1. NOȚIUNI ŞI DEFINIȚII FUNDAMENTALE

1.1. Fiabilitatea

Una dintre cerințele de bază față de sistemele de calcul moderne este fiabilitatea înaltă a componentelor sale.

Principalele obiective ale fiabilității sunt:

a) studiul defecțiunilor echipamentelor (al cauzelor, al proceselor de apariție şi dezvoltare şi al metodelor de combatere a defecțiunilor);

b) aprecierea cantitativă a comportării echipamentelor în timpul exploatării în condiții normale, ținând seama de influența pe care o exercită asupra acestora factorii interni şi externi;

c) determinarea modelelor şi metodelor de calcul şi prognoză ale fiabilității, pe baza încercărilor de laborator şi a urmăririi comportării în exploatare a echipamentelor;

d) analiza fizică a defecțiunilor;

e) stabilirea metodelor de proiectare, constructive, tehnologice şi de exploatare pentru asigurarea, menținerea şi creşterea fiabilității echipamentelor, dispozitivelor şi elementelor componente;

f) stabilirea metodelor de selectare şi prelucrare a datelor privind analiza fiabilității echipamentelor.

Definită din punct de vedere calitativ, fiabilitatea reprezintă capacitatea unui sistem de a funcționa fără defecțiuni, la parametri acceptabili, în decursul unui anumit interval de timp, în condiții de exploatare bine precizate.

Din punct de vedere cantitativ, fiabilitatea unui sistem reprezintă probabilitatea ca acesta să-şi îndeplinească funcțiile sale cu anumite performanțe şi fără defecțiuni, într-un anumit interval de timp şi în condiții de exploatare specificate.

Durata preconizată, de la momentul punerii în funcțiune si până la prima defecțiune, se numeşte **durata de viata** a sistemului. Pentru a stabili fiabilitatea unui sistem şi durata sa de viață se analizează minuțios defectele ce pot surveni, erorile de funcționare şi se introduc metode profilactice pentru a le preîntâmpina.

1.2. Diagnosticarea tehnică

Diagnosticarea tehnică este o ştiinţă aplicată, care include teoria şi metodele de organizare a proceselor de determinare a stării tehnice a circuitelor numerice la nivel de pastilă de siliciu, plachetă sau sistem.

Diagnosticarea tehnică rezolvă două sarcini de bază:

1) Detectarea defectelor hardware ale circuitelor numerice;

2) Localizarea acestor defecte în vederea înlăturării lor.

Diagnosticarea se poate realiza prin testare. Testarea poate fi efectuată cu ajutorul mijloacelor hard şi soft, încorporate în obiect sau separate de el.

Mijloace hard separate: sisteme şi standuri de testare a circuitelor integrate, generatoare de cuvinte, analizoare de semnătură, testere ş.a.

Mijloace hard încorporate; unităţi de control modulo 2, unităşi d everificare prin comparaţie ş.a.

Mijloace soft: test-programe (programe de lucru), care se păstrează în memoria ROM (BIOS) şi se lansează la fiecare racordare a sistemului la reţea.

1.3. Clasificarea defectelor hardware

Defectul sau **defecţiunea** este o imperfecţiune materială sau fizică cauzată de un eveniment de defectare şi care determină modificarea unei variabile logice sau parametru funcţional faţă de valoarea admisă iniţial.

Efectul apariţiei unui defect este **eroarea**. Un defect nu întotdeauna conduce la o eroare. În acest caz, defectul se consideră latent.

După durata de acţiune, defectele se clasifică în ***defecte tranziente, defecte permanente şi defecte intermitente.***

Defectele tranziente se manifestă printr-o funcţionare greşită de scurtă durată a unei componente, dar nu prin defectarea ei definitivă. Cauzele acestor defecte sunt perturbaţiile: zgomotul, interferenţa electromagnetică a semnalelor. În sistemele de calcul contemporane, cea mai mare parte a erorilor sunt tranziente. Aceste defecte sunt mai greu de depistat din cauza caracterului temporar. Ele pot duce la denaturarea informaţiei în timpul transmiterii, păstrării şi prelucrării datelor.

Un exemplu, ar putea fi o celulă de memorie al cărei conținut este schimbat datorită unei interferențe electromagnetice. Rescrierea ei cu conținutul corect face ca eroarea să dispară.

Defectele permanente se produc la un moment dat şi persistă până când sistemul este reparat. În această categorie includem şi defectele de fabricație a componentelor fizice, factorii nefavorabili de exploatare, îmbătrânirea componentelor. Aceste defecte pot fi înlăturate doar prin efectuarea lucrărilor de reparație sau înlocuirea componentelor.

Defecte intermitente: defectul componentei pendulează între o stare activă si o stare benignă (conexiuni slăbite).

După natura sa, defectele se clasifică în ***defecte logice* şi *defecte parametrice***.

Un **defect logic** se manifestă prin faptul că valoarea logică a unui nod din circuit devine opusă valorii specificate.

Un **defect parametric** se manifestă prin degradarea mărimilor specifice pentru curent, tensiune şi timp.

1.4. Tipuri şi modele de defecte logice

Principalele ***tipuri*** ale defectelor logice sunt următoarele:

- blocaje la 0 sau 1 logic la nivelul nodurilor din circuit;
- scurtcircuite cauzate de punți nedorite, care apar de obicei în faza de execuție a lipiturilor, între conductoarele imprimate ale plachetei;
- inversoare false la intrările sau ieşirile porților logice;
- impulsuri logice eronate;
- impulsuri parazite;
- oscilații.

Modele de defecțiuni.

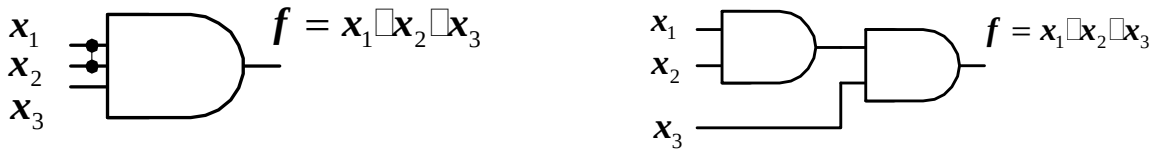
1) modelul de defecțiune **„blocaj la”** este cel mai răspândit şi a apărut odată cu primele familii de circuite logice, RTL (**R**esistor-**T**ransistor-**L**ogic) şi DTL (**D**iode-**T**ransistor-**L**ogic). Acest model se manifestă prin blocarea unuia sau mai multor noduri

de conexiune la valoarea logică 0 - „blocaj la 0” sau la valoarea logică 1 - „blocaj la 1”. Defectele de tip „blocaj la 0” şi „blocaj la 1” se notează ≡0 şi ≡1, respectiv.

Să presupunem că nodul x_1 al porții logice SAU este „blocat la 1”. Indiferent de valoarea semnalului aplicat la intrarea x_1, poarta SAU va percepe acest nod fiind egal cu 1 logic. În acest caz, ieşirea *f* a porții logice va fi egală cu 1 pentru valorile de intrare x_1=0 şi x_2=0. Astfel, setul x_1=0 şi x_2=0 poate fi considerat un test pentru intrarea x1≡1, deoarece există o diferență dintre valoarea ieşirii *f* în absența defectului şi în prezența lui. Pentru x1≡0, test va fi setul x_1=1 şi x_2=0.

2) modelul **„scurtcircuit”.** Defecțiunile de acest tip, care apar în procesul de fabricație al plachetelor, sânt datorate unor punți accidentale ale aliajului de lipire, formate între traseele imprimate alăturate. În cazul circuitelor integrate, acelaşi defect este determinat de imperfecțiuni ale izolației între regiuni ale materialului semiconductor, trasee de metalizare, etc.

