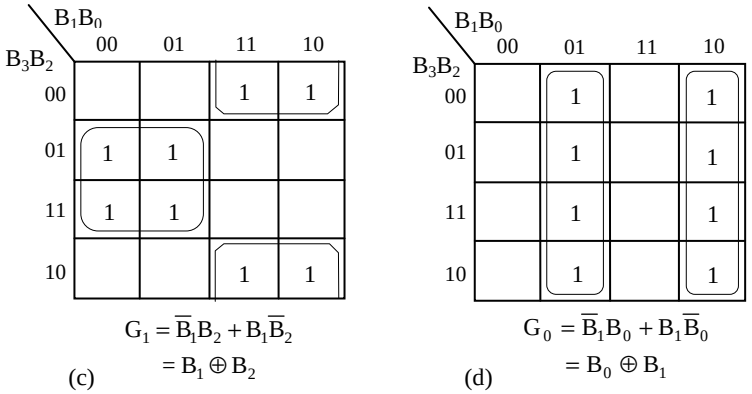



Fig. 4.28. Diagramele VK corespunzătoare funcţiilor de ieşire ale convertorului (continuare)

După minimizare, obţinem următoarele expresii:

$$G_3 = B_3; G_2 = B_2 \oplus B_3; G_1 = B_1 \oplus B_2; G_0 = B_0 \oplus B_1, \quad (4.31)$$

a căror implementare conduce la schema din fig. 4.29.

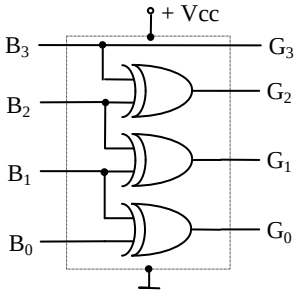



Fig. 4.29. Schema logică minimală a convertorului de cod "binar natural - Gray"

### *4.7.2. Convertorul de cod "Gray - binar natural"*

Schema bloc a unui convertor din *cod Gray* în *cod binar natural* este prezentată în fig. 4.30, iar tabelul de adevăr este tab. 4.11.


Fig. 4.30. Schema bloc a convertorului de cod "Gray - binar natural"

Tab. 4.11. Tabelul de adevăr al convertorului de cod "Gray - binar natural"

| ***Cod Gray*** | | | | ***Cod binar natural*** | | | |
|---|---|---|---|---|---|---|---|
| $G_3$ | $G_2$ | $G_1$ | $G_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

Întrucât aplicarea procedeului de la § 4.7.1 este destul de laborioasă, apelăm la următorul artficiu: cunoscut fiind faptul că $A \oplus A \oplus B = B$, calculăm cu ajutorul relațiilor 4.31 următoarele sume modulo 2:

$$G_2 \oplus G_3,\ G_1 \oplus G_2 \oplus G_3,\ G_0 \oplus G_1 \oplus G_2 \oplus G_3. \tag{4.32}$$

Obținem:

$$\begin{aligned}
&G_3 = B_3; && \Rightarrow B_3 = G_3;\\
&G_2 \oplus G_3 = B_2 \oplus \underbrace{B_3 \oplus B_3}_{0}; && \Rightarrow B_2 = G_2 \oplus G_3;\\
&G_1 \oplus G_2 \oplus G_3 = B_1 \oplus \underbrace{B_2 \oplus B_2}_{0} \oplus \underbrace{B_3 \oplus B_3}_{0}; && \Rightarrow B_1 = G_1 \oplus G_2 \oplus G_3;\\
&G_0 \oplus G_1 \oplus G_2 \oplus G_3 = B_0 \oplus \underbrace{B_1 \oplus B_1}_{0} \oplus \underbrace{B_2 \oplus B_2}_{0} \oplus \underbrace{B_3 \oplus B_3}_{0}; && \\
& && \Rightarrow B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3.
\end{aligned} \tag{4.33}$$

Implementarea relațiilor 4.33 conduce la schema logică a convertorului de cod "Gray - binar natural" din fig. 4.31.

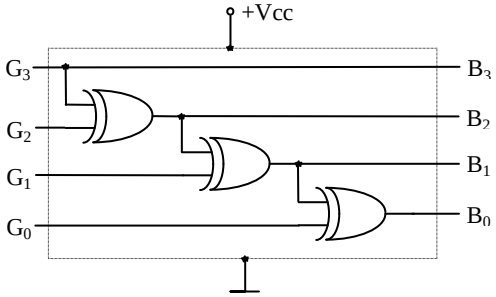



Fig. 4.31. Schema logică a convertorului de cod "Gray - binar natural"

## *4.8. Codificatoare*

Codificatoarele sunt circuite logice combinaţionale cu *n* intrări şi *m* ieşiri de adresă, constituind de fapt subsisteme ale unor circuite integrate pe scară medie (M.S.I.) sau largă (L.S.I.) cum ar fi: convertoarele de cod, circuitele ROM, PLA, etc.

Schema bloc a unui codificator este prezentată în fig. 4.32.

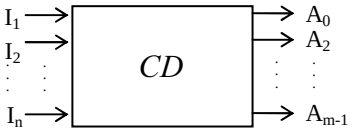



Fig. 4.32. Schema bloc generală a unui codificator

### *4.8.1. Codificatorul de adresă simplu*

Codificatorul de adresă simplu furnizează la ieşire un cuvânt binar de *m* biţi atunci când numai una din cele *n* intrări ale sale este activată.

Tab. 4.12. Tabelul de adevăr al codificatorului de adresă

| ***INTRĂRI*** | | | | | | | ***ADRESE*** | | |
|---|---|---|---|---|---|---|---|---|---|
| $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $A_2$ | $A_1$ | $A_0$ |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Rezultă că numărul cuvintelor furnizate la ieşire este $n=2^m-1$ şi este egal cu numărul intrărilor.

Pentru exemplificare, ne propunem să realizăm sinteza unui codificator de adresă cu n=7 intrări, deci cuvântul de adresă va fi format din m=3 biți.

Pornind de la tabelul de adevăr, tab. 4.12, se deduc expresiile funcțiilor de ieşire, rel. 4.34, 4.35 şi 4.36, şi se obține varianta de implementare din fig. 4.33:

$$A_0 = I_1 + I_3 + I_5 + I_7; \quad (4.34)$$

$$A_1 = I_2 + I_3 + I_6 + I_7; \quad (4.35)$$

$$A_2 = I_4 + I_5 + I_6 + I_7. \quad (4.36)$$

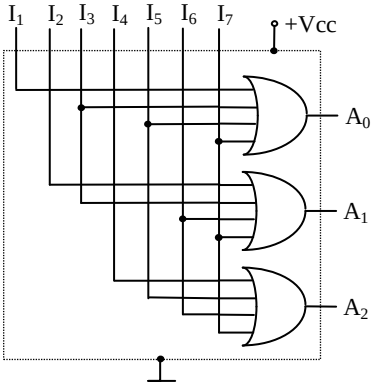



Fig. 4.33. Schema logică a codificatorului de adresă

Observație: este interzisă activarea simultană a mai multor linii de intrare deoarece se pot crea confuzii. De exemplu, activarea simultană a liniilor $I_1$ şi $I_2$ generează cuvântul de cod $A_2$=0, $A_1$=1, $A_0$=1 (011) care corespunde de fapt, într-o funcționare normală, activării lui $I_3$. În cazul în care nu se poate evita activarea simultană a mai multor intrări, se folosesc circuite de codificare (codare) prioritare.

