Minimizarea Logică Euristică

Note de curs
Dr.Ing.Mat. Ion I. Bucur

Minimizarea euristică este, în principiu, motivată prin nevoia reducerii mărimii formelor funcţiilor Boole-ene exprimate prin sume de produse (produse de sume) printr-un proces de calcul cu un consum cât mai redus de resurse (timp şi memorie).

Anumite minimizatoare euristice, cum ar fi *ESPRESSO* spre exemplu, produc acoperiri minimale cu cardinalitate identică, uneori, sau foarte apropiată acoperirilor minime şi au performanţe (timp de calcul si memorie ocupată) deosebit de bune. Aceste rezultate fac ca astfel de minimizatoare să fie utilizate în majoritatea situaţiilor practice, chiar dacă acoperirile calculate sunt doar minimale şi nu minime.

Au fost desemnate mai multe abordări ale minimizării euristice dintre care vor fi considerate numai câteva, cele mai reprezentative, în rândurile care urmează.

Primele minimizatoare euristice calculau toţi implicanţii primi şi utilizau algoritmi euristici pentru acoperirea optimală a tabelei de implicanţi primi.

Cele mai recente minimizatoare logice euristice nu mai calculează mulţimea tuturor implicanţilor primi, evitând astfel blocajele datorate depăşirii volumului disponibil de memorie operativă prin mulţimi foarte mari de implicanţi primi (o funcţie cu n variabile poate avea până la $3^n/n$ implicanţi primi). Acestea calculează, în locul mulţimii tuturor implicanţilor primi, o acoperire primă pornind chiar de la specificaţia iniţială a funcţiei.

Această acoperire este apoi procesată prin modificarea şi/sau înlăturarea unor implicanţi până când se stabileşte o acoperire având o cardinalitate minimală corespunzătoare. Sunt implementate, în marea majoritate a cazurilor, *strategii iterative de îmbunătăţire a cardinalităţii soluţiei*.

Minimizarea logică euristică poate fi privită ca fiind aplicarea repetată a unui set de operatori asupra acoperirii logice a funcţiei. Acest proces porneşte de la acoperirea funcţiei aşa cum este aceasta specificată iniţial, împreună (eventual) cu mulţimea termenilor pentru care funcţia nu este specificată.

În acest mod se calculează iterativ o acoperire minimală a funcţiei. Această acoperire este declarată soluţie a procesului de minimizare logică heuristică, atunci când aplicarea sistematică a operatorilor definiţi în sistem nu mai poate descreşte, mai departe, cardinalitatea acoperirii.

Operatorii utilizaţi cel mai des în minimizarea euristică sunt următorii:

- **Expand**, acest operator calculează o acoperire primă şi minimală în raport cu conţinerea într-un singur implicant. Implicanţii care formează acoperirea sunt procesaţi rând pe rând. Fiecare implicant care nu este prim este expandat într-un implicant prim, adică este înlocuit printr-un implicant prim care-l conţine.

Consecutiv, toţi implicanţii din acoperire (cei încă ne-procesaţi) care sunt conţinuţi în acest implicant prim sunt îndepărtaţi.

- **Reduce**, este un operator care realizează transformarea unei acoperiri oarecare, într-o acoperire ne-primă dar de aceeaşi cardinalitate cu cea iniţială. Implicanţii sunt procesaţi unul câte unul. Acest operator încearcă să înlocuiască fiecare implicant printr-un altul care este conţinut în acesta, cu condiţia că implicanţii reduşi împreună cu cei rămaşi continuă să acopere funcţia.
- **Reshape**, are drept acţiune modificarea acoperirii dar păstrând cardinalitatea acesteia. Implicanţii sunt procesaţi în perechi. Un implicant este expandat în timp ce celălalt implicant este redus cu respectarea permanentă a condiţiei de acoperire a funcţiei prin implicanţii procesaţi, la un moment dat, plus ceilalţi implicanţi din acoperire.

- **Irredundant**, are drept acţiune calculul unei acoperiri iredundante. Operatorul alege un subset al implicanţilor astfel încât nici un implicant al acestui subset nu este acoperit de ceilalţi implicanţi din subset.

Principalii operatori, care au fost enumeraţi anterior, sunt încadraţi în anumite strategii care fac ca rezultatele să fie superioare şi mai eficient calculate comparativ cu aplicarea directă a acestora. Între acestea sunt de remarcat, prin eficienţă, următoarele:

- Identificarea şi extragerea cât mai timpurie a implicanţilor esenţiali.
- Utilizarea complementului funcţiei pentru verificarea eficientă a expansiunii implicantului curent (aflat în expansiune) comparativ cu o expansiune anterior efectuată şi care ar putea conţine deja termenii canonici ai acestuia, invalidând adăugarea acestuia la acoperirea nou calculată.

Minimizatoarele logice euristice pot fi caracterizate în baza operatorilor utilizaţi şi a ordinii în care aceştia sunt aplicaţi în procesul de calcul.

Un minimizator euristic simplu poate fi implementat prin aplicarea operatorului *expand* o singură dată.
Această strategie, spre exemplu, a fost utilizată de programul *PRESTO*. În acest mod se poate obţine *primalitatea* şi *minimalitatea* în raport cu conţinerea într-un singur implicant, pentru o acoperire dată. Latura nefavorabilă a acestei abordări rezidă în faptul că se poate obţine în final o acoperire mult mai largă, comparativ cu cea de cardinalitate minimă.

Pentru ca să se evite soluţiile de calitate joasă, programul *MINI* (dezvoltat de IBM şi Universitatea Berkeley) a fost conceput să itereze prin aplicarea succesivă a operatorilor *expand, reduce* şi *reshape.* Acoperirea primă determinată de operatorul *expand* este procesată apoi de ceilalţi doi operatori astfel încât este favorizată, prin următoarea expandare, apariţia unei acoperiri de mărime redusă. Algoritmul implementat în această aplicaţie se încheie atunci când aplicarea acestor trei operatori nu mai produce acoperiri cu cardinalitate încă mai mică. Se remarcă faptul că procesul de calcul din *MINI* nu asigură iredundanţa acoperirii finale.

O abordare similară a fost aplicată în programele *POP* şi *PRESTO-L,* ambele extinzând abilităţile iniţiale ale programului original *PRESTO*.

Programul *ESPRESSO* utilizează operatorul *irredundant* pentru asigurarea iredundanţei acoperirilor. Algoritmul principal din *ESPRESSO* calculează întâi o acoperire primă şi iredundantă prin aplicarea operatorilor *expand* şi *irredundant*.

Apoi, se îmbunătăţeşte acoperirea, când este posibil, prin calculul unei secvenţe de acoperiri prime şi iredundante de cardinalităţi descrescătoare. Această suită de acoperiri se obţine iterativ prin aplicarea succesivă a operatorilor *reduce, expand* şi *irredundant,* dar şi prin comutarea euristicilor din implementările acestor operatori.

Exemplul 1.

Se consideră funcţia:

f = {(000 | 1), (0010 | 1), (0100 | 1), (1000 | 1), (1010 | 1), (0101 | 1), (0111 | 1), (1001 | 1), (1011 | 1), (1101 | 1)}, şi printr-o anumită metodă (Quine-McCluskey, bunăoară) s-au calculat implicanţii primi ai acestei funcţii:

p = (0--0 | 1), q = (-0-0 | 1), r = (01-- | 1), s = (10-- | 1), t = (1-01 | 1) şi u = (-101 | 1).

Implicanţii primi r şi t sunt esenţiali, deoarece aceştia acoperă mintermii (0111 | 1) şi (1011 | 1), respectiv.

Se presupune că se aplică întâi operatorul *expand*. Mintermii funcţiei sunt acoperirea iniţială a acesteia iar ordinea de aplicare a operatorului *expand* se presupune că este ordinea în care aceşti mintermi apar listaţi.

Mintermii care vor fi acoperiţi în urma expansiunii vor fi îndepărtaţi.
Astfel, făcând expansiunea mintermului (0000 | 1) acesta trece în implicantul prim p = (0--0 | 1).