Defectele de tip „scurtcircuit” pot fi de două tipuri: defecte cauzate de apariția punților dintre două sau mai multe linii de intrare şi defecte cauzate de apariția punților dintre linia de ieşire şi cea de intrare a circuitului. Nu toate defectele de acest tip duc la erori în circuitele logice. De exemplu, apariția unei punți dintre liniile de intrare x_1 şi x_2 a porții logice ŞI cu trei intrări este echivalentă cu introducerea unei porți logice fictive ŞI, ceea ce nu schimbă funcția logică realizată de poarta dată.



În acelaşi timp, apariția unei punți dintre liniile de intrare x_1 şi x_2 a porții logice SAU cu trei intrări va schimba funcția logică realizată de această poartă.

În cazul apariției unei punți nedorite dintre ieşirea unui circuit logic şi una sau mai multe intrări, acesta trece în regim de oscilare sau se transformă într-un circuit secvențial asincron. Spre exemplu, pentru circuitul prezentat mai jos, scurtcircuitul dintre ieşirea *f* şi intrările x_1 şi x_2 va duce la generarea oscilațiilor de frecvență înaltă, în cazul când $x_1=x_2=x_3=1$ şi $x_4=0$. În acelaşi timp, scurtcircuitul dintre ieşirea *f* şi intrările x_3 şi x_4 va transforma acelaşi circuit combinațional în unul secvențial asincron pentru $x_3=x_4=1$.

(a) (b)

3) modelul de defecțiune **de timp** este necesar în situația, în care comportarea dinamică a schemei este dependentă strict de valorile de timp. În principiu, evitarea utilizării circuitelor logice asincrone conduce la minimizarea riscului de apariție a erorilor datorate parametrilor de timp. Înserarea acestui tip de defecțiune într-un circuit funcțional corect sau simularea devine extrem de dificilă.

2. Concepte de bază ale testării circuitelor numerice

2.1. Teste

În linii generale, un sistem numeric poate fi reprezentat printr-o mulțime de circuite logice: porți logice, bistabile, registre, numărătoare, memorii ROM şi RAM, microprocesoare s. a. Pentru efectuarea testării sistemului pot fi accesate doar unele conexiuni.

Intrările, valorile logice ale cărora pot fi nemijlocit controlate, se numesc **intrări primare**.

Ieşirile, valorile logice ale cărora pot fi nemijlocit observate, se numesc **ieşiri primare**.

Detectarea defectelor poate fi efectuată prin aplicarea unei succesiuni de teste şi observarea valorilor de ieşire într-un circuit logic. Practic, majoritatea testelor sunt generate, presupunând că în circuit are loc un singur defect (*defect singular*).

Testul T_k reprezintă mulțimea semnalelor logice aplicate la intrările primare x_i şi mulțimea reacțiilor de la ieşirile primare y_j pentru un circuit logic corect: $T_k = (x_1, ..., x_n; y_1, ..., y_m)$.

Dacă reacția circuitului este diferită față de cea aşteptată, în acest circuit este prezent un defect singular.

Scopul testării la nivelul porților logice din circuit este verificarea corectitudinii funcționării componentelor şi a interconexiunilor dintre ele. Pentru aceasta e necesar a genera teste, care să detecteze defectele singulare pentru toate nodurile din circuit.

Un circuit logic combinațional (CLC) cu *n* intrări poate fi testat prin aplicarea a 2^n combinații de intrare. Evident, numărul de teste va creste exponențial, odată cu creşterea numărului variabilelor de intrare.

Pentru un circuit logic secvențial (CLS) cu n intrări şi cu m bistabile, numărul total de combinații aplicate la intrare pentru efectuarea unei testări complete este de $2^n \cdot 2^m = 2^{n+m}$. De exemplu, pentru n=20 şi m=40 vom avea 2^{60} teste. La o rată de 10 000 teste pe secundă, timpul total de testare va fi de aproximativ 3,65 milioane ani.

Pentru a efectua testarea unui circuit numeric nu este necesară aplicarea tuturor combinațiilor de intrare, ci doar a celor care permit detectarea defectelor singulare. Mulțimea acestor combinații de intrare şi valorile aşteptate ale funcției de ieşire, se numeşte **mulțimea de teste** pentru circuitul logic.

Mulțimea de teste este completă, dacă ea garantează verificarea prezenței oricărui defect singular din circuitul logic testat.

Mulțimea de teste este minimală, dacă ea conține cel mai mic număr de teste.

2.2. Principii de elaborare a testelor

Un test va detecta un defect doar în cazul când valoarea ieşirii primare în prezența defectului va fi diferită față de valoarea ieşirii primare în absența defectului.

Pentru a detecta un defect într-un circuit, acesta trebuie pentru început *excitat* sau *manifestat*. Procedura constă în aplicarea unei astfel de combinații la intrarea circuitului, care să asigure pe nodul testat valoarea logică opusă valorii defectului. Apoi, defectul trebuie *sensibilizat*. Această procedură constă în propagarea univocă a semnalului de la nodul testat spre ieşirea primară a circuitului.

Vom analiza circuitul logic din figura 1.6 cu nodul G_2 blocat la 1 ($G_2 \equiv 1$). Pentru a asigura manifestarea defectului, la intrările primare ale circuitului trebuie aplicate următoarele valori: $x_1=x_2=x_3=1$, deoarece doar în acest caz valoarea nodului G_2 va primi valoarea opusă defectului analizat ($G_2=0$). Petru a propaga defectul prin poarta G_3, vom considera $x_4=1$. În cazul absenței defectului $F=0$, iar în cazul prezenței defectului $F=1$. Deci, testul pentru $G_2 \equiv 1$ este:

$$T_{G2\equiv1} =(x_1, x_2, x_3, x_4; F)=(1, 1, 1, 1; 0).$$

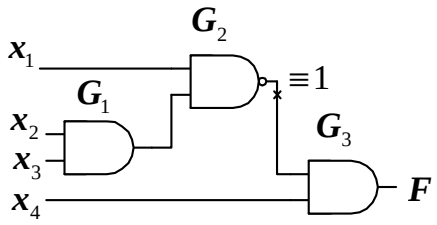


Figura 1.6. CLC cu defectul $G_2 \equiv 1$

2.3. Defecte nedetectabile

Un defect se consideră nedetectabil, dacă este imposibil a atribui asemenea valori la intrările primare ale circuitului pentru a asigura manifestarea lui sau de a propaga univoc valoarea semnalului de la nodul testat spre ieşirea primară a circuitului, pentru sensibilizarea lui. Cu alte cuvinte, un defect este nedetectabil, dacă nu există un test pentru verificarea lui.

Pentru a ilustra cele spuse mai sus, vom analiza circuitul logic din figura 1.7. Pentru a verifica defectul $G_2 \equiv 1$ este necesar a seta nodul G_2 în 0 logic, ceea ce nu este posibil. Deci, defectul $G_2 \equiv 1$ nu poate fi manifestat şi se consideră nedetectabil.

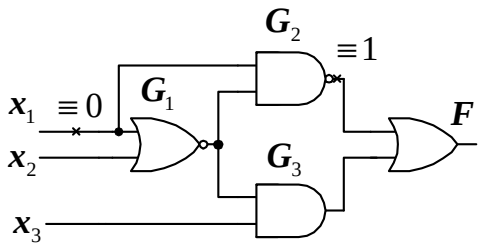


Figura 1.7. Circuit logic cu defecte nedetectabile

Defectul $x_1 \equiv 0$ poate fi manifestat aplicând $x_1=1$ şi $x_2=0$. Dar nu există nici o cale de propagare univocă a acestui defect spre ieşirea primară F, aceasta fiind egală cu 1 logic atât în prezența cât şi în absența defectului. Nodurile x_1 şi G_2 nu pot fi testate, deoarece circuitul este redundant, iar porțile şi conexiunile suplimentare creează contradicții de propagare univocă a semnalelor.