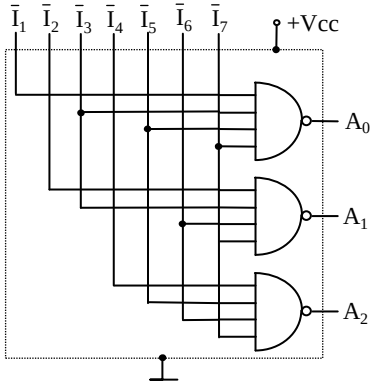



Fig. 4.34. O altă variantă de implementare a codificatorului de adresă

O altă variantă de implementare a CD cu 7 intrări şi 3 ieşiri de adresă se poate obţine aplicând relaţiilor 4.34, 4.35 şi 4.36 principiul dublei negaţii şi una din relaţiile lui De Morgan:

$$A_0 = \overline{\overline{I_1 + I_3 + I_5 + I_7}} = \overline{\bar{I}_1 \bar{I}_3 \bar{I}_5 \bar{I}_7} \tag{4.37}$$

$$A_1 = \overline{\overline{I_2 + I_3 + I_6 + I_7}} = \overline{\bar{I}_2 \bar{I}_3 \bar{I}_6 \bar{I}_7} \tag{4.38}$$

$$A_2 = \overline{\overline{I_4 + I_5 + I_6 + I_7}} = \overline{\bar{I}_4 \bar{I}_5 \bar{I}_6 \bar{I}_7} \tag{4.39}$$

Se obţine schema prezentată în fig. 4.34.

## *4.9. Decodificatoare*

Decodificatoarele sunt circuite logice combinaţionale cu n intrări şi m ieşiri, realizate în tehnologie MSI, care activează una sau mai multe ieşiri în funcţie de cuvântul de cod aplicat la intrare ($m=2^n$).

Schema bloc a unui decodificator este prezentată în fig. 4.35.

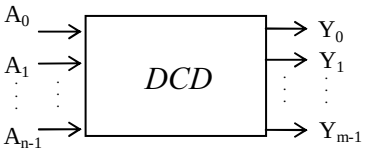



Fig. 4.35. Schema bloc generală a unui decodificator

### *4.9.1. Decodificatorul de adresă*

Decodificatorul de adresă activează linia de ieşire a cărei adresă codificată binar este aplicată la intrări.

Schema bloc a unui decodificator de adresă cu n=2 intrări şi $m=2^2=4$ ieşiri este prezentată în fig. 4.36.

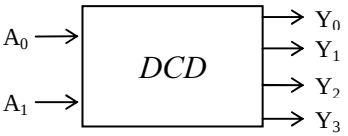



Fig. 4.36. Schema bloc a unui decodificator cu 2 intrări şi 4 ieşiri

Din tabelul de adevăr, tab. 4.13, se obţin expresiile 4.40 ale funcţiilor de ieşire şi varianta de implementare din fig. 4.37.

Tab. 4.13. Tabelul de adevăr al decodificatorului cu 2 intrări şi 4 ieşiri

| $A_1$ | $A_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

$$Y_0 = \overline{A}_1\overline{A}_0 \; ; \; Y_1 = \overline{A}_1 A_0 \; ; \; Y_2 = A_1\overline{A}_0 \; ; \; Y_3 = A_1 A_0 \qquad (4.40)$$

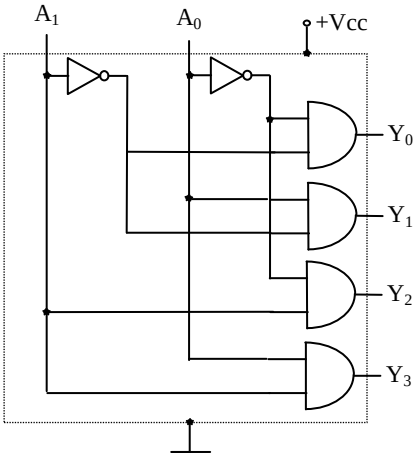



Fig. 4.37. Schema logică a decodificatorului cu 2 intrări şi 4 ieşiri

### *4.9.2. Decodificatorul BCD-zecimal*

Prescurtarea BCD semnifică în limba română "zecimal codat binar".
Schema bloc a unui decodificator BCD-zecimal este prezentată în fig. 4.38.

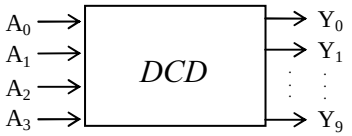



Fig. 4.38. Schema bloc a decodificatorului BCD - zecimal

Spre deosebire de codul binar natural, BCD nu include combinaţiile binare 1010, 1011, 1100, 1101, 1110, 1111, combinaţii ce corespund numerelor zecimale 10, 11, 12, 13, 14 şi 15.

Apariţia oricăreia din cele 6 combinaţii de intrare excluse, duce toate ieşirile în starea "1". Se spune că decodificatorul rejectează datele false.

Funcţionarea decodificatorului din fig. 4.38 (în variantă integrată - CDB 442) este prezentată în tab. 4.14.

Tab. 4.14. Tabelul de adevăr al decodificatorului BCD - zecimal

| | $A_3$ | $A_2$ | $A_1$ | $A_0$ | $\overline{Y}_0$ | $\overline{Y}_1$ | $\overline{Y}_2$ | $\overline{Y}_3$ | $\overline{Y}_4$ | $\overline{Y}_5$ | $\overline{Y}_6$ | $\overline{Y}_7$ | $\overline{Y}_8$ | $\overline{Y}_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 13 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### *4.9.3. Decodificatorul BCD - 7 segmente*

Decodificatorul BCD - 7 segmente prezintă schema bloc din fig. 4.39,

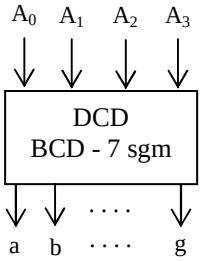



Fig. 4.39. Schema bloc a unui decodificator BCD - 7 segmente

acceptă un cod de intrare BCD şi produce ieşirile adecvate pentru selectarea segmentelor unui digit cu 7 segmente utilizat pentru reprezentarea numerelor zecimale 0, 1, .., 9.

Dacă cele 7 ieşiri ale decodificatorului sunt active în stare "sus", ele se notează cu a, b, …, g şi vor comanda un display cu 7 segmente, fig. 4.40 a, în care LED-urile se află în conexiune *catod comun* (KC), fig. 4.40 b.

Dacă ieşirile decodificatorului sunt active în stare "jos", ele se notează cu $\overline{a},\overline{b},...,\overline{g}$ şi vor comanda un digit ale cărui LED-uri se află în conexiune *anod comun* (AC), fig. 4.40 c.

Este uşor de înţeles faptul că, în condiţiile în care LED-urile au catozii legaţi împreună (KC) şi conectaţi la masă, singurul potenţial care, aplicându-se pe anozi, poate deschide LED-urile, este $+V_{CC}$, deci 1 logic.
Un raţionament similar poate fi făcut pentru conexiunea AC.

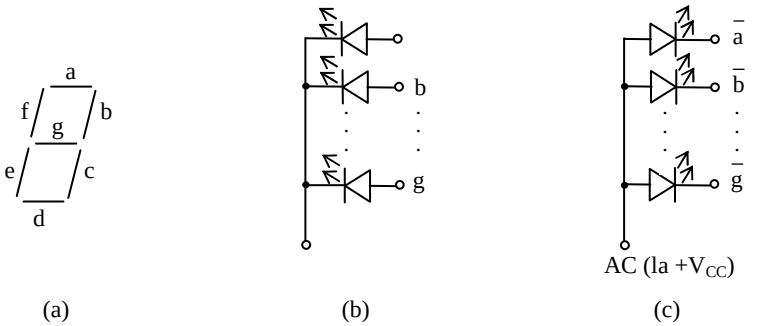



Fig. 4.40. Display-ul cu 7 segmente
a) notarea segmentelor; b) schema electrică pentru KC; c) schema electrică pentru AC.

### *4.9.3.1. Decodificatorul BCD - 7 segmente cu componente discrete*

Ca şi în cazul celorlalte circuite logice combinaţionale studiate până în prezent, ne propunem să realizăm sinteza unui decodificator BCD - 7 segmente cu componente discrete.