Atunci se produce îndepărtarea mintermilor {(0010 | 1), (0100 | 1), (0110 | 1)} din acoperirea iniţială, deoarece aceştia sunt acoperiţi de p.

Se poate remarca faptul că mintermul (0000 | 1) putea avea şi alte expansiuni diferite.
Regulile euristice sunt cele care determină direcţia de expansiune.

Se presupune, spre exemplu, că expansiunea tuturor termenilor canonici, exceptând ultimul, a produs implicanţii primi p, q, r şi s.

Atunci expansiunea termenului canonic (1101 | 1) poate produce fie implicantul prim t = (1-01 | 1), fie implicantul prim u = (-101 | 1), în funcţie de direcţia de expansiune.

Dacă se presupune că s-a generat t atunci se obţine acoperirea {p, q, r, s, t} cu cardinalitatea 5.

Acoperirea calculată este primă, redundantă şi minimală în raport cu conţinerea într-un singur termen produs. Se poate remarca la această acoperire că diferă de lista tuturor implicanţilor primi ai funcţiei (are mai puţini termeni).

Acum se aplică operatorul *reduce* implicanţilor primi din acoperirea rezultată anterior.

Implicantul prim *p* poate fi redus la *implicantul vid* deoarece toţi termenii săi canonici sunt acoperiţi de ceilalţi implicanţi primi din acoperirea curentă.

Implicantul prim *q* = (-0-0 | 1), poate fi redus la q^* = (00-0 | 1), deoarece o parte din acesta este acoperită de *r*. Similar *t* = (1-01 | 1), poate fi redus la t^* = (1101 | 1). Rezultatul final a aplicării operatorului *reduce* este acoperirea de cardinalitate 4: {q^*, *r, s,* t^*}.

Un exemplu posibil de aplicare al operatorului *reshape* este următorul.
Perechea de implicanţi (q^*, *r*) poate fi schimbată cu perechea (*q,* r^*), unde r^* = (10-1 | 1). Aceasta conduce acum la acoperirea { *q,* r^*, *s,* t^*}.

O altă expansiune ar putea conduce la acoperirea {*q, r, s, t*} care este primă şi minimă.

În concluzie, este de reţinut că nu se garantează nici acoperirea minimă, nici iredundanţa acesteia numai prin iterarea operatorilor *expand, reduce* şi *reshape*.

Iredundanţa este garantată, doar, prin invocarea operatorului *irredundant* după operatorul *expand*, ceea ce ar detecta imediat, în exemplul acesta, că fie *p,* fie *q,* trebuie îndepărtat din acoperire.

□

Este important de realizat că minimizatoarele logice euristice pot diferi foarte mult prin modul în care sunt implementaţi operatorii, deoarece toţi operatorii conţin euristici.

Ordinea în care sunt consideraţi implicanţii, spre exemplu, pe durata unui proces de expansiune afectează mărimea acoperirii rezultate. Din acest motiv sunt implementate reguli euristice de ordonare a implicanţilor.

Drept urmare, rezultă că două implementări ale operatorului *expand* produc ambele acoperiri prime şi minimale în raport cu conţinerea într-un singur termen produs, dar *foarte posibil cu cardinalităţi diferite*.

Anumiţi algoritmi prin care se implementează operatorii au complexitatea supra-polinomială. Timpul de execuţie şi necesarul de memorie operativă depind de heuristicile locale ale operatorilor, care sunt factorul esenţial în realizarea minimizatoarelor euristice eficiente viabile în abordarea funcţiilor cu acoperiri complexe.

Exemplul 2.

Fişierul de intrare, de date, pentru programul *ESPRESSO*, este foarte intuitiv, fiind alcătuit din imaginea unei tabele de adevăr însoţită de câteva linii de comandă.

Fie funcţia scalară de patru variabile
$f(a,b,c,d) = m_4 + m_5 + m_6 + m_8 + m_9 + m_{10} + m_{13}$,

având următorii termeni canonici neprecizaţi:
$m_0 + m_7 + m_{15}$.

Iată fişierul de date scris pentru Espresso, corespunzător acestei funcţii:

```
.i      4
.o      1
.ilb    a       b       c       d
.ob     f
```

```
.p      10
0100    1
0101    1
0110    1
1000    1
1001    1
1010    1
1101    1
0000    -
0111    -
1111    -
.e
```

Primele două linii descriu numărul de intrări şi ieşiri ale funcţiei.

Următoarele două linii introduc numele variabilelor de intrare şi numele funcţiei.

Cea de-a cincea linie defineşte numărul de termeni produs utilizaţi pentru specificarea funcţiei.

În acest caz sunt în total zece (10) termeni produs din care şapte definiţi cu valoarea 1 şi trei neprecizaţi (s-a utilizat simbolul – pentru a desemna valori neprecizate).

Ultima linie arată încheierea fişierului de date de intrare.

Rezultatul minimizării, fişierul de ieşire, se prezintă astfel:

```
.i      4
.o      1
.ilb    a       b       c       d
.ob     f
.p      3
1-01    1
10-0    1
01--    1
.e
```

Rezultatul minimizării, exprimat literal, este reprezentabil astfel:

f(a,b,c,d) = ac'd + ab'd' + a'b.

De reţinut că minimizatorul *ESPRESSO* acceptă definiţii de funcţii utilizând şi termeni ne-canonici, adică, implicanţi de ordinul 1 sau mai mare.

Arii programabile de porţi

O soluţie ingenioasă de implementare a circuitelor logice combinaţionale o constituie blocurile programabile PLA (*programmable array logic*) sau PLA (*programmable logic arrays*).
Aceste blocuri logice programabile ad-hoc sunt structuri generalizate conţinând porţi ŞI şi porţi SAU (sau porţi SAU-NU ori porţi ŞI-NU) ale căror conexiuni sunt *programabile* astfel încât să poată fi implementate anumite funcţii specificate de utilizator.
În figura 1 este prezentată structura generică a unei arii de porţi programabile. Astfel de dispozitive multi-intrări / multi-ieşiri, sunt organizate tipic într-o arie de porţi ŞI şi o arie de porţi SAU. Prima arie implementează termenii produs în funcţie de

conexiunile programabile ale intrărilor la porţile ŞI. A doua arie utilizează termenii produs calculaţi în prima arie pentru conectarea programată a acestora la porţile SAU realizând astfel disjuncţia termenilor doriţi pentru fiecare funcţie în parte.
Un dispozitiv PLA poate implementa o colecţie, mai degrabă, restrânsă de funcţii de complexitate considerabilă. Această complexitate este determinată de numărul de linii de intrare, de numărul de termeni produs (numărul de porţi ŞI) şi de numărul de funcţii, de ieşiri, respectiv de porţi SAU ale circuitului PLA.

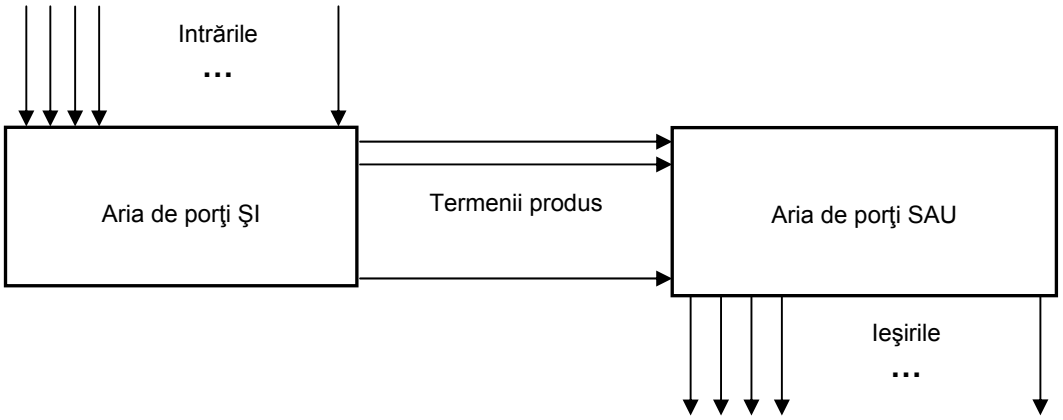

Figura 1. Structura unei arii programabile.