Un circuit logic se numeşte *redundant* dacă el conține conexiuni sau porți, eliminarea cărora nu implică schimbarea funcției logice a circuitului. Orice circuit redundant va avea defecte nedetectabile.

Elaborarea unei mulțimi de teste pentru un circuit se bazează pe presupunerea că doar un singur defect este prezent în circuit în momentul aplicării testului. Astfel, prezența simultană a defectelor detectabile şi nedetectabile nu este în acord cu această presupunere. Mai mult decât atât, prezența defectelor nedetectabile poate denatura procesul de localizare a defectelor detectabile.

2.4. Testabilitatea, controlabilitatea şi observabilitatea

Testabilitatea este o măsură a capabilităților circuitului de a putea fi testat pentru verificarea funcționalității sale. Cele două aspecte majore ale testabilității sunt:

Controlabilitatea - măsura capabilității circuitului de a stabili valoarea logica a unei variabile interne sau a unei ieşiri.

Observabilitatea - măsura capabilității circuitului de a propaga valoarea logica a unei intrări sau a unei variabile interne până la ieşire.

3. TESTAREA CIRCUITELOR LOGICE COMBINAȚIONALE

3.1. Clasificarea metodelor de generare a secvențelor de test

Metodele de generare a secvențelor de test, după principiul utilizat, pot fi clasificate în metode *deterministe* şi metode *probabilistice*.

Metodele deterministe reprezintă anumiți algoritmi formali de generare a secvențelor de test prin analiza funcțiilor logice şi/sau a structurii circuitului. Aceste metode pot fi clasificate în felul următor:

- *metode structurale*: generarea secvențelor de test are loc în urma analizei structurii circuitului;
- *metode analitice*: generarea secvențelor de test are loc în urma analizei funcției logice;
- *metode structural-analitice*: generarea secvențelor de test are loc în urma analizei atât a structurii circuitului logic cât şi a funcției logice.

Exemple de asemenea metode sunt: metoda activării unei căi, metoda algoritmului D, metoda derivatelor booleene, metoda formei echivalente normale ş. a.

Metodele probabilistice se bazează pe generarea aleatorie sau pseudoaleatorie a testelor şi se folosesc în cazul circuitelor mari.

3.2. Metoda activării unei căi

Metoda activării unei căi este una dintre primele abordări de generare a testelor de detectare a defectelor singulare în CLC iredundante. Ea a fost propusă de colaboratorul firmei „IBM” Stiglets şi colaboratorul firmei „Bell Telephone Laboratories” Armstrong.

Această metodă structurală este bazată pe alegerea unei căi de propagare a defectului de la un punct de manifestare spre ieşirea primară a circuitului logic.

Procedura de elaborare a testelor constă din următoarele etape:

1) Se asigură obținerea pe conexiunea defectă a nivelului logic opus presupusei erori (condiția manifestării defectului);

2) Se selectează în mod arbitrar o cale de la locul de manifestare a defectului la una din ieşirile primare ale circuitului;

3) Se activează calea selectată, asigurând astfel condiția de observabilitate a defectului prin propagarea univocă a lui până la ieşirea primară a circuitului (procedura constă în *sensibilizarea* porților logice din calea selectată);

4) Se determină unul sau mai multe teste pentru detectarea defectului analizat, atribuind valori intrărilor primare, astfel încât să se producă semnalele dorite la ieşirile diverselor porți logice din circuit;

5) Dacă nu s-a epuizat mulțimea căilor de propagare a tuturor defectelor analizate spre ieşirea primară a circuitului, se reia cu etapa 2, dacă da, atunci generarea testelor s-a încheiat.

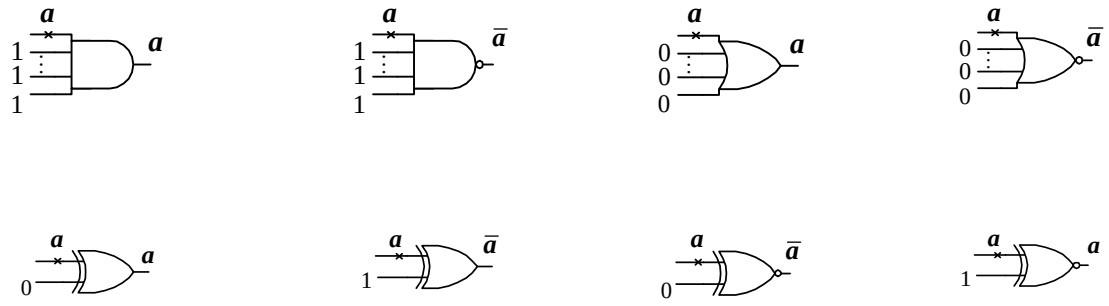
Etapele 1-3 reprezintă *faza de trecere înainte,* iar etapa 4 – *faza de consistență* sau *faza de trecere înapoi.*

Vom analiza mai detaliat etapa 3 a metodei. Pentru a sensibiliza o poartă logică cu o intrare presupusă defectă (*a*), e necesar a atribui celorlalte intrări asemenea valori, încât valoarea ieşirii să depindă doar de valoarea lui *a.* Spre exemplu, în cazul porții logice ŞI cu două intrări valoarea de sensibilizare este egală cu 1 logic. Într-adevăr, pentru *a=1* valoarea de la ieşire va fi egală cu 1 logic, iar pentru *a=0* această valoare va fi egală cu 0 logic. În cazul când vom atribui celei de-a doua intrări valoarea logică 0, ieşirea porții ŞI va fi egală cu 0 atât pentru *a=1* cât şi pentru *a=0*, deci poarta nu este sensibilizată.

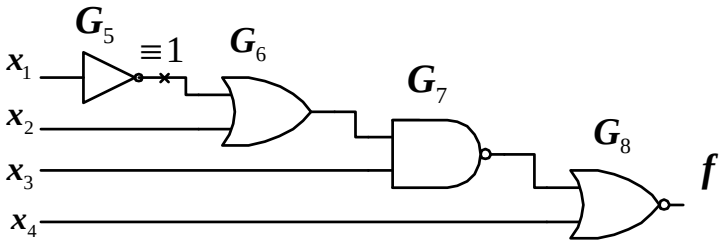
Generalizând cele expuse, se poate uşor observa că pentru porțile logice ŞI şi ŞI-NU valorile de intrare pentru sensibilizare sunt egale cu 1 logic, iar pentru porțile SAU şi SAU-NU – cu 0 logic. În cazul porților logice SAU Exclusiv (XOR) şi

SAU-NU Exclusiv (XNOR) valoarea de intrare pentru sensibilizare poate fi atât 0 cât şi 1 logic.

Porțile logice cu o intrare presupusă defectă (*a*) şi cu valori de intrare pentru sensibilizare:



În continuare vom analiza procedura de activare a unei căi de propagare a defectului $G_5 \equiv 1$.