În acest scop, alcătuim tabelul de adevăr al decodificatorului, tab. 4.15, trecând în prima coloană numerele zecimale de la 0 la 15, în coloanele 2 … 5 – combinaţiile logice de intrare corespunzătoare numerelor zecimale din prima coloană (cod binar natural), iar în următoarele 7 coloane – ieşirile *a*, *b*, …, *g*, active în 1 logic.

Se completează, linie cu linie, cele 7 coloane corespunzătoare funcţiilor de ieşire, astfel încât segmentele activate să formeze cifra înscrisă în prima coloană a tab. 4.15, conform corespondenţei din fig. 4.41.

Tab. 4.15. Tabelul de adevăr al decodificatorului BCD – 7 segmente

| | $A_3$ | $A_2$ | $A_1$ | $A_0$ | *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 |
| 10 | 1 | 0 | 1 | 0 | x | x | x | x | x | x | x |
| 11 | 1 | 0 | 1 | 1 | x | x | x | x | x | x | x |
| 12 | 1 | 1 | 0 | 0 | x | x | x | x | x | x | x |
| 13 | 1 | 1 | 0 | 1 | x | x | x | x | x | x | x |
| 14 | 1 | 1 | 1 | 0 | x | x | x | x | x | x | x |
| 15 | 1 | 1 | 1 | 1 | x | x | x | x | x | x | x |

De exemplu, combinaţiei binare 0000 îi corespunde în zecimal cifra 0 a cărei vizualizare presupune aprinderea LED-urilor *a, b, c, d, e* şi *f,* deci activarea prin 1 logic a liniilor de ieşire corespunzătoare ale decodificatorului. Prin urmare, se completează prima linie a tabelului 4.15 cu 1 logic, exceptând locaţia corespunzătoare ieşirii *g*, care rămâne în 0 logic.

Se procedează similar pentru toate combinaţiile binare corespunzătoare numerelor de la 0 la 9.

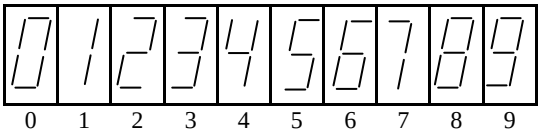



Fig. 4.41. Vizualizarea cifrelor zecimale pe un display cu 7 segmente

Pentru combinaţiile binare care corespund numerelor de la 10 la 15, interzise în BCD, starea ieşirilor decodificatorului este “indiferentă”, situaţie pe care o marcăm prin “x” în tab. 4.15.

Observăm că funcţiile de ieşire *a*, *b*, …, *g*, corespunzătoare celor 7 segmente, sunt incomplet definite, v. § 1.6.3, fapt de care va trebui să ţinem seama în procesul de minimizare.

Se completează diagramele Veitch-Karnaugh ale celor 7 funcţii de ieşire, fig. 4.42, şi se alege minimizarea de tip conjunctiv, deoarece din analiza diagramelor se constată că locaţiile care conţin 0 logic sunt mai puţine.

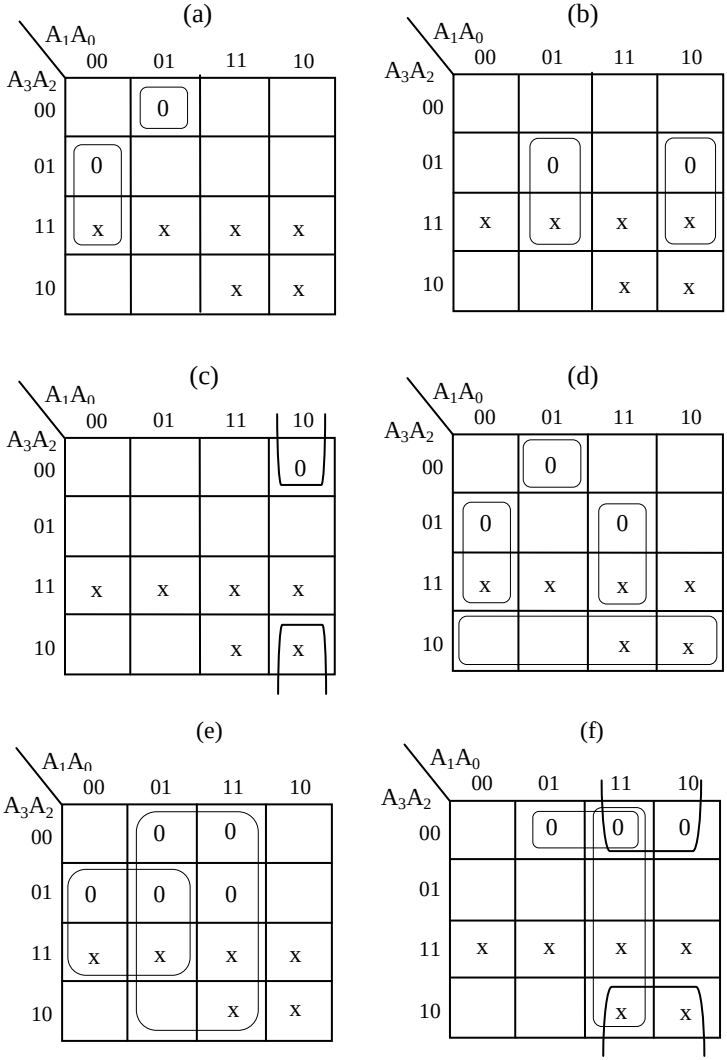



Fig. 4.42. Diagramele VK corespunzătoare celor 7 segmente

(g)

| $A_3A_2$ \ $A_1A_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | | |
| 01 | | | 0 | |
| 11 | x | x | x | x |
| 10 | | | x | x |

Fig. 4.42. Diagramele VK corespunzătoare celor 7 segmente (continuare)

Observație: Locațiile libere din diagramele VK sunt cele în care în mod normal ar fi trebuit înscrisă valoarea logică 1. Din motive de simplitate a desenului şi uşurință a grupărilor, locațiile respective au fost lăsate libere.

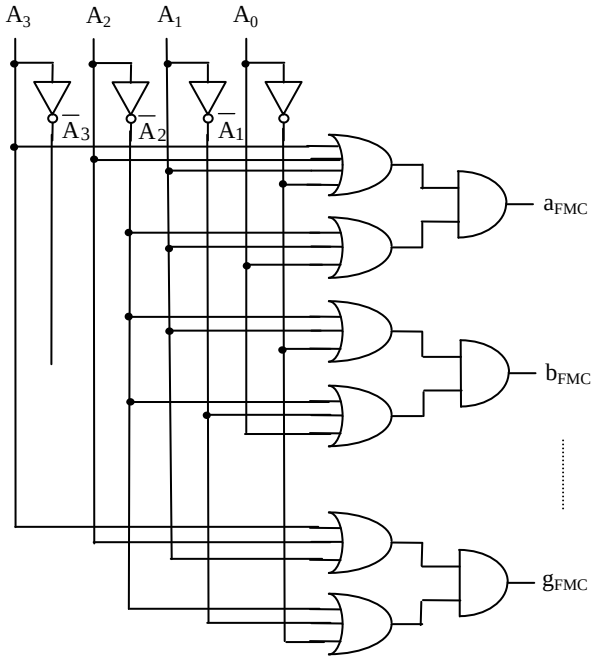



Fig. 4.43. Schema sintetizată a decodificatorului BCD – 7 segmente

Expresiile formelor minimale conjunctive sunt:

$$\begin{aligned} a_{FMC} &= (A_3 + A_2 + A_1 + \overline{A}0)\cdot(\overline{A}2 + A_1 + A_0), \\ b_{FMC} &= (\overline{A}2 + A_1 + \overline{A}0)\cdot(\overline{A}2 + \overline{A}1 + A_0), \\ &\vdots \\ g_{FMC} &= (A_3 + A_2 + A_1)\cdot(\overline{A}2 + \overline{A}1 + \overline{A}0), \end{aligned} \tag{4.41}$$

iar implementarea lor conduce la schema decodificatorului BCD – 7 segmente din fig. 4.43.