O arie logică programabilă ad-hoc (*field-programmable logic array* ***FPLA***) tipică, realizată în tehnologie TTL, spre exemplu, poate avea 16 linii de intrare şi 8 linii de ieşire. Încapsulată cu 24 de pini această arie programabilă este echivalentul a 48 de porţi ŞI cu 16 linii de intrare fiecare, plus opt porţi SAU cu câte 48 de linii de intrare fiecare. Aceste date sunt precizate ca să se remarce eficienţa utilizării acestor dispozitive.
O cale convenabilă de descriere a funcţiilor implementate prin PLA-uri este aşa-numita *matrice de implementare*. Această matrice este, în fapt, o transcriere a tabelei de adevăr a sistemului de funcţii care se va implementa printr-o PLA.

Exemplul 3.

Se presupune că se doreşte implementarea următoarelor patru funcţii:
$f_0 = a + b'c'$; $f_1 = ac' + ab$; $f_2 = b'c' + ab$; $f_3 = b'c + a$.
Se poate caracteriza implementarea prin numărul de variabile (trei: *a, b* şi *c*), termenii produs unici (*a, b'c', ac', ab, bc'*) şi funcţiile, patru la număr. Aceste date corespund numărului de intrări în aria de circuite ŞI, numărului de ieşiri din porţile acestea (corespunzător numărul de termeni produs şi respectiv de linii de intrare în porţile SAU) şi numărului de linii de ieşire din PLA.

Tabelul 1.
Matricea de implementare pentru funcţiile exemplului 3

Termenii produs	Intrări *a b c*	Ieşiri $f_0\ f_1\ f_2\ f_3$
ab	1 1 -	0 1 1 0
b'c	- 0 1	0 0 0 1
ac'	1 - 0	0 1 0 0
b'c'	- 0 0	1 0 1 0
a	1 - -	1 0 0 1

Matricea de implementare pentru acest sistem de funcţii este prezentată mai sus. Se poate uşor remarca din aceasta că de îndată ce un produs este utilizat

în realizarea a mai mult de două funcţii, pe linia respectivului produs apar tot atâtea unităţi în dreptul funcţiilor respective. Se presupune, pentru uşurinţa urmăriri exemplului, că este disponibilă o arie logică programabilă având trei linii de intrare, patru linii de ieşire şi putând implementa cinci produse.
În figura 2 este prezentată aria logică programabilă înainte de programare.

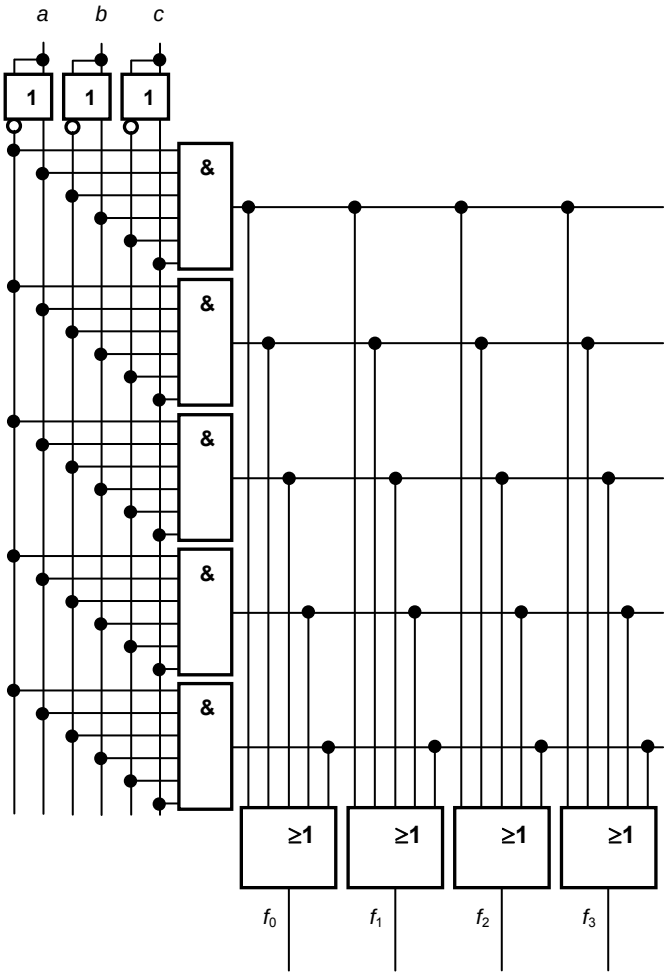

Figura 2. Circuitul PLA înainte de programare, exemplul 3.

Se remarcă, în figura 2, prezenţa unor inversoare ataşate fiecărei linii, facilitând implementarea termenilor produs care conţin variabile complementate. De asemenea, se observă conectarea completă a întregului set de linii de intrare la toate porţile ŞI, precum şi conectarea exhaustivă a liniilor de ieşire din porţile ŞI la liniile de intrare în porţile SAU.
Practic sunt prezente toate conexiunile posibile înainte de programare. Programarea dispozitivelor PLA se face cu un dispozitiv special numit programator şi care funcţionează după cum este realizată tehnologic programarea.
O tehnică frecvent utilizată pentru realizarea programării PLA-urilor este folosirea fuzibililor. Fuzibilii sunt reprezentaţi, în figura 2, prin punctele ingroşate plasate la intersecţia liniilor interne din aria logică programabilă. Fuzibilii sunt plasaţi, în general, oriunde se doreşte întreruperea unei conexiuni.

Programarea constă în eliminarea conexiunilor nedorite, iar fizic aceasta are loc prin trecerea unui curent de valoare bine definită care provoacă arderea fuzibilului respectiv fără să fie afectată funcţionarea restului dispozitivului.

Conexiunile intrărilor unei porţi ŞI la liniile de intrare (atât cele asertate cât şi cele complementate) sunt făcute prin fuzibili, spre exemplu. Astfel, termenii produs sunt implementaţi prin arderea fuzibililor corespunzători literalilor neutilizaţi.

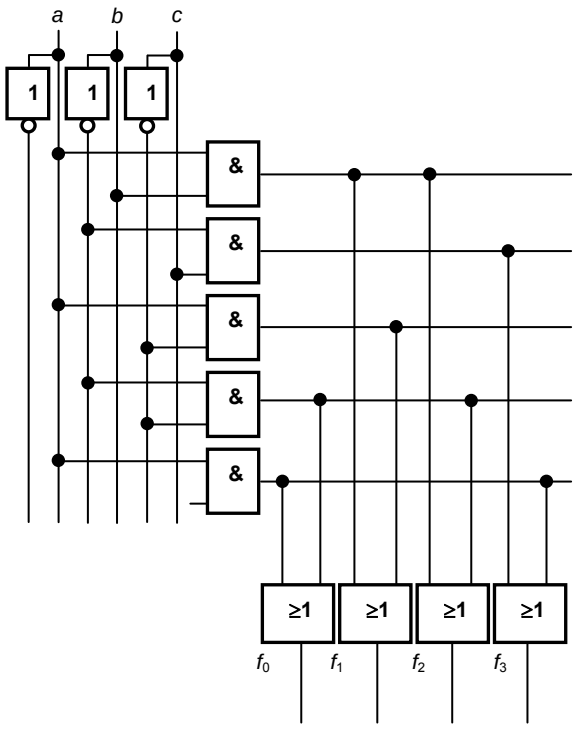

Figura 3. Circuitul PLA după programare, exemplul 3.

Modul generic sau convenţional, cel mai uzitat, de reprezentare a unui circuit PLA este prezentat în figura 4.