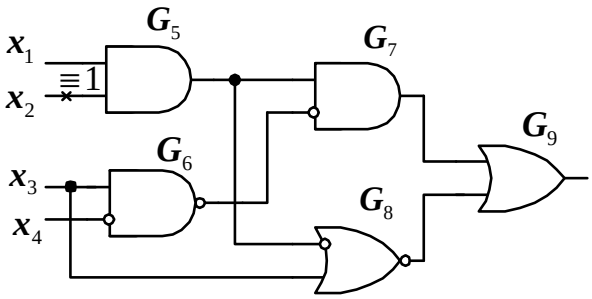
Pentru a asigura condiția de manifestare a defectului $G_5 \equiv 1$ e necesar a obține $G_5=0$. Pentru aceasta vom considera $x_1=1$. Condiția de sensibilizare pentru poarta logică G_6 este $x_2=0$, pentru poarta logică G_7: $x_3=1$ şi pentru poarta logică G_8: $x_4=0$.

În urma activării căii (6, 7, 8) am obținut următorul test:

$$T_{G5\equiv1}=(x_1, x_2, x_3, x_4; f)=(1, 0, 1, 0; 0).$$

Deci, în lipsa defectului $G_5 \equiv 1$, valoarea ieşirii primare *f* va fi egală cu 0 logic (*f*=0), iar în prezența defectului: *f*=1. Astfel, defectul $G_5 \equiv 1$ este detectabil.

Să analizăm mai detaliat procedura de generare a testului pentru detectarea defectului $x_2 \equiv 1$ pe calea (5, 8, 9) pentru CLC arbitrar. Un circuit logic se numeşte *arbitrar* dacă conține ramificări atât ale intrărilor primare cât şi a conexiunilor interne.



Paşii de generare a testului sunt arătați în tabel. Pentru început se asigură condiția de manifestare a defectului $x_2 \equiv 1$ prin setarea intrării primare x_2 în 0 logic (pasul 1). Apoi se sensibilizează pe rând porțile G_5, G_8 şi G_9 pentru a activa calea selectată (paşii 2, 3 şi 4). În continuare se trece la faza de consistență şi anume se asigură asemenea valori pentru x_3 şi x_4, astfel încât să se producă semnalele dorite la ieşirile porților G_7 şi G_6 (paşii 5 şi 6).

Tabelul 2.1. Paşii de generare a unui test

Nr	Intrări primare				Conexiuni interne				Ieşire primară	Comentariu
	x_1	x_2	x_3	x_4	G_5	G_6	G_7	G_8	G_9	
1		**0**								$x_2=0$
2	**1**	0			0					Sensib. G_5
3			**0**		0			0		Sensib. G_8
4							**0**	0	0	Sensib. G_9
5					0	**1**	0			$G_6=1$
6			**0**	*****		1				$x_3=0$
	1	**1**	**0**	*****	0	1	0	0	**0**	Testul obținut

Generarea testului pentru detectarea defectului conexiunii interne $G_6 \equiv 1$

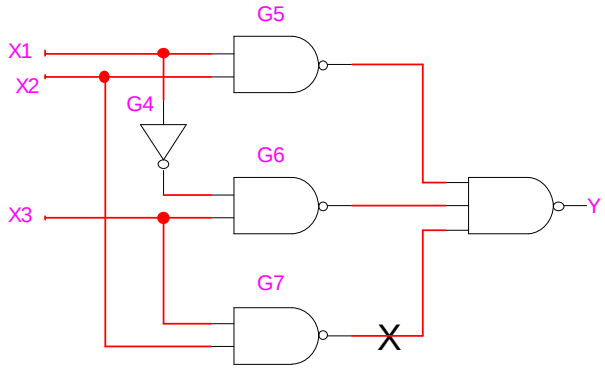
Nr	Intrări primare				Conexiuni interne				Ieşire primară	Comentariu
	x_1	x_2	x_3	x_4	G_5	G_6	G_7	G_8	G_9	
1			1	0		**0**				$G_6=0$
2					1	0	1			Sens. G_7

3							1	0	1	Sens.G_9
4			1		1			0		G_8=0
5	1	1			1					G_5=1
	1	**1**	**1**	**0**	1	0	1	0	**1**	тестul

$T_{G6\equiv1}$=(1, 1, 1, 0; 1).

Metoda activării unei căi nu conduce întotdeauna pentru orice tip de circuit la un test de diagnostic. De exemplu, dacă un circuit conține porți logice redundante, acestea nu pot fi testate.

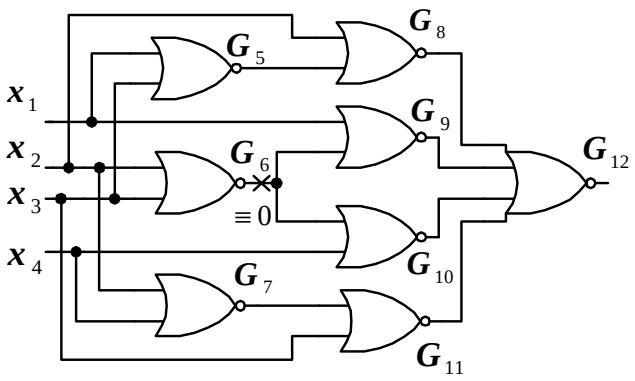
Exemplu: $\boldsymbol{F = x_1x_2 + \bar{x}_1x_3 + x_2x_3}$



Def.	Intr. Pr.	Conexiuni interne	Ieşiri pr.	Calea
	1 2 3	4 5 6 7	8	
$G_7 \equiv 1$	0 1 **1**	**1** 1 **1** 0	1	7, 8

Conflictul apare deoarece nodul G_6 nu poate fi setat în 1.

Generarea testelor prin metoda activării unei căi este o procedură simplă, care nu necesită calcule voluminoase. Datorită acestui fapt ea poate fi utilizată şi la generarea testelor pentru circuite secvențiale. Totodată, există şi un dezavantaj semnificativ: activarea unei singure căi nu conduce întotdeauna la elaborarea unui test, care ar putea fi găsit prin activarea simultană a mai multor căi. Un exemplu clasic, în acest sens, este circuitul propus de Schneider.



Condiția de manifestare a defectului $G_6 \equiv 0$ este: $x_2=x_3=0$. Condiția de observabilitate a defectului prin activarea căii (10, 12) este: $x_4=0$ şi $G_8=G_9=G_{12}=0$. Pentru a obține $G_9=0$ e necesar a seta x_1 în 1 logic, ceea ce va duce la $G_5=0$ şi, deoarece $x_2=0$, vom obține $G_8=1$. Aceasta este în contradicție cu condiția de sensibilizare a porții logice G_8, şi anume $G_8=0$. La fel, este imposibil a activa calea (9, 12). Totuşi, este evident că atribuind $x_1=x_4=0$ vom putea activa simultan două căi de propagare a defectului $G_6 \equiv 0$, iar testul obținut va fi T=(0,0,0,0;1).

3.4. Exemplu de generare a testelor pentru un CLC arbitrar

Fie dată funcția booleană $f=\sum(3,7,16,17,24,25,26,27)$.

Pentru început se efectuează minimizarea acestei funcții utilizând diagrama Karnaugh.

x_4x_5 \ $x_1x_2x_3$	000	001	011	010	110	111	101	100
00					1			1
01					1			1
11	1	1			1			
10					1			

Pentru a realiza un circuit arbitrar, expresia logică obținută după minimizare se transcrie în felul următor:

$$F = x_1x_2\bar{x}_3 + x_1\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_4x_5 = x_1\bar{x}_3\overline{\overline{(x_2 + \bar{x}_4)}} + \bar{x}_2x_4\bar{x}_1x_5 =$$
$$= x_1x_2\overline{(\bar{x}_2x_4)} + \bar{x}_1x_3x_4x_5$$

Circuitul realizat conform acestei expresii logice, adaptat pentru simularea testelor în sistemul de proiectare digitală Logic Works, este prezentat în figura 2.7.