### *4.9.3.2. Decodificatorul BCD - 7 segmente în variantă integrată*

O variantă a decodificatorului BCD – 7 segmente o constituie circuitul integrat CDB 447, ale cărui ieşiri sunt active în 0 logic, v.tab. 4.16, impunându-se din acest motiv utilizarea unui display cu 7 segmente cu anod comun, fig. 4.40 c.

Tab. 4.16. Tabelul de funcţionare al decodificatorului BCD - 7 segmente integrat (CDB 447)

| *ZECIMAL SAU FUNCŢIA* | *INTRĂRI* | | | | | | | *IEŞIRI* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\overline{LT}$ | $\overline{RBI}$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | $\overline{BI}$ / $\overline{RBO}$(b) | $\overline{a}$ | $\overline{b}$ | $\overline{c}$ | $\overline{d}$ | $\overline{e}$ | $\overline{f}$ | $\overline{g}$ |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | x | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 1 | x | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | x | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 1 | x | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 1 | x | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 1 | x | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | x | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | x | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | x | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 10 | 1 | x | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 11 | 1 | x | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 12 | 1 | x | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 13 | 1 | x | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 14 | 1 | x | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 15 | 1 | x | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $\overline{BI}$(b) | x | x | x | x | x | x | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $\overline{RBI}$(b) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $\overline{LT}$(b) | 0 | x | x | x | x | x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Nota (b):

$\overline{RI}$ / $\overline{RBO}$ (Blanking Input / Ripple Blanking Output);

$\overline{BI}$ - în "aer" sau la "1" dacă dorim funcțiile de ieşire 0÷15;

$\overline{RBI}$ - în "aer" sau la "1" dacă afişarea lui 0 nu este dorită;

$\overline{LT}$ (Lamp Test Input).

Din fig. 4.44 se observă că segmentele activate pentru obținerea cifrelor 6 şi 9 realizează o vizualizate mai puțin agreabilă a acestora, iar cele corespunzătoare combinațiilor logice de intrare interzise în BCD (ce corespund numerelor zecimale 10, 11, ..., 15), nu au practic nici o semnificație.


Fig. 4.44. Formarea cifrelor zecimale cu ajutorul celor 7 segmente

## *4.10. Memorii ROM*

Memoria ROM (Read Only Memory = memorie numai cu citire) este o memorie fixă în sensul că odată înscrisă informația în ea, aceasta nu mai poate fi ştearsă sau modificată, ci numai citită.

Memoria ROM poate fi privită ca un convertor de cod format dintr-un decodificator de adresă şi un codificator, fig. 4.45.

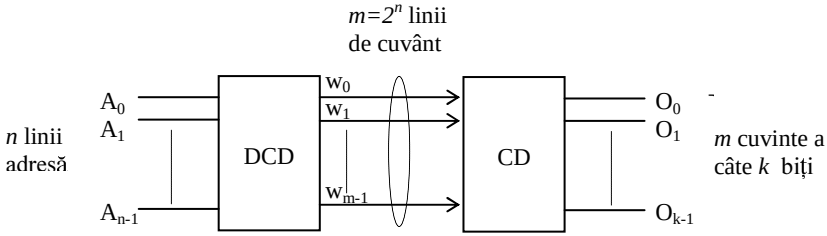



Fig. 4.45. Schema bloc a memoriei ROM

Decodificatorul are la intrare un vector de adresă format din *n* variabile (*n* linii de adresă) ale căror combinații logice activează succesiv cele $m=2^n$ linii de ieşire.

Codificatorul are la intrare cele *m* linii (de cuvânt) activate succesiv, fiecare linie $w_p$, cu p=0, 1, …, m-1, fiind capabilă prin activare să citească şi să transmită la ieşirile $O_0$, $O_1$, …, $O_{k-1}$, câte un cuvânt format din *k* biți.

Intuitiv, codificatorul ar putea fi imaginat sub forma unui dulap cu *m* sertare, fig. 4.46, în fiecare sertar aflându-se câte *k* bile albe şi negre, simbolizând valorile logice 1, respectiv 0. După ce au fost umplute cu bile, sertarele sunt încuiate şi cheia este aruncată, astfel încât configuraţia alb-negru a bilelor din sertare rămâne definitivă.

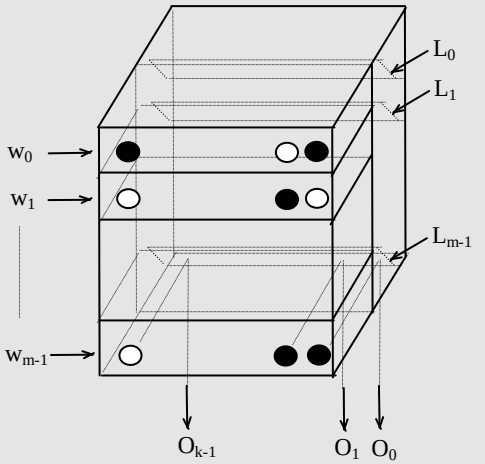



Fig. 4.46. O prezentare intuitivă a codificatorului memoriei ROM

Activarea uneia dintre liniile de intrare $w_p$, va face ca în sertarul corespunzător să se aprindă un bec, astfel încât, prin intermediul oglinzilor $L_0$, $L_1$, … $L_{m-1}$, combinaţia alb-negru (deci 1 şi 0 logic) a bilelor din sertarul respectiv va putea fi citită la ieşirile $O_0$, $O_1$, …, $O_{k-1}$. Operaţia de citire a conţinutului oricărui sertar poate fi repetată la infinit, fără a afecta în acest mod conţinutul sertarului.

Capacitatea C a unei memorii ROM este determinată de numărul de biţi ai matricei de memorare, care pentru *m* linii de cuvânt a câte *k* biţi fiecare, este:

$$C=m\cdot k=2^n\cdot k. \qquad (4.42)$$

Datele furnizate la ieşirea codificatorului, sub forma a *m* cuvinte a câte *k* biţi fiecare, reprezintă informaţia înmagazinată în codificator.

În funcţie de locul unde se realizează înscrierea informaţiei în codificator, distingem memorii ROM programabile la producător, respectiv - la utilizator.

După tipul tehnologiei de fabricaţie utilizate, memoriile ROM pot fi realizate în tehnologie integrată bipolară sau unipolară.

Evident, structura codificatorului diferă de la un tip de memorie ROM la altul.

### *4.10.1. Memorii ROM bipolare*

Memoriile ROM bipolare se caracterizează prin timpi reduşi de acces la informaţia memorată (de ordinul zecilor de nanosecunde).

#### *4.10.1.1. Memorii ROM bipolare programabile la producător*

În fig. 4.47 prezentăm a *m*-a parte din structura codificatorului unei memorii ROM bipolare programabile la producător, şi anume acea parte care corespunde unei linii de cuvânt oarecare, $w_p$.

Schema conţine *k* repetoare pe emiter realizate cu tranzistoarele $T_0$, $T_1$, …, $T_{k-1}$.

Procesul de fabricaţie al circuitului integrat care înglobează memoria ROM este oprit înainte de realizarea legăturilor $l_0$, $l_1$, …, $l_{k-1}$, dintre bazele tranzistoarelor şi linia $w_p$, şi nu este reluat, din motive de rentabilitate, decât în momentul în care s-au primit suficiente comenzi pentru o anumită configuraţie de 0 şi 1 logic a matricei de memorare.

Zonele $l_i$ corespunzătoare locaţiilor în care se doreşte înscrierea informaţiei 1 logic vor fi metalizate, iar cele ce corespund locaţiilor care trebuie să conţină 0 logic vor rămâne nemetalizate.