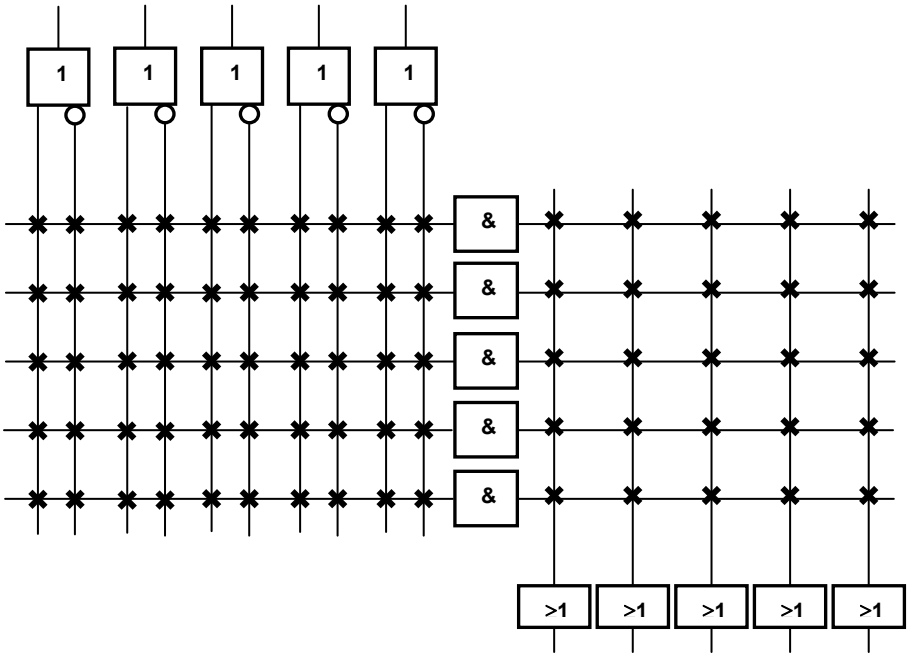

Figura 4. Reprezentarea convenţională a dispozitivelor PLA.

Se remarcă utilizarea unor „fire” generice care leagă literalii de circuitele ŞI, situaţie similară fiind pentru formarea sumelor de produse. Aceste fire trebuie înţelese ca fiind multiple iar X-urile reprezintă locaţiile fuzibililor.

Minimizarea euristică a funcţiilor vectoriale booleene

Un circuit PLA, aşa cum s-a putut remarca, este o macro-celulă rectangulară constând dintr-o arie de tranzistoare aliniate astfel încât să formeze rândurile în corespondenţă cu termenii produs şi coloanele în corespondenţă cu relaţia intrări-ieşiri. Coloanele intrărilor şi ieşirilor partiţionează *PLA* în două sub-arii: planul de intrare şi respectiv planul de ieşire (aşa cum se poate urmări în figura 4).

Fiecare rând al *PLA*-ului este în corespondenţă biunivocă cu un termen produs aparţinând respectivei sume de produse.
Fiecare tranzistor din planul de intrare este în corespondenţă biunivocă cu un literal al formei sumă de produse.
Oricare tranzistor din planul de ieşire este legat de o ieşire scalară a termenului produs.
Din aceste motive prima ţintă a minimizării logice este reducerea numărului de produse şi ţinta secundară este reducerea literalilor din produse.

Alte obiective de optimizare sunt relevante atunci când funcţiile modelate prin forme în două nivele sunt implementate altfel decât prin *PLA*-uri.
Reprezentări logice în două nivele pentru funcţii scalare, spre exemplu, pot fi implementate prin porţi complexe a căror mărime este corelată cu numărul de literali din forma factorizată a acelei reprezentări. În astfel de situaţii obiectivul major este minimizarea numărului de literali.

Minimizarea logică pentru o funcţie scalară sau o funcţie vectorială se face după aceleaşi principii, dar cazul vectorial este mult mai complex.
Minimizarea disjunctă a componentelor scalare ale unei funcţii vectoriale poate conduce la rezultate suboptimale deoarece optimizarea nu poate exploata comunitatea unor termeni produs.

Un rezultat important în optimizarea logică în două nivele este echivalenţa funcţiilor vectoriale booleene de variabile booleene cu funcţiile booleene scalare de variabile multi-valorice. Din astfel de raţiuni, pentru început abordarea se va concentra asupra tehnicilor de optimizare ale funcţiilor scalare de variabile binare şi multi-valorice.

În vederea reliefării diferenţelor dintre minimizarea vectorială şi minimizarea scalară a componentelor unei funcţii vectoriale se consideră decodificatorul dispozitivului de afişare cu şapte segmente din exemplul următor.

Exemplul 8.

Se consideră proiectarea unui decodificator (a se vedea figura 5) care să preia informaţia de pe o magistrală cu patru linii (x_8, x_4, x_2 şi x_1) pe care sunt vehiculate valori zecimale codificate binar şi să activeze corespunzător un dispozitiv de afişare cu şapte segmente.

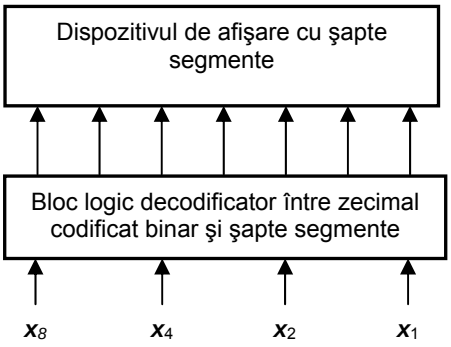

Figura 5. Schema bloc a decodificatorului.

Modul în care sunt formate cifrele zecimale cu dispozitivul de afişare este arătat în figura 6.

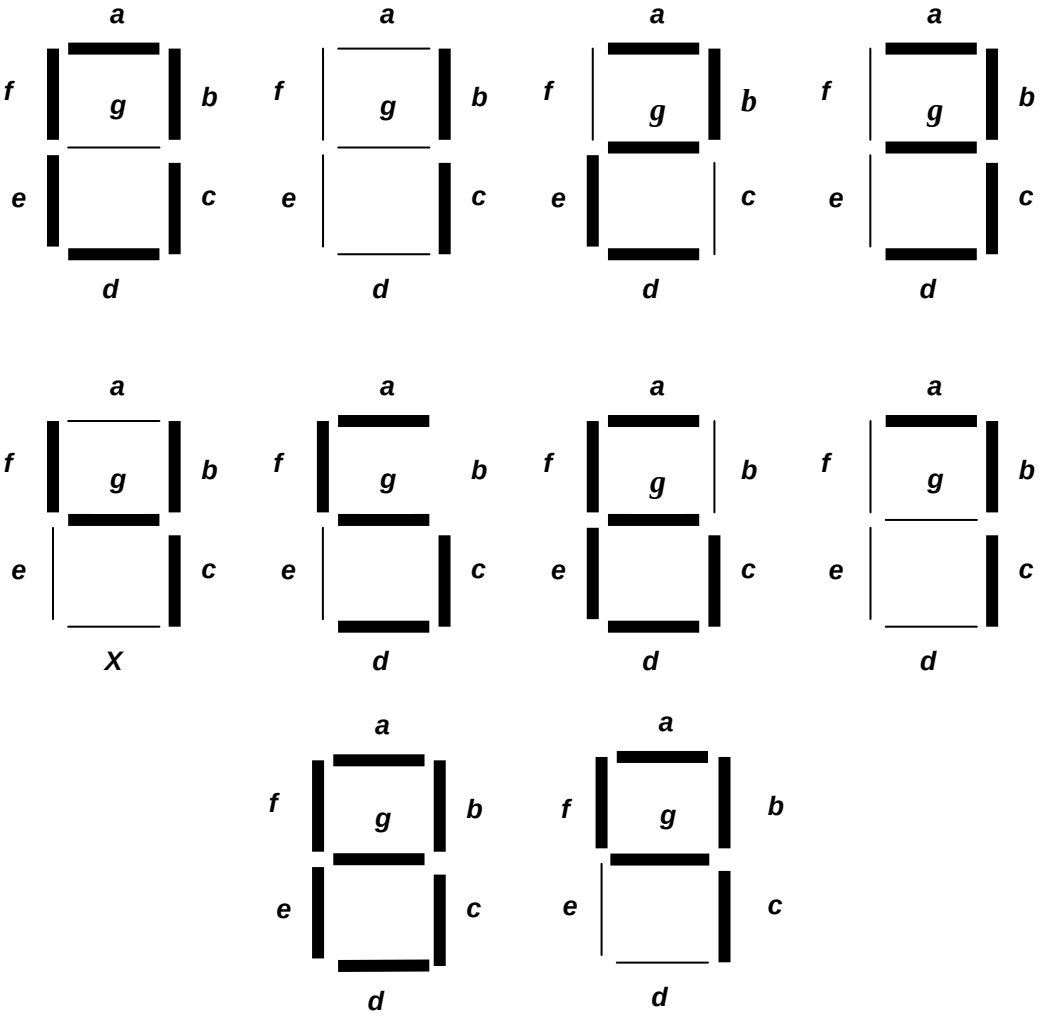

Figura 6. Configuraţia cifrelor zecimale pentru dispozitivul de afişare cu 7 segmente.