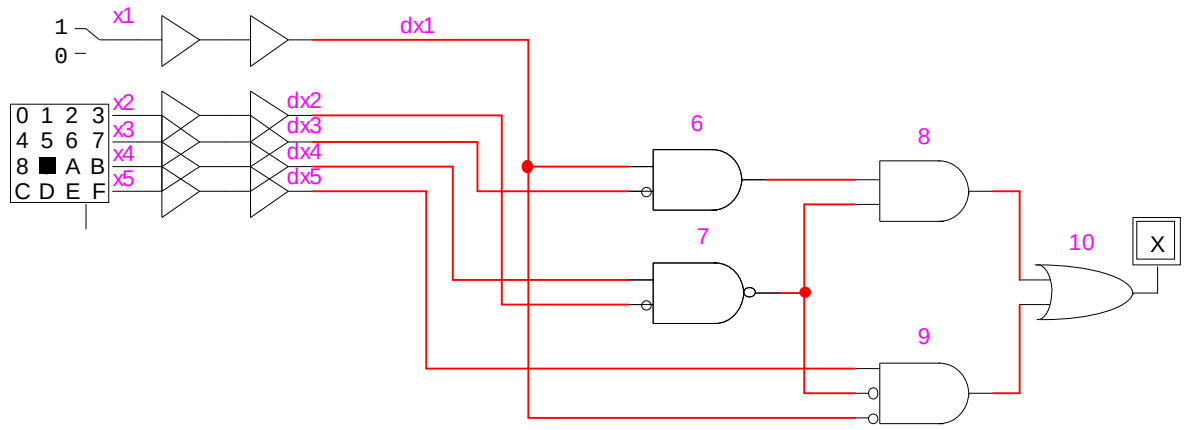


Figura 2.7. CLC arbitrar realizat în Logic Works

Testele elaborate sunt prezentate în tabelul 2.2. La elaborarea testelor au fost folosite următoarele notații:

- ***** - semnifică faptul că nodul respectiv are o valoare indiferentă (0 sau 1 logic);
- **0/*** şi ***/0** - determină trei combinații posibile pe două noduri (01, 10 şi 00).

Un test, în general, poate detecta mai mult decât un singur defect, iar mai multe teste pot detecta acelaşi defect. Astfel, obiectivul major la generarea testelor este minimizarea lor prin determinarea testelor echivalente.

Două teste sunt echivalente dacă:

1) coincid toate valorile definite pentru intrările şi ieşirile primare;

2) coincid toate valorile nedefinite pentru intrările primare (în acest caz în testul rezultant se va alege una din valorile 0 sau 1 pentru valorile nedefinite);

3) valorile definite pentru intrările primare în unul din teste corespund unor valori nedefinite în celălalt test (în acest caz, în testul rezultant se va alege valoarea definită).

Tabelul inițial al testelor

Nr.	Def.	Intrări primare					Conex. Interne				Ieş.pr.	Calea
		x_1	x_2	x_3	x_4	x_5	6	7	8	9	10	
1	$x_1 \equiv 0$	**1**	1/*	0	*/0	*	1	1	1	0	1	6,8,10
2	$x_1 \equiv 0$	**1**	0	*	1	1	*	0	0	0	0	9,10
3	$x_1 \equiv 1$	**0**	1/*	0	*/0	*	0	1	0	0	0	6,8,10
4	$x_1 \equiv 1$	**0**	0	*	1	1	*	0	0	1	1	9,10
5	$x_2 \equiv 0$	1	**1**	0	1	*	1	1	1	0	1	7,8,10
6	$x_2 \equiv 0$	0	**1**	*	1	1	0	1	0	0	0	7,9,10
7	$x_2 \equiv 1$	1	**0**	0	1	*	1	0	0	0	0	7,8,10
8	$x_2 \equiv 1$	0	**0**	*****	1	1	0	0	0	1	1	7,9,10
9	$x_3 \equiv 0$	1	1/*	**1**	*/0	*	0	1	0	0	0	6,8,10
10	$x_3 \equiv 1$	1	1/*	**0**	*/0	*	1	1	1	0	1	6,8,10
11	$x_4 \equiv 0$	1	0	0	**1**	*	1	0	0	0	0	7,8,10
12	$x_4 \equiv 0$	0	0	*	**1**	1	0	0	0	1	1	7,9,10
13	$x_4 \equiv 1$	1	0	0	**0**	*****	1	1	1	0	1	7,8,10
14	$x_4 \equiv 1$	0	0	*	**0**	1	0	1	0	0	0	7,9,10
15	$x_5 \equiv 1$	0	0	*	1	**0**	0	0	0	0	0	7,8,10
16	$x_5 \equiv 0$	0	0	*	1	**1**	0	0	0	1	1	7,9,10

Testele minimizate

Nr.	Nr. testelor inițiale	Teste cu toate valorile definite	Defectele detectate
		x_1,x_2,x_3,x_4,x_5;10	
1	1,5,10	1,1,0,1,0,1	$x_1 \equiv 0$, $x_2 \equiv 0$, $x_3 \equiv 1$
2	2,7,11	1,0,0,1,1,0	$x_1 \equiv 0$, $x_2 \equiv 1$, $x_4 \equiv 0$
3	3,6	0,1,0,1,1,0	$x_1 \equiv 0$, $x_4 \equiv 0$, $x_5 \equiv 0$

4	4,8,12,16	0,0,0, 1,1,1	$x_1 \equiv 1$, $x_2 \equiv 1$, $x_4 \equiv 0$, $x_5 \equiv 0$
5	9	1,1,1,0,0,0	$x_3 \equiv 0$
6	13	1,0,0,0,0,1	$x_4 \equiv 1$
7	14	0,0,0,0,1,0	$x_4 \equiv 1$
8	16	0,0,0,1,0,0	$x_5 \equiv 1$

2.5. Algoritmul *D*

Algoritmul ***D*** este o metodă structurală de elaborare a testelor, care conduce la obținerea unui test de diagnosticare a unui defect în termenii intrărilor şi ieşirii porții defecte, generând simultan toate căile posibile de propagare a defectului la toate ieşirile primare ale circuitului. La fiecare pas al algoritmului se verifică convergența căilor, renunțându-se la caile care nu sunt divergente. În final, se urmăresc până la intrările primare toate căile de propagare generate, căutându-se în mod automat valorile logice ale semnalelor de intrare, care evidențiază manifestarea defectului la ieşirile primare.

Algoritmul ***D*** utilizează noțiunile de *cub singular* sau *cub de definiție* şi *cub* ***D*** pentru descrierea porților logice din circuit.

2.5.1. Cuburi singulare

Funcționarea porților logice poate fi descrisă prin tabele de adevăr. Dacă în aceste tabele valorile de intrare, ce nu influențează valoarea semnalului de ieşire, se vor considera valori indiferente (notate prin simbolul *), vom obține un nou tabel. Acest tabel va fi format din cuburi singulare, totalitatea cărora va forma *acoperirea singulară a funcției logice,* care descrie comportamentul porții respective.

De exemplu, pentru determinarea cuburilor singulare a porții logice ŞI cu două intrări se ține seama de următoarele considerente:

- este de ajuns ca o singură intrare să fie egală cu 0 pentru a avea la ieşire valoarea 0;

- este necesar ca ambele intrări să fie egale cu 1 pentru a avea la ieşire valoarea 1.

Obținerea cuburilor singulare din tabelul de adevăr al porții logice ŞI cu două intrări :

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$\Rightarrow$

x_1	x_2	y	
0	*	0	CS_1
*	0	0	CS_2 $\leftarrow$ cuburi singulare
1	1	1	CS_3

Elementele cubului singular sunt denumite coordonate sau noduri. De exemplu, cubul singular 0*0 are nodurile 000 şi 010 cu coordonatele x1, x2 şi y.