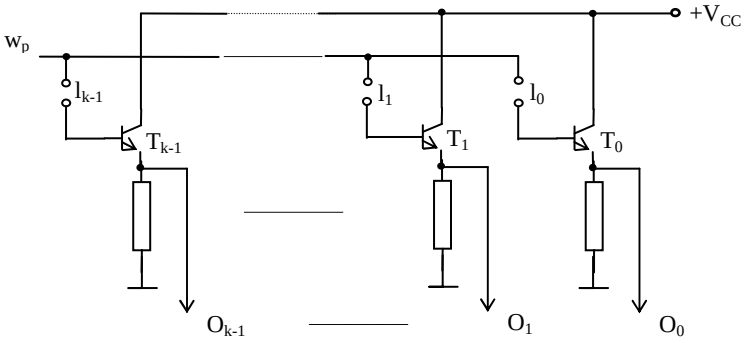



Fig. 4.47. A *m*-a parte din codificatorul memoriei ROM bipolare programabile la producător

De exemplu, dacă la ieşirea $O_0$ dorim să citim 1 logic, zona $l_0$ va fi metalizată, astfel încât activarea liniei $w_p$ va însemna aplicarea unui potenţial apropiat de $+V_{CC}$, corespunzător lui 1 logic, pe baza tranzistorului $T_0$, saturarea acestuia şi obţinerea în emiterul său a potenţialului:

$$V_{O_0} = V_{CC} - V_{CE0sat} = V_{CC} - 0{,}1V \cong V_{CC} \ , \qquad (4.43)$$

deci 1 logic.

Dacă la aceeaşi ieşire $O_0$ dorim să obţinem 0 logic, legătura $l_0$ va rămâne nemetalizată, astfel încât, indiferent de potenţialul sau valoarea logică a liniei $w_p$, tranzistoarul $T_0$ va rămâne permanent blocat şi potenţialul masei se va transfera la ieşire prin rezistenţa din emiter. Rezultă $V_{O_0} = 0V$, deci 0 logic.

Programarea memoriei ROM este, prin urmare, o etapă a procesului de fabricaţie, legăturile dintre bazele tranzistoarelor şi liniile $w_p$ fiind realizate prin metalizare, după aplicarea pe chip-ul semiconductor a unei măşti care lasă libere numai acele zone $l_i$ care urmează a fi metalizate. Se spune că această memorie ROM este *programabilă prin mască.*

### *4.10.1.2. Memorii ROM bipolare programabile la utilizator*

Acest tip de memorie este cunoscut sub denumirea de PROM (Programmable ROM).

În fig. 4.48 prezentăm acea parte a codificatorului care corespunde liniei $w_p$.

Elementele de memorie sunt pelicule fuzibile subţiri de crom-nichel ($f_0$, $f_1$, …, $f_{k-1}$) care pot fi arse prin trecerea unui curent de programare $I_p$, având o intensitate de ordinul zecilor sau sutelor de miliamperi şi o durată de câteva zeci de milisecunde. Acest curent ia naştere prin aplicarea unei anumite diferenţe de potenţial între ieşirea corespunzătoare locaţiei de memorie respective şi masă, v. fig. 4.48, în timp ce linia $w_p$ este activată.

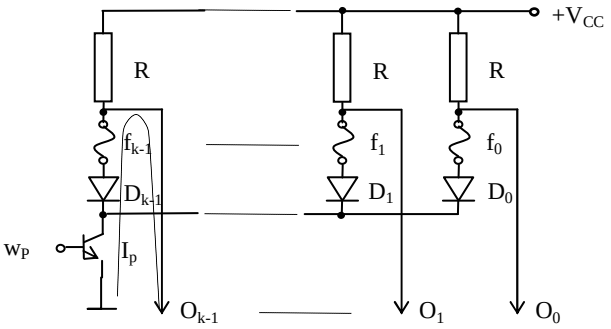



Fig. 4.48. A *m*-a parte din codificatorul memoriei ROM bipolare programabile la utilizator

În timpul operaţiei de citire a memoriei ROM, $w_p$ se activează şi la ieşirile ce corespund fuzibilelor arse vom avea 1 logic (potenţialul $+V_{CC}$ transferat prin rezistenţele R corespunzătoare), în timp ce la celelalte ieşiri vom avea 0 logic ($V_{Oi}=V_{CEsat}+V_{DiON}=0,1+0,7=0,8V\approx 0V$).

Evident, reprogramarea unei astfel de memorii este imposibilă.

### *4.10.2. Memorii ROM unipolare*

Memoriile ROM unipolare au capacităţi mari, dar timpi de acces mai slabi decât ai memoriilor bipolare (sute de nanosecunde).

#### *4.10.2.1. Memorii ROM unipolare programabile la producător*

Ca şi în cazurile anterioare, prezentăm numai o parte din structura codificatorului, fig. 4.49. Este vorba despre *k* inversoare NMOS statice, ale căror drivere au grilele conectate la linia de cuvânt $w_p$.

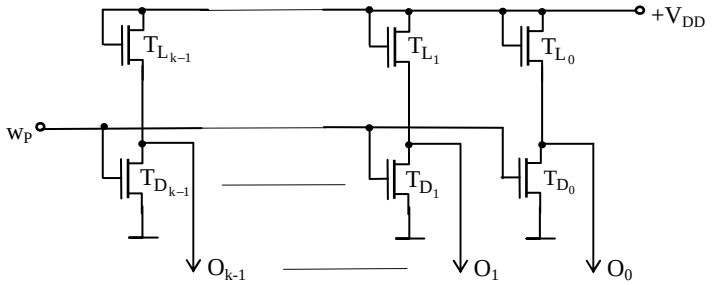



Fig. 4.49. A *m*-a parte din codificatorul memoriei ROM unipolare programabile la producător

Programarea la producător se face prin mască, pe baza unei *hărţi logice* furnizate de către utilizator şi constă în realizarea unui strat izolator al porţii mai gros la tranzistoarele ce urmează a fi dezactivate. Acestea vor rămâne blocate indiferent de nivelul logic al lui $w_p$, la ieşirile corespunzătoare transferându-se potenţialul $+V_{DD}$ prin tranzistorul sarcină respectiv, deci 1 logic.

Ieşirile corespunzătoare celorlalte tranzistoare vor furniza 0 logic în momentul activării liniei $w_p$, potenţialul masei fiind transferat la ieşire prin tranzistorul driver respectiv (v. funcţionarea inversorului NMOS static, § 3.2.2.1.1).

#### *4.10.2.2. Memorii ROM unipolare programabile la utilizator*

Acest tip de memorii ROM se împarte în două categorii şi anume:

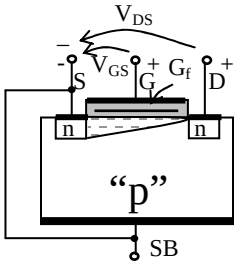
- EPROM (Erasable PROM = PROM cu posibilitate de ştergere);
- $E^2$PROM (Electrically Erasable PROM = PROM cu posibilitate de ştergere pe cale electrică), sau EAROM (Electrically Alterable ROM = ROM cu posibilitate de modificare pe cale electrică).

*4.10.2.2.1. Memorii EPROM*

Memoriile EPROM prezintă o structură a codificatorului identică cu cea prezentată în fig. 4.49.

Deosebirea față de memoriile ROM unipolare programabile la producător, constă în construcția specială a tranzistoarelor driver. Acestea sunt prevăzute cu o grilă flotantă, neconectată la circuitul exterior şi plasată în interiorul stratului de oxid ce separă grila principală de canalul virtual.

În fig. 4.50 am prezentat structura fizică şi simbolizarea unui astfel de tranzistor.




a) Structura fizică b) Simbolul

Fig. 4.50. Tranzistorul driver cu grilă flotantă

*Înscrierea* unui 1 logic într-o locație a memoriei se va face dezactivând tranzistorul driver respectiv prin încărcarea porții flotante cu sarcini electrice negative. Acestea vor respinge electronii din substratul de bază şi vor face imposibilă inducerea canalului, indiferent de potențialul aplicat pe grila de lucru.