Tabelul 2.
Valorile liniilor de ieşire ale decodificatorului pentru dispozitivul de afişare cu şapte segmente în funcţie de valorile liniilor de intrare.

Index	x_8	x_4	x_2	x_1	*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	0	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**

Corespondenţa dintre valorile binare ale vectorului intrărilor (x_8 x_4 x_2 x_1) în decodificator şi modul de activare al segmentelor de afişare, conform figurii 6, este prezentat în tabelul 1 (înfăţişat anterior).
Deoarece pentru cifrele zecimale sunt utilizate doar codurile binare de la 0000 la 1001, pentru codurile binare rămase neutilizate (de la 1010 la 1111) valorile ieşirilor decodificatorului sunt nespecificate (valori ***X***).

Pentru realizarea decodificatorului trebuie aleasă o modalitate de implementare. Se poate implementa decodificatorul cu ajutorul unei memorii cu conţinut fix, spre exemplu. În această situaţie proiectarea se încheie, practic, aici.
Tot ceea ce mai trebuie făcut este încărcarea conţinutului coloanelor ***a***, ***b***, …, ***g*** din primele 10 linii ale tabelului într-o memorie cu cel puţin 10 locaţii de câte, minimum, 7 biţi fiecare.
Se pot găsi, uşor, astfel de memorii, chiar dacă sunt mai mari decât ar fi necesar.
Se poate întâmpla ca o astfel de soluţie să nu fie de utilitate din vari motive (costul memoriei, disponibilitatea altor circuite etc.).

De remarcat faptul că există disponibile componente, circuite integrate MSI, deja manufacturate care implementează această funcţie combinaţională. Este cazul, spre exemplu, componentei ***TTL 74LS48*** şi se mai pot găsi şi altele similare.

S-ar putea ca din motive de afişare a cifrei 6 să nu fie acceptabile nici una dintre aceste componente şi în această situaţie trebuie găsite alte soluţii.
Deoarece decodificatorul poate fi implementat printr-un circuit combinaţional reprezentabil printr-o formulă booleană, sumă de produse, se poate recurge la utilizarea diagramelor Karnaugh sau la utilizarea programului *ESPRESSO* pentru minimizarea euristica (sau exactă).

Ambele căi de abordare vor fi utilizate şi comparate pentru înţelegerea particularităţilor respectivelor metode.
Metoda diagramelor Karnaugh este relativ simplă de aplicat deoarece este convenabil numărul de variabile.

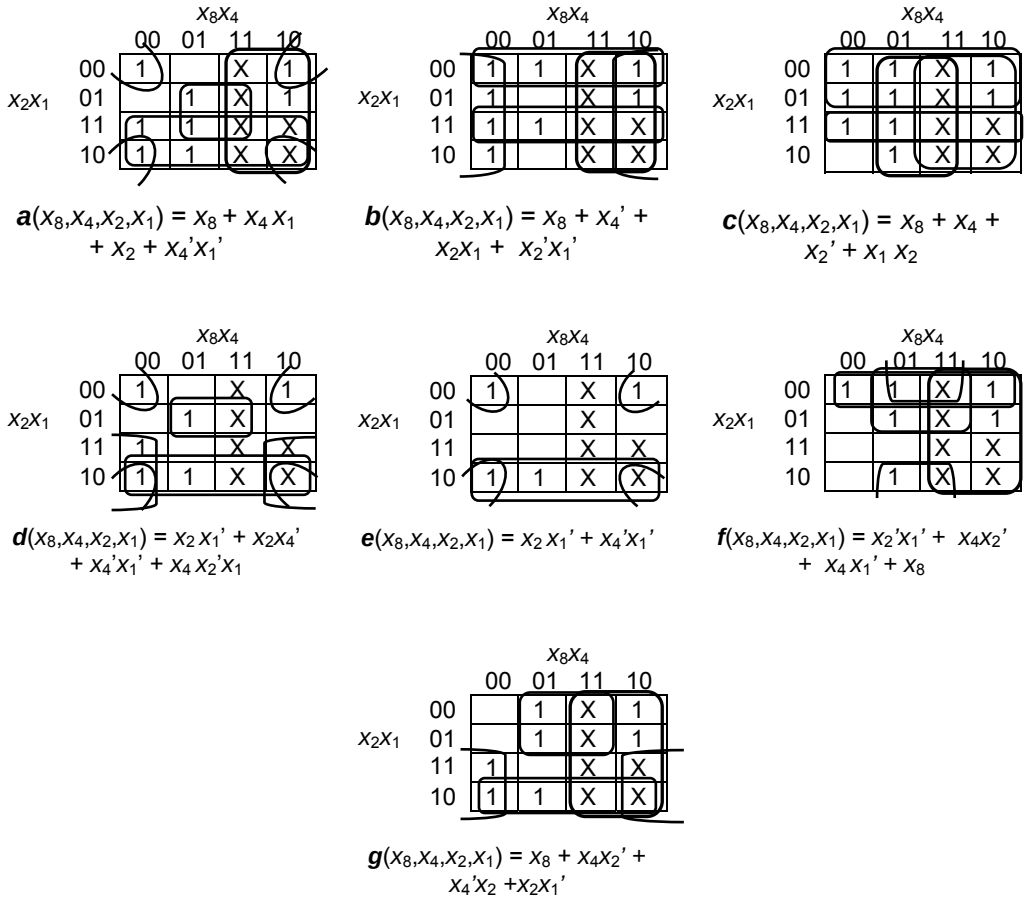

Figura 7. Minimizarea scalară, prin metoda diagramelor Karnaugh, a funcţiilor decodificatorului pentru 7 segmente.

Utilizând rezultatele obţinute prin metoda Karnaugh, fără să se considere posibilii termeni partajaţi, este necesar un circuit PAL cu patru variabile de intrare, şapte linii de ieşire cu cel puţin patru termeni produs (pentru fiecare linie de ieşire) pentru implementarea decodificatorului acesta.

S-ar putea folosi, spre exemplu, componenta P16H8 PAL. Se poate întâmpla ca printre condiţiile proiectării circuitului să se specifică utilizarea unui circuit PLA pentru implementare, atunci trebuie ţinut cont de faptul că una dintre limitările, relativ severe, ale acestor componente este numărul de termeni produs unici care sunt utilizaţi.
Aceştia corespund numărului de „fire” orizontale din structura circuitului.

O componentă PLA tipică poate oferi 16 linii de intrare, opt linii de ieşire şi 48 de termeni produs (cum este, spre exemplu, componenta F100 PLA).

Metodele de optimizare a circuitelor reprezentate prin sume de produse (sau circuite în două nivele) pun întotdeauna în evidenţă termenii partajabili.