Cuburile singulare ale porților logice ŞI, SAU, ŞI-NU şi SAU-NU cu două intrări:

	ŞI			ŞI-NU			SAU			SAU-NU		
x_1	0	*	1	0	*	1	1	*	0	1	*	0
x_2	*	0	1	*	0	1	*	1	0	*	1	0
y	0	0	1	1	1	0	1	1	0	0	0	1

2.5.2. Cuburi D de propagare

Pentru a obține un cub $\boldsymbol{D}$ pentru o poartă logică se intersectează două cuburi singulare cu valori diferite ale ieşirii conform următoarelor reguli:

$$
\begin{aligned}
&0 \cap 0 = 0 \cap * = * \cap 0 = 0 \\
&1 \cap 1 = 1 \cap * = * \cap 1 = 1 \\
&* \cap * = * \\
&1 \cap 0 = \boldsymbol{D} \\
&0 \cap 1 = \bar{\boldsymbol{D}}
\end{aligned}
\qquad (2.1)
$$

De exemplu, pentru a obține cuburile $\boldsymbol{D}$ ale porții logice ŞI (vezi figura 2.9) putem intersecta CS_3 cu CS_1 şi CS_3 cu CS_2:

$$
\begin{array}{r} \boldsymbol{CS}_3 = 1\,1\,1 \\ \boldsymbol{CS}_1 = 0\,*\,0 \\ \boldsymbol{CS}_3 \cap \boldsymbol{CS}_1 = \boldsymbol{D}1\boldsymbol{D} \end{array}
\qquad
\begin{array}{r} \boldsymbol{CS}_3 = 1\,1\,1 \\ \boldsymbol{CS}_2 = *\,0\,0 \\ \boldsymbol{CS}_3 \cap \boldsymbol{CS}_2 = 1\boldsymbol{DD} \end{array}
$$

Cubul $\boldsymbol{D}$ exprimă dependența semnalului de la ieşirea porții logice față de cea aplicată la una din intrările ei. $\boldsymbol{D}$ poate avea două valori: 0 sau 1. De exemplu, cubul

$D1D$ are nodurile 010 şi 111. Aceasta mărturiseşte despre faptul că, dacă poarta e corectă, atunci semnalul de la ieşirea *y* este determinat doar de semnalul de la intrarea x_1, x_2 fiind fixat în 1 logic. Astfel, apariția defectului $x_1 \equiv 1$ ($x_1 \equiv 0$) este detectată la ieşirea porții respective.

Dacă semnalul oricărei coordonate a cubului D, notat prin D, are valoarea 1(0), atunci toate celelalte coordonate, notate prin D, vor fi egale cu 1(0), iar semnalele coordonatelor, notate prin $\bar{D}$, vor fi egale cu 0(1).

Se deosebesc *cuburi* D *singulare* şi *cuburi* D *multiple*. În cuburile D singulare doar o singură intrare e notată prin D sau $\bar{D}$. În cuburile D multiple două sau mai multe intrări sunt notate prin simbolurile D sau $\bar{D}$. Necesitatea utilizării cuburilor D multiple se explică prin faptul că la intrările unei porți logice se pot întâlni mai multe semnale D, atunci când în circuit sunt prezente ramificații.

Cuburile D ale porților logice se mai numesc *cuburi* D *de propagare* (CDP). CDP ale porților logice ŞI, SAU, ŞI-NU şi SAU-NU cu două intrări.

	ŞI				ŞI-NU				SAU				SAU-NU			
x_1	D	1	D	D	D	1	D	D	D	0	D	D	D	0	D	D
x_2	1	D	D	$\bar{D}$	1	D	D	$\bar{D}$	0	D	D	$\bar{D}$	0	D	D	$\bar{D}$
y	D	D	D	0	$\bar{D}$	$\bar{D}$	$\bar{D}$	1	D	D	D	1	$\bar{D}$	$\bar{D}$	$\bar{D}$	0

2.5.3. Cuburi D ale defectelor (CDD)

Cuburile D ale defectelor permit exprimarea testelor în termenii intrării şi ieşirii porții defecte. Ele se utilizează atunci când este necesar a testa nodurile interne ale circuitului. În CDD simbolul D e interpretat ca semnal 1 logic pentru starea corectă şi ca semnal 0 logic pentru starea defectă. Simbolul $\bar{D}$ e interpretat invers – 0 logic pentru starea corectă şi 1 logic pentru starea defectă.

CDD pentru un nod blocat la 0 poate fi obținut prin intersecția fiecărui cub singular cu valoarea ieşirii egală cu 1 logic pentru poarta corectă cu fiecare cub singular al cărei ieşire este egală cu 0 logic pentru o poartă defectă. În mod similar, CDD pentru un nod blocat la 1 poate fi obținut prin intersecția fiecărui cub singular

cu valoarea ieşirii egală cu 0 logic pentru poarta corectă cu fiecare cub singular al cărei ieşire este egală cu 1 logic pentru o poartă defectă.

CDD ale porților logice ŞI, SAU, ŞI-NU şi SAU-NU.

Defecte	ŞI			ŞI-NU			SAU			SAU-NU		
	x_1	x_2	y	x_1	x_2	y	x_1	x_2	y	x_1	x_2	y
≡ 0	**1**	**1**	D	**0**	*	D	**1**	*	D	**0**	**0**	D
				*	**0**	D	*	**1**	D			
≡ 1	**0**	*	$\bar{D}$	**0**	**0**	$\bar{D}$	**1**	**1**	$\bar{D}$	**1**	*	$\bar{D}$
	*	**0**	$\bar{D}$							*	**1**	$\bar{D}$

2.5.4. Intersecția D

Generarea unei căi de propagare a defectului spre ieşirea primară a circuitului este realizată prin intermediul intersecției D.

Fie date două cuburi D:

$A = (a_1, a_2, \ldots, a_n)$

$B = (b_1, b_2, \ldots, b_n)$,

unde $a_i, b_i \in \{1, 0, *, D, \bar{D}\}$, $i = \overline{1,n}$.

Intersecția D se efectuează doar pentru coordonate identice conform următoarelor reguli:

1) $* \bigcap_D a_i = a_i$

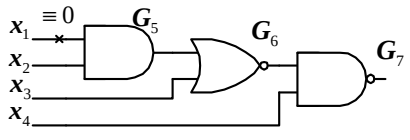
$* \bigcap_D b_i = b_i$

2) Dacă $a_i \neq *$ şi $b_i \neq *$, atunci

$$a_i \bigcap_D b_i = \begin{Bmatrix} a_i, & pentru & a_i = b_i \\ \varnothing, & pentru & a_i \neq b_i \end{Bmatrix}.$$

Intersecția D reprezintă o formă de descriere a propagării defectului de la nodul analizat spre ieşirea primară a circuitului.

Să analizăm o cale arbitrară, selectată pentru propagarea defectului $x_1 \equiv 0$



Utilizând intersecția $\boldsymbol{D}$ dintre cuburile $\boldsymbol{D}$ ale porților logice din calea respectivă vom obține cubul $\boldsymbol{D}$ al circuitului .

Formarea cubului $\boldsymbol{D}$ al circuitului

Cuburi	Coordonate						
	x_1	x_2	x_3	x_4	G_5	G_6	G_7
CDP5	$\boldsymbol{D}$	1	*	*	$\boldsymbol{D}$	*	*
CDP6	*	*	0	*	$\boldsymbol{D}$	$\bar{\boldsymbol{D}}$	*
CDP7	*	*	*	1	*	$\bar{\boldsymbol{D}}$	$\boldsymbol{D}$
CD al circuitului	$\boldsymbol{D}$	1	0	1	$\boldsymbol{D}$	$\bar{\boldsymbol{D}}$	$\boldsymbol{D}$

Cubul $\boldsymbol{D}$ obținut ($\boldsymbol{D}$, 1, 0, 1, $\boldsymbol{D}$, $\bar{\boldsymbol{D}}$, $\boldsymbol{D}$) verifică conexiunile x_1, G_5, G_6 în baza ieşirii G_7, fiind fixate valorile semnalelor intrărilor primare x_2, x_3, x_4.