Injectarea sarcinilor electrice negative pe grila izolată se face prin aplicarea unei tensiuni drenă-sursă mari, cu + pe drenă, şi a unei tensiuni grilă-sursă pozitive, cu + pe grilă. În aceste condiții are loc penetrarea canalului, deci formarea în canal, foarte aproape de zona drenei, a unei regiuni libere de sarcini electrice care se comportă ca un izolator (v. cursul de D.C.E.). Între extremitățile acestei regiuni, în lungul canalului, se aplică practic întreaga tensiune $V_{DS}$, luând naştere un câmp electric intens care accelerează electronii din canal către drenă. O parte dintre aceşti electroni, primesc energie suficientă pentru a străpunge stratul de oxid şi a ajunge pe grila flotantă, formând un nor de sarcini electrice negative care dezactivează tranzistorul respectiv, înscriind practic un 1 logic la locația corespunzătoare a memoriei EPROM.

*Ştergerea* informațiilor din locațiile memoriei EPROM se realizează prin expunerea la radiații ultraviolete a grilelor tranzistoarelor driver, situate în dreptul unei ferestre din cuarț de pe suprafața circuitului integrat. Electronii de pe grilele flotante primesc de la radiația ultravioletă energia necesară pentru a străpunge în sens invers peliculele de oxid care despart grilele flotante de substratul de bază, revenind astfel în substrat şi reactivând tranzistoarele driver.

Memoria EPROM este acum gata pentru o nouă înscriere.

*4.10.2.2.2. Memoria $E^2PROM$*

Memoria $E^2PROM$, EEPROM sau EAROM elimină inconvenientul pe care-l reprezintă, în cazul memoriei EPROM, duratele mari de expunere la radiații ultraviolete în timpul procesului de ştergere.

Memoria $E^2PROM$ realizează o ştergere relativ rapidă a informațiilor stocate, combinând o modificare a structurii fizice a tranzistorului driver cu utilizarea unui procedeu electric simplu de ştergere.

Astfel, stratul de oxid care separă grila flotantă de substratul de bază este mult mai subțire către zona drenei, atingând valori de ordinul 0,01μm.

*Înscrierea* se face prin aplicarea, transversal față de pelicula de oxid, între drenă şi grila de lucru, a unei tensiuni de cca. 10V care determină străpungerea oxidului, formarea norului electronic care dezactivează tranzistorul şi, implicit, înscrierea unui 1 logic la locația respectivă a memoriei.

*Ştergerea* se realizează prin inversarea polarității tensiunii necesare înscrierii unui 1 logic.

## *4.10.3. Organizarea unei memorii ROM de 8Kb*

Pornind de la schema bloc a memoriei ROM din fig. 4.45 şi luând: n=10 linii de adresă, $m=2^{10}=1024$ linii de cuvânt, şi o lungime a cuvântului de cod k=8 biți, obținem o capacitate a memoriei: C=m·k=1024·8=8Kbiți (1024 biți=1Kilobit=1Kb).

În acest exemplu, decodificatorul ar trebui să prezinte 1024 linii de ieşire, fiecare dintre acestea selectând câte un cuvânt de cod de 8 biți.

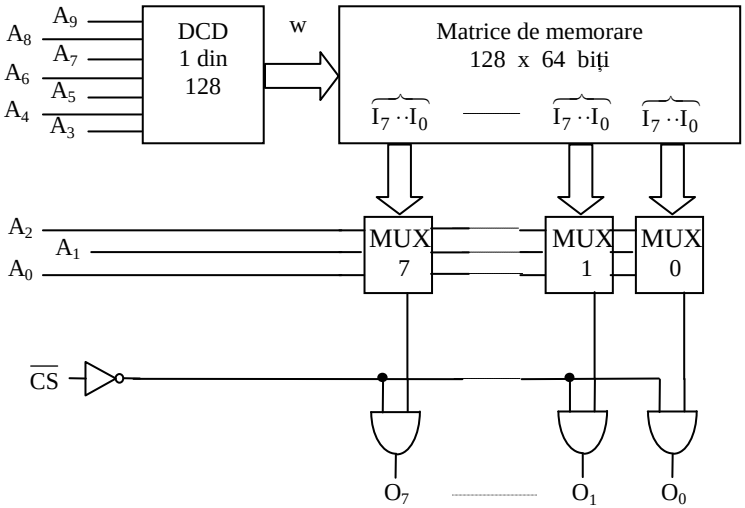



Fig. 4.51. Structura unei memorii ROM de 8Kb

O organizare mai judicioasă din punctul de vedere al numărului de porți utilizate, este cea prezentată în fig. 4.51.

În această schemă, decodificatorul prezintă numai n=7 linii de adresă ($A_3$, $A_4$, …, $A_9$) care activează m=$2^7$=128 linii de cuvânt, fiecare dintre acestea selectând câte un cuvânt de cod format din 64 biți, grupați 8 câte 8 la intrările a 8 multiplexoare.

Primele 3 linii de adresă, şi anume cele corespunzătoare celor mai puțin semnificativi 3 biți ($A_2$, $A_1$, $A_0$), sunt destinate selecției succesive a câte 8 din cei 64 biți de la intrarea MUX-urilor şi dirijării acestora către ieşirile $O_0$, $O_1$, …, $O_7$, v. tab. 4.17.

Tab. 4.17. Explicativ pentru funcționarea memoriei ROM de 8 Kb

| ***Linia selectată*** | $2^9$ | $2^8$ | . . . | $2^3$ | $2^2$ | $2^1$ | $2^0$ | ***Intrările MUX-urilor care au acces la ieşiri*** |
|---|---|---|---|---|---|---|---|---|
| | ***$A_9$*** | ***$A_8$*** | **. . .** | ***$A_3$*** | ***$A_2$*** | ***$A_1$*** | ***$A_0$*** | |
| $w_0$ | 0 | 0 | . . . | 0 | 0 | 0 | 0 | $I_0$ |
| | 0 | 0 | . . . | 0 | 0 | 0 | 1 | $I_1$ |
| | ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | 0 | 0 | . . . | 0 | 1 | 1 | 1 | $I_7$ |
| $w_1$ | 0 | 0 | . . . | 1 | 0 | 0 | 0 | $I_0$ |
| | 0 | 0 | . . . | 1 | 0 | 0 | 1 | $I_1$ |
| | ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | 0 | 0 | . . . | 1 | 1 | 1 | 1 | $I_7$ |
| ⋮ | ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $w_{127}$ | 1 | 1 | . . . | 1 | 1 | 1 | 1 | $I_7$ |

Astfel, pentru combinația binară $A_9A_8$ … $A_3A_2A_1A_0$=00 … 0000, biții de adresă $A_9A_8$ … $A_3$=00 … 0 vor activa linia de ieşire $w_0$ a decodificatorului, care va selecta la rândul ei un prim cuvânt de cod de 64 biți, transmițându-l la cele 8x8 intrări ale MUX-urilor.

Liniile de adresă $A_2A_1A_0$=000, v.tab. 4.17, vor permite celor 8 intrări $I_0$ să acceadă la ieşirile MUX-urilor şi, presupunând că bara de selecție $\overline{CS}$ ($\overline{\text{Chip Select}}$) = 0 , primul cuvânt de 8 biți, $O_7$ … $O_1O_0$, va avea acces la ieşirile memoriei ROM.

Următoarea combinație de adresă $A_9A_8$ … $A_3A_2A_1A_0$=00 … 0001 va păstra linia $w_0$ activă ($A_9A_8$ … $A_3$=00 … 0) şi va permite accesul către ieşiri al următoarelor 8 intrări ($I_1$) ale MUX-urilor ($A_2A_1A_0$=001). Cel de-al doilea cuvânt de cod de 8 biți a fost citit la ieşirea memoriei ROM.

Procesul continuă până când ultimii 8 biți din cei 64 de pe linia $w_0$ sunt citiți la ieşire.