Fişierul de date de intrare pentru minimizatorul *ESPRESSO* reflectă tabelul de corespondenţă al decodificatorului:

```
.i 4
.o 7
.ilb x8 x4 x2 x1
.ob a b c d e f g
.p 16
0000 1111110
0001 0110000
0010 1101101
0011 1111001
0100 0110011
0101 1011011
0110 1011111
0111 1110000
1000 1111111
1001 1110011
1010 - - - - - - -
1011 - - - - - - -
1100 - - - - - - -
1101 - - - - - - -
1110 - - - - - - -
1111 - - - - - - -
.e
```

Rezultatele obţinute prin metoda de minimizare euristică conduc la doar 9 termeni produs distincţi, după cum arată fişierul cu rezultate al minimizatorului *ESPRESSO*:

```
.i 4
.o 7
.ilb x8 x4 x2 x1
.ob a b c d e f g
.p 9
-10-  0000001
-01-  0001001
-0-1  0110000
-101 1011010
--00  0110010
--11  1110000
-0-0  1101100
1---   1000011
-110  1011111
.e
```

Corespunzător acestor termeni produs se pot scrie expresiile fiecărei linii de ieşire:

$\boldsymbol{a} = x_4x_2'x_1 + x_2x_1 + x_4'x_1' + x_8 + x_4x_2x_1'$
$\boldsymbol{b} = x_4'x_1 + x_2'x_1' + x_2x_1 + x_4'x_1'$
$\boldsymbol{c} = x_4'x_1 + x_4x_2'x_1 + x_2'x_1' + x_2x_1 + x_4x_2x_1'$
$\boldsymbol{d} = x_4'x_2 + x_4x_2'x_1 + x_4'x_1' + x_4x_2x_1'$
$\boldsymbol{e} = x_4'x_1' + x_4x_2x_1'$
$\boldsymbol{f} = x_4x_2'x_1 + x_2'x_1' + x_8 + x_4x_2x_1'$
$\boldsymbol{g} = x_4x_2' + x_4'x_2 + x_8 + + x_4x_2x_1'$

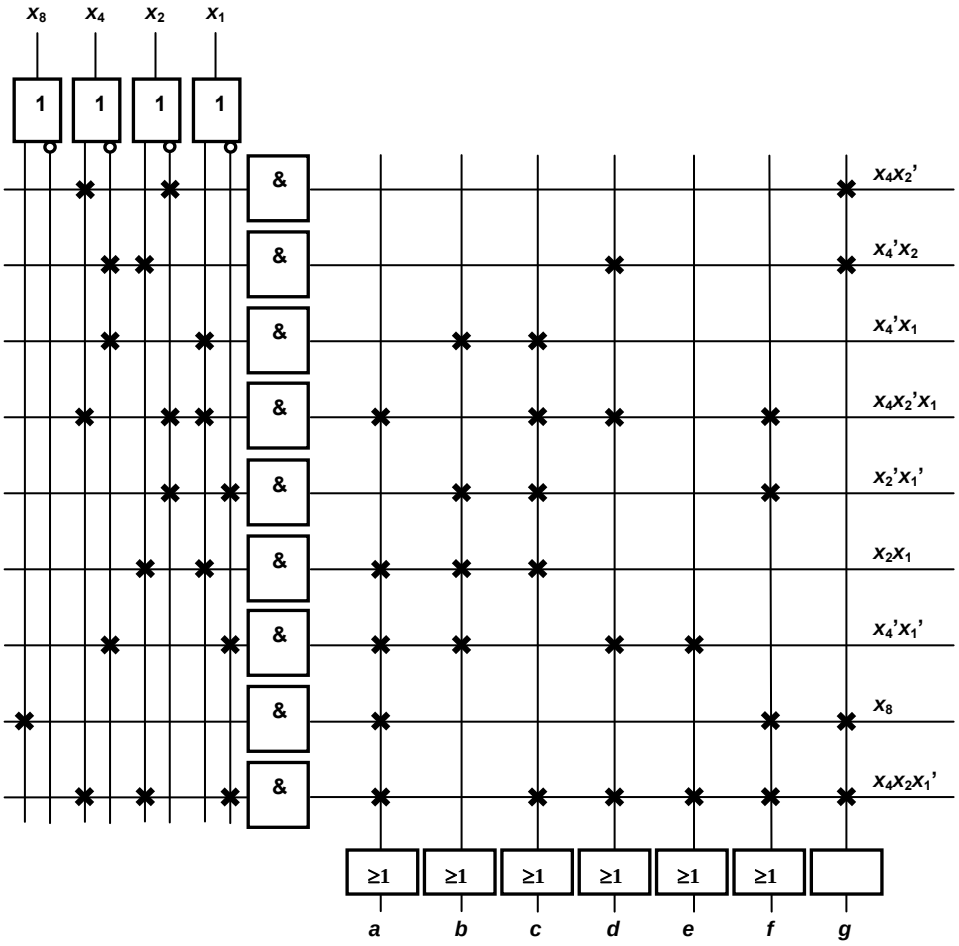

Figura 8. Implementarea minimizării *ESPRESSO* a decodificatorului pentru afişajul cu 7 segmente.

Se remarcă, într-o primă aproximaţie, apariţia unei complexităţi mai mari a funcţiilor segmentelor de afişare (63 de literali) comparativ cu rezultatele obţinute prin metoda de minimizare a diagramei Karnaugh.
La o privire mai atentă, totuşi, se poate observa faptul că numărul de termeni produs distincţi a scăzut de la 15 la 9.
Chiar dacă expresiile individuale ale fiecărei funcţii au, fiecare în parte, un număr mai mare de termeni produs, datorită gradului ridicat de partajare a acestora complexitatea, globală a acestei variante, este mai mică.
Mărimea dispozitivelor PLA este determinată, în primul rând, de numărul de termeni produs. Minimizatorul *ESPRESSO*, a fost proiectat, între altele, pentru astfel de circuite.
Partajarea termenilor produs nu este de nici un folos în implementarea circuitelor combinaţionale prin componente PAL care nu pot pune în valoare termenii produs partajaţi dintre liniile de ieşire.
Dacă se urmăreşte o implementare prin dispozitive PAL este indicat să se specifice minimizatorului *ESPRESSO* o abordare individuală, scalară, a fiecărei funcţii în parte (nu în sistem).

Aspectele practice ale proiectării circuitelor combinaţionale

Până acum abordarea a fost concentrată asupra aspectelor matematice ale reprezentării reţelelor combinaţionale prin funcţii booleene precum şi asupra tehnicilor şi algoritmilor de obţinere de expresiilor booleene minimizate. În această secţiune atenţia se va focaliza asupra unor chestiuni cu caracter mai practic, vizând tehnologiile folosite pentru implementarea funcţiilor combinaţionale, modul în care sunt prezentate porţile în circuitele integrate produse industrial precum şi tehnicile standard pentru documentarea schemelor logice.
Pentru un proiect dat, anumit, există o tehnologie de implementare vizată care specifică atât elementele primare disponibile cât şi proprietăţile acestor elemente. Adiţional, tehnologia de implementare aduce cu sine, adeseori, o colecţie de caracteristici restrictive de care trebuie ţinut seama atunci când se face proiectarea.
În cele ce urmează vor fi abordate funcţiile porţilor primare împreună cu proprietăţile acestor şi vor fi succint caracterizate restricţiile şi compromisurile care trebuiesc luate în consideraţie atunci când se ţinteşte o implementare printr-o tehnologie dată.

Parametrii tehnologici

Atunci când sunt luate în consideraţie aspecte ale performanţei unui circuit combinaţional se dovedeşte că există diferenţe sensibile, de la caz la caz, între tehnologiile utilizabile ceea ce poate face ca o tehnologie să fie preferată, într-un context sau altul.