Deoarece $\boldsymbol{D}$ poate lua două valori logice – 0 şi 1, cubul $\boldsymbol{D}$ al circuitului se foloseşte la detectarea a două defecte: $x_1 \equiv 0$ şi $x_1 \equiv 1$.

Pentru $x_1 \equiv 0$, considerăm $\boldsymbol{D}=1$ şi obținem testul:

$$T_{x1\equiv 0}=(x_1, x_2, x_3, x_4; G_7)=(1, 1, 0, 1; 1).$$

Pentru $x_1 \equiv 1$, considerăm $\boldsymbol{D}=0$ şi obținem testul:

$$T_{x1\equiv 1}=(x_1, x_2, x_3, x_4; G_7)=(0, 1, 0, 1; 0).$$

2.5.5. Etapele algoritmului $\boldsymbol{D}$

Pentru început se determină cuburile singulare (CS) şi cuburile $\boldsymbol{D}$ de propagare (CDP) a fiecărei porți logice din circuitul logic combinațional (CLC).

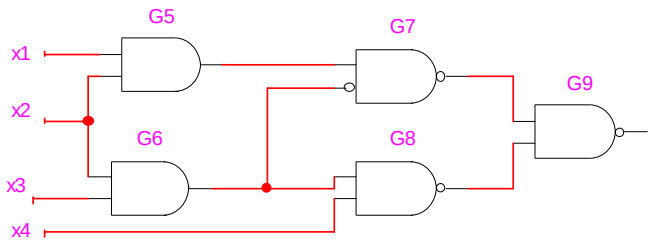
Generarea testelor prin metoda algoritmului $\boldsymbol{D}$ constă din următoarele etape:

1) Construirea cubului $\boldsymbol{D}$ al defectului (CDD);
2) Propagarea defectului prin efectuarea intersecției CDD cu CDP a porților logice de pe calea aleasă până la ieşirea primară;
3) Verificarea consistenței (trecerea înapoi) prin intersecția cubului rezultant cu CS ale porților logice, care nu au fost utilizate la prima intersecție;

4) Repetarea etapelor 1-3 până când se obțin testele pentru toate defectele analizate pe toate căile singulare şi multiple;
5) Minimizarea testelor.

La etapa de verificare a consistenței se pot obține elemente vide pe anumite coordonate. În acest caz întreaga intersecție se consideră vidă şi se renunță la calea dată, deoarece ea este inconsistentă.

Să analizăm mai detaliat primele trei etape de generare a testelor pentru circuitul logic prezentat.



Testul pentru detectarea defectelor intrării primare $x_1 \equiv 0$ şi $x_1 \equiv 1$ pe calea (5,7,9):

Et.	Explicații	Cub	Coordonate		
			1 2 3 4	5 6 7 8	9
Et. 1	CDD	C_1	D * * *	* * * *	*
Et. 2	Intersectăm C_1 cu CDP al G_5	C_2	D 1* *	D * * *	*
	Intersectăm C_2 cu CDP al G_7	C_3	D 1 * *	D 0 $\bar{D}$ *	*
	Intersectăm C_3 cu CDP al G_9	C_4	D 1 * *	D 0 $\bar{D}$ 1	D
Et. 3	Intersectăm C_4 cu CS al G_8	C_5	D 1 * *	D 0 $\bar{D}$ 1	D
	Intersectăm C_5 cu CS al G_6	C_6	D 1 0 *	D 0 $\bar{D}$ 1	D

Din cubul rezultant C_6 se obțin două teste:

$T_{x2\equiv0}=(x_1,x_2,x_3,x_4;8)=(1,1,0,*;1)$ şi

$T_{x2\equiv1}=(x_1,x_2,x_3,x_4;8)=(0,1,0,*;0)$.

Testul pentru detectarea defectului conexiunii interne $G_6 \equiv 0$ pe calea (8,9). Deoarece în CDD ale porților logice simbolul D poate lua doar valoarea 1, iar $\bar{D}$ doar valoarea 0, nu este posibil a detecta printr-un singur test defectele $G_6 \equiv 0$ şi $G_6 \equiv 1$, fiind necesare două CDD diferite.

Et.	Explicații	Cub	Coordonate		
			1 2 3 4	5 6 7 8	9
Et. 1	CDD	C_1	* 1 1 *	* D * *	*
Et. 2	Intersectăm C_1 cu CDP al G_8	C_2	* 1 1 1	* D * $\bar{D}$	*
	Intersectăm C_2 cu CDP al G_9	C_3	* 1 1 1	0 D 1 $\bar{D}$	D
Et. 3	Intersectăm C_3 cu CS al G_7	C_4	* 1 1 1	0 D 1 $\bar{D}$	D
	Intersectăm C_4 cu CS al G_5	C_5	0 1 1 1	0 D 1 $\bar{D}$	D

Din cubul rezultant C_4 se obține următorul test:

$T_{G5\equiv 0}$=(x_1,x_2,x_3,x_4;8)= (0,1,1,1;1).

Pentru a obține testul detectării defectului $G_6 \equiv 1$ vom considera CDD=(x_2,x_3,6)=(0,*, D) sau CDD=(x_2,x_3,6)=(*,0, D).

Algoritmul D este o metodă bine formalizată şi poate fi uşor programată, ceea ce permite automatizarea procesului generării testelor. În prezent există mai multe modificații ale algoritmului D, care permit accelerarea procesului de generare a testelor. De exemplu, procedura PODEM, în care fiecare pas de propagare este urmat de paşii de trecere înapoi până la intrările primare ale circuitului, depistându-se mai repede căile inconsistente. Pe lângă aceasta, spre deosebire de metoda activării unei căi, algoritmul D garantează obținerea testului, dacă acesta există, datorită faptului că sunt analizate toate căile de propagare a defectului, inclusiv şi cele multiple.

Exemplu. Vom obține testele pentru intrările primare pe toate căile posibile, inclusiv şi cele multiple.

Nr	Def.	Intrări primare	Conexiuni interne	Ieş.	Calea
		x1 x2 x3 x4	5 6 7 8	9	
1	x_1	D 1 0 *	D 0 $\bar{D}$ 1	D	5,7,9
2	x_2	1 D 0 *	D 0 $\bar{D}$ 1	D	5,7,9
3	x_2	0 D 1 1	0 D 1 $\bar{D}$	D	6,8,9
4	x_2	1 D 1 1	D D 1 $\bar{D}$	D	5-6,7-8,9
5	x_2	D 1 1 1 1 $1 \cap D = \varnothing$	1 D Cale inconsistentă		6,7,9
6	x_3	1 1 D 0	1 D D 1	$\bar{D}$	6,7,9
7	x_3	0 1 D 1	0 D 1 $\bar{D}$	D	6,8,9
8	x_3	1 D 1	1 D D $\bar{D}$ Cale convergentă	1	5,7,9
9	x_4	* 1 1 D	* 1 1 $\bar{D}$	D	8,9

Din cele 7 cuburi rămase, pot fi obținute 14 teste, înlocuind simbolul D cu valorile 0 şi 1, în dependență de defectul analizat .