Urmează combinația logică $A_9A_8$ … $A_3A_2A_1A_0$=00 … 1000, care va activa linia de cuvânt $w_1$, selectând astfel un nou set de 64 biți care vor ajunge la ieşirea memoriei ROM sub forma altor 8 cuvinte a câte 8 biți fiecare, ş.a.m.d.

Cele 8 porți logice care permit accesul la ieşire a celor 1024 cuvinte a câte 8 biți fiecare, sunt fie circuite cu colectorul în gol, fie circuite logice cu 3 stări, ambele variante permițând cuplarea memoriei ROM pe o magistrală de date.

Simbolizarea unei memorii ROM de 8Kbiți este prezentată în fig. 4.52.

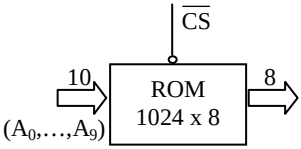



Fig. 4.52. Simbolizarea memoriei ROM de 8Kb

## *4.10.4. Extinderea capacității memoriilor ROM*

Cunoscut fiind faptul că dimensiunea (capacitatea) unei memorii ROM este dată de produsul dintre numărul de cuvinte de cod $m=2^n$ (unde *n* reprezintă numărul de linii de intrare) şi lungimea *k* a cuvântului de cod (de ieşire), rezultă că extinderea capacității se poate realiza prin interconectarea la intrare, la ieşire sau mixtă a mai multor memorii.

### *4.10.4.1. Extinderea la intrare a capacității memoriei ROM*

Extinderea la intrare (de adresă) a capacității memoriei ROM, implică o creştere a numărului de cuvinte de cod *m* şi păstrarea neschimbată a lungimii *k* a cuvântului, fig. 4.53.

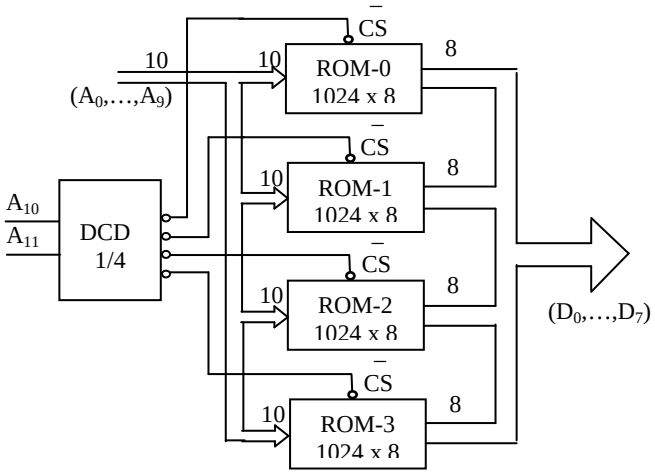



Fig. 4.53. Extinderea la intrare a capacității memoriei ROM

Se observă activarea succesivă de către combinațiile de cod ale liniilor suplimentare de adresă, $A_{10}$ şi $A_{11}$, a celor 4 memorii ROM de câte 8Kb fiecare.

Astfel, pentru combinația de adresă $A_{11}A_{10}$=00, va fi activată memoria ROM-0 al cărei conținut de 1024 cuvinte de cod a câte 8 biți fiecare, va avea acces la ieşire. Urmează activarea memoriei ROM-1 ($A_{11}A_{10}$=01), ş.a.m.d.

La ieşirea circuitului se obțin 4x(1024x8)biți=(4096x8)biți=(4x8)Kbiți.

### *4.10.4.2. Extinderea la ieşire a capacității memoriei ROM*

Extinderea la ieşire a capacității memoriei ROM implică o creştere a lungimii cuvântului de cod *k* şi păstrarea neschimbată a numărului cuvintelor de cod *m* furnizate la ieşire.

Concret, pentru a obține k=32biți, vom comanda cele 4 memorii ROM de 8Kb cu aceleaşi 10 linii de adresă, ieşirile memoriilor respective urmând a fi citite în paralel.

La ieşirea circuitului se obțin (1024x8x4)biți=(1024x32)biți=(1x32)Kbiți, adică 1024 cuvinte a câte 32 biți fiecare.

### *4.10.4.3. Extinderea mixtă a capacității memoriei ROM*

Extinderea mixtă a capacității memoriei ROM implică creşterea simultană a numărului de cuvinte de cod *m*, cât şi a lungimii *k* a cuvintelor, fig. 4.54.

Se observă activarea simultană, pentru $A_{10}$=0, a memoriilor ROM-0 şi ROM-2, urmată de activarea memoriilor ROM-1 şi ROM-3, pentru $A_{10}$=1.

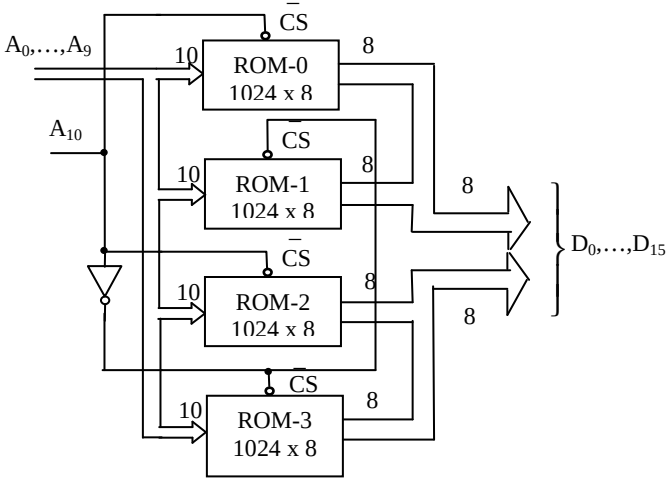



Fig. 4.54. Extinderea mixtă a capacității memoriei ROM

În prima fază se obţin (1024x8x2)biţi=(1024x16)biţi=(1x16)Kbiţi, pentru ca în final să rezulte (2x16)Kbiţi.

Dintre numeroasele aplicaţii ale memoriei ROM amintim:

- memorarea instrucţiunilor şi datelor în sistemele de calcul şi automatele secvenţiale;
- efectuarea transformărilor de adresă şi înmagazinarea microinstrucţiunilor în microprogramare;
- implementarea circuitelor logice combinaţionale cu un număr mare de intrări şi ieşiri, fără a mai fi necesară minimizarea;
- conversia de cod în vederea afişării pe 7 segmente sau prin 35 puncte;
- realizarea unor tabele de funcţii;
- generarea unor secvenţe de impulsuri, etc.

## *4.11. Arii logice programabile*

În cazul unor aplicaţii cu un număr mare de variabile de intrare şi viteze de lucru ridicate, utilizarea memoriilor fixe programabile la utilizator (PROM, EPROM, $E^2$PROM) poate deveni improprie sau neeconomică.

De asemenea, în situaţiile în care este necesară construirea unor circuite logice combinaţionale complexe care nu se fabrică în tehnologie integrată, implementarea acestora ar conduce la utilizarea mai multor circuite integrate interconectate între ele, ocupând un spaţiu mai mare pe circuitul imprimat, cu un consum sporit şi o fiabilitate mai redusă.

În toate aceste situaţii, *ariile logice programabile* prin mască la producător (*Programmable Logic Array = PLA*) sau pe cale electrică (*Field PLA = FPLA*) la utilizator, reprezintă o soluţie salvatoare.

Ca şi în cazul memoriei ROM, PLA / FPLA se compune dintr-un decodificator format dintr-o matrice programabilă de porţi ŞI, un codificator format dintr-o matrice programabilă de porţi SAU, precum şi amplificatoare de ieşire programabile.

Considerând schema logică a unei FPLA, fig. 4.55, observăm că aceasta prezintă 16 intrări ($I_0$, $I_1$, …, $I_{15}$), 3 niveluri de programare (la intrările porţilor ŞI, la intrările porţilor SAU şi la intrările porţilor SAU-EXCLUSIV), precum şi un nivel de porţi TSL pentru cuplarea celor 8 ieşiri ($O_0$, $O_1$, …, $O_7$) la magistrala de date.