Circuitele digitale sunt construite cu circuite integrate. Un circuit integrat (IC este abrevierea anglo-saxonă, curent utilizată) este un cristal semiconductor din siliciu (numit, tradiţional, *chip*) care conţine toate componentele electronice care alcătuiesc porţile digitale şi elementele cu memorie. Diferitele componente electronice sunt interconectate pe chip. Chipul este montat într-o capsulă ceramică ori de plastic, iar conexiunile chipului la pinii externi ai capsulei se realizează prin sudură, obţinându-se în acest mod circuitul integrat. Numărul de pini poate fi cuprins între14 (pentru circuitele integrate de dimensiuni modeste) şi câteva sute (în cazul capsulelor mari ale circuitelor digitale complexe, cum ar fi microprocesoarele). Fiecare circuit integrat are o anumită informaţie de identificare. Această informaţie este, de regulă, imprimată pe suprafaţa capsulei circuitului. Fiecare producător de circuite integrate tipăreşte foi de catalog pentru fiecare circuit integrat propus. Aceste foi de catalog conţin toate informaţiile necesare utilizării respectivului circuit integrat.

Nivelurile de integrare

În timp, pe măsură ce tehnologia de fabricaţie a circuitelor integrate s-a îmbunătăţit, numărul de porţi integrate, pe un singur chip, a crescut considerabil. Tradiţional, caracterizarea unui circuit integrat nominalizează un anumit grad de integrare, un anumit număr de porţi integrate pe un chip.
Astfel, există categorii de circuite integrate pe scară mică, medie, largă şi foarte largă. Aceste categorii diferenţiază circuitele integrate cu doar câteva porţi per chip, de circuitele integrate cu câteva mii de porţi ori faţă de circuitele integrate cu câteva zeci de milioane de porţi.

Principalele categorii de circuite integrate sunt:

- Circuitele integrate pe scară mică (abrevierea anglo-americană este SSI, pentru *small-scale integrated*) conţin, într-o singură capsulă, câteva porţi primare independente. Liniile de intrare şi ieşire ale fiecărei porţi sunt conectate la pinii capsulei. Numărul de porţi, în mod uzual, al unui astfel de circuit integrat este cel mult 10, fiind limitat de numărul de pini ai capsulei.
- Circuitele integrate pe scară medie (abrevierea anglo-americană este MSI, pentru *medium-scale integrated*) conţin, de regulă, până la 100 de porţi într-o singură capsulă. Aceste circuite sunt dedicate, uzual, unor funcţii digitale elementare specifice, cum ar fi sumarea numerelor binare cu patru ranguri.
- Circuitele integrate pe scară largă (abrevierea anglo-americană este LSI, pentru *large-scale integrated*) conţin între 100 şi câteva mii de porţi, într-o singură capsulă. Această categorie de circuite cuprinde procesoare de mică complexitate, memorii cu volume reduse şi module programabile.
- Circuitele integrate pe scară foarte largă (abrevierea anglo-americană este VLSI, pentru *very large-scale integrated*) conţin de la câteva mii până la zeci de milioane de porţi, într-o singură capsulă. Exemple, tipice, de circuite integrate din această categorie sunt microprocesoarele complexe şi procesoarele digitale de semnal (DSP). Dimensiunile mici ale tranzistoarelor utilizate, densitatea mare şi preţurile, comparativ, mici au revoluţionat proiectarea electronică şi digitală. Această categorie de circuite integrate a făcut posibilă crearea unor structuri complexe care, anterior, nu erau economic realizabile.

Tehnologii ale circuitelor integrate

Circuitele integrate digitale sunt clasificate nu numai după funcţionalitatea acestora. Un alt criteriu important este tehnologia specifică de integrare. Fiecare tehnologie are propriile dispozitive electronice de bază şi propriile structuri de circuit peste care sunt dezvoltate funcţii şi circuite digitale complexe. Dispozitivele electronice, specifice, utilizate în construcţia circuitelor de bază dau şi numele tehnologiei.

Primele circuite digitale au avut la bază tranzistoarele bipolare având drept material de bază siliciul. Circuitele digitale bipolare sunt reprezentate prin cele două mari familii:

- TTL (*trasistor-transistor logic*) şi
- ECL (*emitter-coupled logic*).

Circuitele integrate care au urmat, utilizând ca semiconductor tot siliciul, au drept dispozitive electronice de bază tranzistoarele MOS (abrevierea denumirii anglo-americane *Metal Oxide Semiconductor*). Tehnologia CMOS (abrevierea denumirii anglo-americane *Complementary Metal Oxide Semiconductor*), domină datorită densităţii mari a circuitelor, performanţei ridicate şi consumului redus de putere. Această tehnologie utilizează atât tranzistoare MOS cu canal *p* cât şi tranzistoare MOS cu canal *n*. Tehnologii alternative bazate pe arseniura de galiu (GaAs)şi siliciu-germaniu (SiGe) sunt utilizate selectiv pentru circuite de foarte mare viteză.

În tehnologia MOS cea mai cunoscută este familia de circuite CMOS (*complementary* MOS). Această tehnologie utilizează atât tranzistoare MOS cu canal *p* cât şi tranzistoare MOS cu canal *n*.

Parametrii tehnologici

Pentru fiecare tehnologie specifică care urmează să fie utilizată pentru o anumită implementare, există detalii care diferă atât prin parametrii circuitelor integrate cât şi prin proiectarea circuitelor electronice respective. Printre cei mai importanţi parametrii utilizaţi pentru caracterizarea unei tehnologii de implementare se pot enumera:

- Numărul de linii de intrare disponibile pentru o poartă. Acest parametru tehnologic este numit , tradiţional, *fan-in*. În tehnologiile de mare viteză numărul de linii de intrare într-o poartă este, adesea, limitat. Astfel, pentru porţile primare fan-in este limitat la valori relativ mici, patru ori cinci linii de intrare într-o poartă, spre exemplu.
- Numărul de conexiuni ale liniei de ieşire ale unei porţi. Acest parametru vizează numărul de sarcini standard care por fi conectate acestei linii fără să se deprecieze performanţa porţii. Sarcinile standard pot fi definite printr-o largă varietate de moduri, dependent de tehnologie. Acest parametru tehnologic este numit , tradiţional, *fan-out*. O abordare a evaluării *fan-out*-ului constă în utilizarea modelului *sarcinii standard.* Orice poartă *H* având una dintre intrările sale conectate la ieşirea porţii *G* constituie o sarcină pentru poarta *G*. Această sarcină se măsoară în unităţi standard. Linia de intrare a unui inversor anumit *H*, spre exemplu, poate încărca linia de ieşire a porţii *G* printr-o sarcină standard având valoarea 1,0. Dacă poarta *G* conduce şase asemenea inversoare, atunci *fan-out*-ul porţii *G* este 6. O poartă, în general, are specificată o anumită valoare maximă sarcinii ce-o poate conduce. Această valoare maximă este *fan-out*-ul porţii respective. Determinarea *fan-out*-ului maxim al unei porţi depinde de familia logică din care face parte poarta respectivă. Datorită utilizării larg a tehnologiei CMOS, în continuare, consideraţiile se vor restrânge numai asupra acestei familii tehnologice. Pentru porţile CMOS încărcarea liniei de ieşire prin cablaj şi liniile de intrările porţilor conectate se modelează prin capacitatea echivalentă. Încărcarea capacitivă nu prezintă nici un efect asupra nivelelor logice ale semnalelor, aşa cum se întâmplă în alte familii logice. Efectul încărcării capacitive se manifestă prin timpul necesar de trecere a liniei de ieşire a porţii din valoarea coborâtă (notată, tradiţional, simbolic L) în valoarea ridicată (notată, tradiţional, simbolic H) şi invers. Dacă sarcina ieşirii este crescută atunci, acest timp numit *timpul de tranziţie*, creşte. Cu alte cuvinte, *fan-out*-ul maxim al unei porţi este numărul de sarcini standard ale capacităţii ce poate fi condusă (încărcată) într-un timp mai scurt, sau cel mult egal cu timpul de tranziţie. Astfel, o poartă având un fan-out egal cu maximum de 8 unităţi standard de sarcină va putea conduce, comuta, până la 8 inversoare care prezintă pe intrarea lor cel mult o sarcina standard. Deoarece *fan-out*-ul reprezintă, în fapt, o capacitate plasată la ieşirea unei porţi rezultă că *fan-out*-ul porţii afectează, deasemenea, timpul de propagare al porţii respective.
- Costul unei porţi, specifică o măsură a contribuţiei acesteia la costul circuitului integrat care-o conţine. Pentru circuitele integrate, costul unei porţi primare se bazează pe aria ocupată de respectiva poartă primară. Aria ocupată este proporţională cu mărimea tranzistoarelor şi cu conexiunile porţii pe chip (ceea ce se numeşte *layout*). Dacă se ignoră suprafaţa conexiunilor, atunci suprafaţa unei porţi este proporţională cu numărul de tranzistoare din poartă. Numărul de tranzistoare dintr-o poartă este proporţional cu numărul liniilor de intrare în

poartă. În cazul în care se cunoaşte aria actuală a poziţionării (*layout*-ului) unei porţi, atunci o valoare normalizată a acestei arii poate furniza o estimare mai corectă a costului porţii decât simplul număr de linii de intrare al porţii.