Fuzibilele cu ajutorul cărora se face programarea, sunt simbolizate în fig. 4.55 prin cerculeţe.

În condiţiile în care toate fuzibilele sunt intacte, toţi termenii $P_k$ sunt nuli (în structura lor apar variabilele de intrare luate atât direct cât şi negate, v. principiul contradicţiei, § 1.2), termenii sumă $S_r$ – la fel, deci toate ieşirile circuitului vor fi în 0 logic.

Arderea fuzibilelor de la nivelul intrărilor matricei ŞI, va permite formarea termenilor produs de forma:

$$P_k = \prod_{0}^{15} \left( i_n \cdot I_n + j_n \cdot \overline{I_n} \right), \tag{4.44}$$

cu k=0, 1, …, 47 şi:

$i_n = j_n = 0$, dacă intrarea este neprogramată;

$i_n = \overline{j_n}$, dacă intrarea este programată;

$i_n = j_n = 1$, dacă intrarea este redundantă.

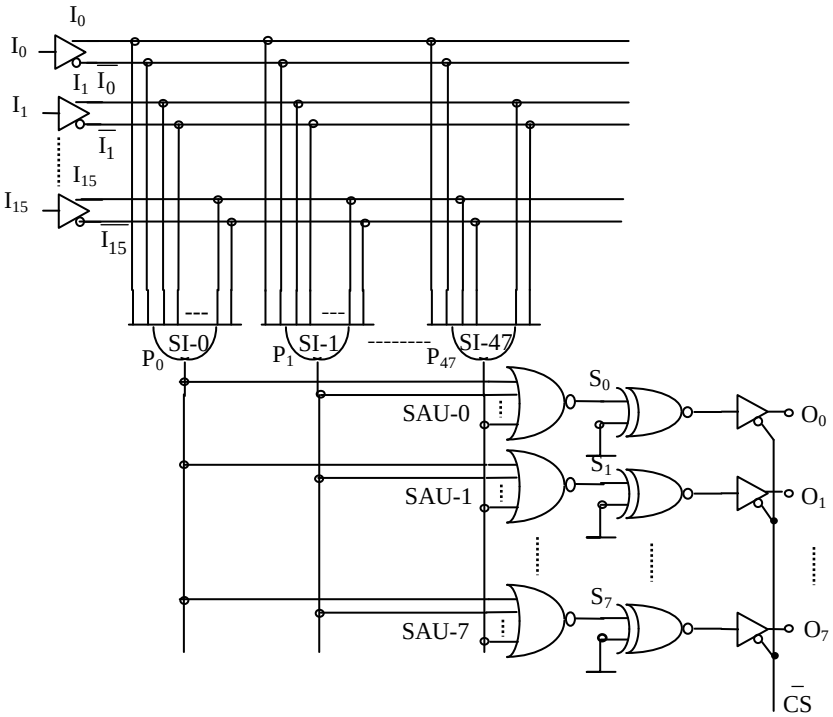



Fig. 4.55. Schema logică a unei FPLA

Programarea la nivelul intrărilor matricei SAU, permite formarea termenilor sumă de forma:

$$S_r = \sum_{0}^{47} t_k \cdot P_k ,$$

cu r=0, 1, …, 7 şi:

$t_k$=0, dacă $P_k$ este inactiv (programat);

$t_k$=1, dacă $P_k$ este activ (neprogramat);

În fig. 4.56 am prezentat o schemă concretă a unei FPLA, în care porţile ŞI sunt pasive şi realizate cu diode Schottky înseriate cu pelicule fuzibile de crom-

nichel, iar porţile SAU sunt constituite din tranzistoare în conexiune colector comun (repetor pe emiter), având pelicula fuzibilă conectată în emiter.

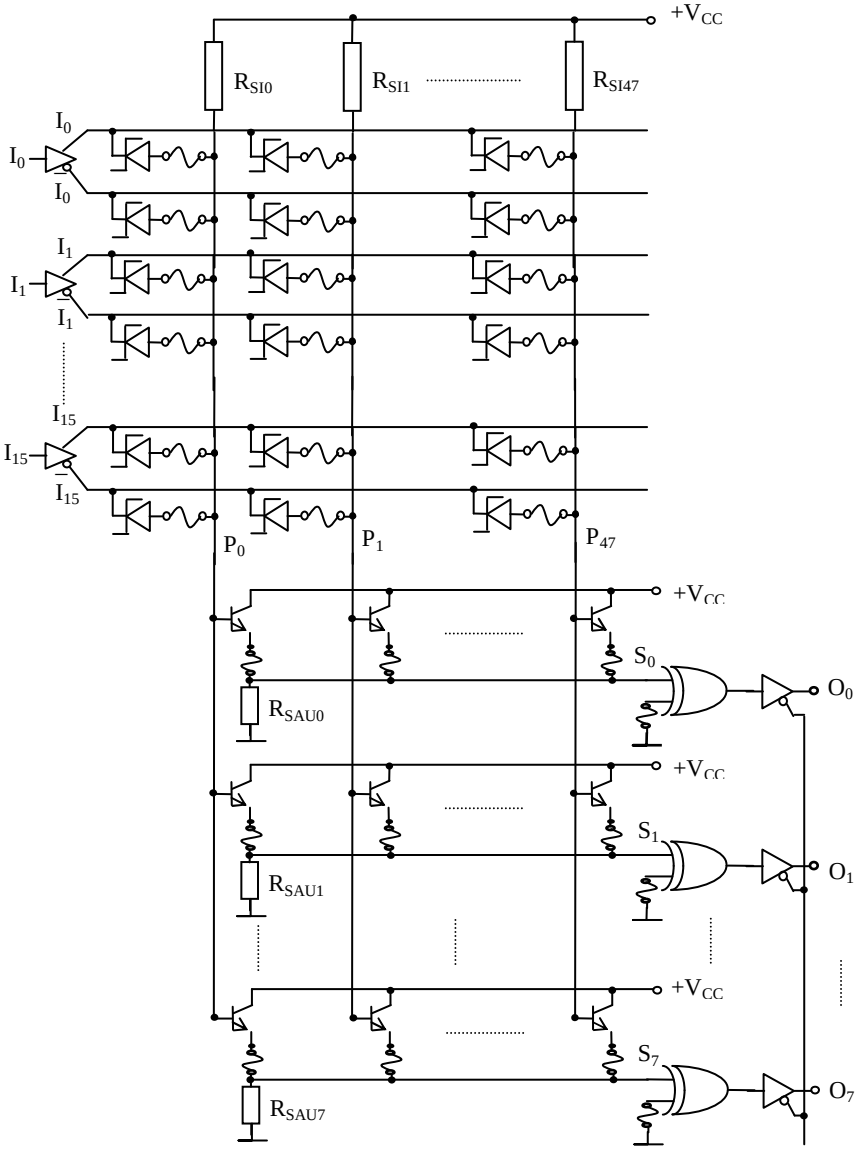



Fig. 4.56. Schema concretă a unei FPLA

Programarea FPLA se realizează pe baza unui tabel, cu ajutorul unui programator care permite selecţia şi arderea prin impulsuri de curent a fuzibilelor a căror întrerupere este necesară.

Comparativ cu o memorie ROM cu acelaşi număr de intrări (16) şi de ieşiri (8), PLA / FPLA este mult mai economică, prezentând o capacitate mult mai mică, 48 cuvinte x 8 biţi, faţă de $2^{16}$ cuvinte x 8 biţi în cazul memoriei ROM.

În general, în cazul unor aplicaţii care presupun un număr mare de variabile de intrare, principalele avantaje ale PLA / FPLA faţă de memoria ROM constau în posibilitatea programării matricei ŞI şi a complementării variabilelor de ieşire.

Ca şi în cazul memoriilor ROM, extinderea capacităţii PLA / FPLA este posibilă şi uzuală.