- Întârzierea unei porţi se referă la timpul necesar pentru ca o poartă să reacţioneze la schimbarea valorii uneia sau mai multora dintre liniile sale de intrare. Atunci când schimbarea valorii unei linii de intrare într-o poartă are drept efect schimbarea valorii liniei de ieşire a respectivei porţi, apare o întârziere între momentul aplicării schimbării valorii liniei de intrare şi schimbarea valorii liniei de ieşire. Durata acestei întârzieri se numeşte, în general, timpul de comutaţie al porţii. Întârzierea introdusă, în circuit, de-o poartă se măsoară între momentul în care semnalul de intrare, cauzator al schimbării valorii liniei de ieşire, atinge 90% din valoarea finală şi până când linia de ieşire atinge 90% din valoarea finală. Viteza de operare a unui circuit este invers dependentă de întârzierile cele mai mari prin circuitul respectiv.
- Marginea de zgomot este tensiunea externă maximă suprapusă unei valori de intrare normale a unei porţi, şi care nu va cauza o schimbare nedorită a valorii liniei de ieşire pentru respectiva poartă.
- Puterea disipată este puterea consumată de la sursa de alimentare şi disipată de o poartă. Puterea consumată este disipată prin căldura radiată. Din acest motiv puterea disipată trebuie corelată cu temperatura de funcţionare şi cerinţele de răcire ale chipului. Se poate distinge o putere disipată static (atunci când circuitul respectiv nu-şi schimbă starea liniei de ieşire, nu comută) şi o putere disipată dinamic (atunci când circuitul logic îşi schimbă valoarea liniei de ieşire). Puterea consumată static este caracterizabilă prin specificul tehnologiei utilizate în timp ce puterea consumată dinamic este caracterizabilă prin *activitatea logică*. Activitatea logică a unei porţi, este evaluabilă prin frecvenţa schimbării valorii liniei de ieşire a respectivei porţi şi este dependentă de frecvenţa de lucru a circuitului global în care inclusă poarta respectivă. Circuitele bipolare consumă multă putere atât static cât şi dinamic. Puterea consumată static de porţile bipolare este produsă în mod covârşitor prin funcţionarea lor intrinsecă. Circuitele MOS consumă foarte puţină putere statică (cauzată majoritar de curenţii de pierderi) dar, consumă putere dinamică care creşte odată cu frecvenţa la care lucrează.

Logica pozitivă şi logica negativă

Excluzând fenomenele tranzitorii, semnalele binare care apar pe liniile de intrare şi ieşire, ale unei porţi logice, pot avea doar două valori, privite ca tensiuni electrice.

	Valoarea semnalului	
Logica	*H*	*L*
Pozitivă	1	0
Negativă	0	1

Figura 9. Polaritatea logicii în funcţie de atribuirea valorilor 1 şi 0.

Una dintre tensiuni va fi corespunzătoare tensiunii ridicate (notată *H*) de funcţionare şi cealaltă va fi corespunzătoare tensiunii coborâte (notată *L*) de funcţionare.
Este evident că una dintre aceste tensiuni reprezintă 1 logic iar cealaltă 0 logic. Există două atribuiri distincte a tensiunilor electrice pentru valorile logice.

Alegerea valorii 1 corespunzătoare tensiunii ridicate (*H*) defineşte *logica pozitivă* în timp ce alegerea valorii 1 corespunzătoare tensiunii coborâte (*L*) defineşte *logica negativă*.
Foile de catalog ale circuitelor integrate definesc porţile atât în termenii valorilor logice cât şi în termenii valorilor de tensiune ale semnalelor (*H* şi *L*).

Dacă o poartă logică este definită în termenii tensiunilor de semnal, atunci utilizatorul poate să decidă atribuire unei logici pozitive ori negative porţii respective urmând ca aceasta să fie utilizată corespunzător.

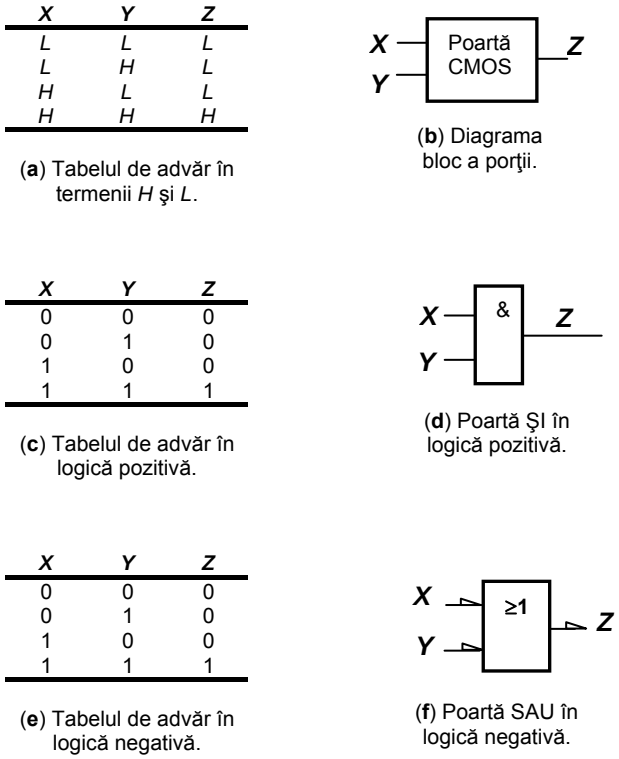

Figura 10. Atribuirea logicii unui circuit specificat în termeni de semnal.

Se consideră poarta logică CMOS din figura 10(b), descrisă în termenii tensiunilor de semnal prin tabelul de adevăr din figura 10(a). Acest tabel specifică comportamentul porţii atunci când nivelul de tensiune H este 5 volţi iar nivelul de tensiune L este 0 volţi.

Tabelul de adevăr din figura 10(c) presupune adoptarea unei logici pozitive, valoarea logică 1 fiind asociată nivelului de tensiune *H* iar valoarea logică 0 având asociat nivelul *L* de tensiune.
Acest tabel corespunde unei funcţii logice ŞI, având simbolul grafic din figura 10(d).

Adoptând o logică negativă, valoarea 1 corespunde nivelului *L* iar valoarea 0 corespunde nivelului *H*, se obţine tabelul de adevăr din figura 9(e) care este identic cu cel al unei funcţii logice SAU.

Simbolul grafic din figura 10(f) corespunde unei porţi SAU în logică negativă. Triunghiurile ataşate liniilor de intrare şi de ieşire, din figura 10(f), sunt *indicatori de polaritate*.
Prezenţa unui indicator de polaritate pe o linie de intrare sa de ieşire dintr-o poartă semnifică asocierea unei logici negative liniilor respective.

Conversia din logică pozitivă în logică negativă şi invers este o operaţie care schimbă valorile 1 în valori 0 şi valorile 0 în valori 1, atât la intrările cât şi la ieşirile unei porţi. Deoarece interschimbarea valorilor 1 şi 0 face parte din calculul dualei, operaţia de convertire a logicilor produce duala funcţiei.
Astfel, schimbarea tuturor intrărilor şi ieşirilor dintr-o polaritate într-alta are drept rezultat convertirea operatorilor ŞI în operatori SAU şi reciproc.