Tabelul inițial al testelor

Nr.	Def	Testul
		x_1, x_2, x_3, x_4; 9
1	$x_1 \equiv 0$	1 1 0 * 1
2	$x_1 \equiv 1$	0 1 0 * 0
3	$x_2 \equiv 0$	1 1 0 * 1
4	$x_2 \equiv 1$	1 0 0 * 0
5	$x_2 \equiv 0$	0 1 1 1 1
6	$x_2 \equiv 1$	0 0 1 1 0
7	$x_2 \equiv 0$	1 1 1 1 1

Nr.	Def	Testul
		x_1, x_2, x_3, x_4; 9
8	$x_2 \equiv 1$	1 0 1 1 0
9	$x_3 \equiv 0$	1 1 1 0 0
10	$x_3 \equiv 1$	1 1 0 0 1
11	$x_3 \equiv 0$	0 1 1 1 1
12	$x_3 \equiv 1$	0 1 0 1 0
13	$x_4 \equiv 0$	* 1 1 1 1
14	$x_4 \equiv 1$	* 1 1 0 0

Tabelul testelor minimizate

Nr	Nr. t.inițiale	x_1, x_2, x_3, x_4; 9	Defectele detectate
1	1,3,10	1 1 0 0 1	$x_1 \equiv 0$, $x_2 \equiv 0$, $x_3 \equiv 1$
2	2,12	0 1 0 1 0	$x_1 \equiv 1$, $x_3 \equiv 1$
3	3,10	1 1 0 0 1	$x_2 \equiv 0$, $x_3 \equiv 1$
4	4	1 0 0 0 0	$x_2 \equiv 1$
5	5,11,13	0 1 1 1 1	$x_2 \equiv 0$, $x_3 \equiv 0$, $x_4 \equiv 0$
6	6	0 0 1 1 0	$x_2 \equiv 1$
7	8	1 0 1 1 0	$x_2 \equiv 1$
8	9,14	1 1 1 0 0	$x_3 \equiv 0$, $x_4 \equiv 1$

3. METODA FORMEI ECHIVALENTE NORMALE (FEN)

Metoda FEN este o metodă structural-analitica propusă de Armstrong în 1966. Ea constă în utilizarea unei forme analitice a funcției logice care conține anumite atribute ce permit stabilirea unei corespondențe univoce între circuit şi FEN.

FEN permite elaborarea testelor pentru CLC iredundante ce constau din porți logice ŞI, SAU, ŞI-NU, SAU-NU, XOR şi XNOR.

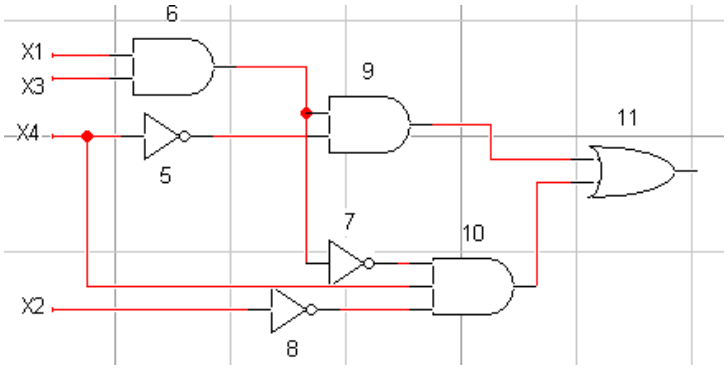
Pentru un CLC cu o ieşire, FEN se obține în felul următor:

1. Fiecărei porți logice din circuit i se atribuie un număr.

2. Se scrie expresia logică a CLC, începând de la ieşirea primară. Fiecărei porți Gi îi va corespunde o pereche de paranteze indexate cu simbolul i: Gi $\rightarrow$ ( )i. Toate variabilele din expresia logică vor corespunde intrărilor primare.

3. Expresia se modifică în forma sumei canonice. La omiterea unei perechi de paranteze indicele aderent se atribuie fiecăreio variabile din paranteze. Termenii redundanți nu se omit.

Exemplu de obținere a FEN:



$$\boldsymbol{F} = \quad (9+10)_{11} =$$

$$((6*5)_9 + (7*4*8)_{10})_{11} =$$

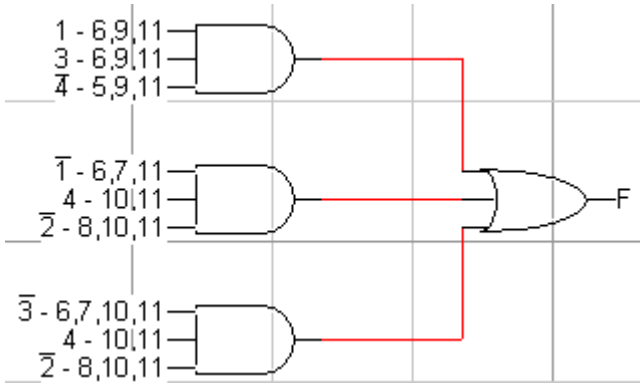
$$= (((1*3)_6 * \overline{4}_5)_9 + (\overline{6}_7 * 4 * \overline{2}_8)_{10})_{11} =$$

$$= ((1_6 * 3_6 * \overline{4}_5)_9 + ((\overline{1*3})_6)_7 * 4 * \overline{2}_8)_{10})_{11} =$$

$$= (1_{6,9} * 3_{6,9} * \overline{4}_{5,9}) + (\overline{1}_{6,7} + \overline{3}_{6,7}) * 4 * \overline{2}_8)_{10})_{11} =$$

$$= 1_{6,9,11} * 3_{6,9,11} * \overline{4}_{5,9,11} + \overline{1}_{6,7,11} * 4_{10,11} * \overline{2}_{8,10,11} + \overline{3}_{6,7,10,11} * 4_{10,11} * \overline{2}_{8,10,11}$$

FEN permite de a obține un circuit ipotetic care are doar două nivele, indiferent de circuitul inițial. Acest fapt simplifică procedura de obținere a testelor.



Proprietățile FEN

1. FEN este o disjuncție de conjuncții, şi anume forma normală funcției booleene.

2. Variabila şi indicele ei în expresia FEN indică calea de la intrarea respectivă a circuitului spre ieşirea primară.

3. FEN poate conține termeni şi variabile redundante, chiar dacă circuitul inițial este iredundant.

4. Dacă calea legată de o anumită variabilă conține un număr impar de inversări , atunci variabila respectivă intră în FEN inversată.

Generarea testelor

Procedura de generare a testului:

1. Se atribuie variabilei selectate valoarea logică opusă defectului (condiția de manifestare a defectului).

2. În termenul ales toate celelalte variabile primesc valoarea 1 logic.

3. În restul termenilor cel puțin o variabilă va primi valoarea 0 logic pentru a asigura transportarea valorii analizate în mod univoc spre ieşire.

Pentru a reduce numărul de teste se iau în considerație 2 reguli:

1. În FEN se alege termenul care conține cel mai mic număr de variabile.

2. În termenul selectat se verifică în primul rând variabila care apare de cele mai multe ori în ceilalți termeni.

Vom elabora testele pentru exemplul analizat. Toți termenii au un număr egal de variabile. Variabila x4 se conține în toți termenii. Deci ordinea de elaborare a testelor va fi: x4, x1, x2, x3.

Tabelul testelor

Def.	FEN	Testul
	$1*3*\bar{4} + \bar{1}*4*\bar{2} + \bar{3}*4*\bar{2}$	1 2 3 4 ; 11
x4 ≡ 0	0 1 0 1 **1** 1 0 1 1	0 0 1 1 1
x4 ≡ 1	0 1 1 1 **0** 1 0 0 1	0 0 1 0 0
x1 ≡ 0	**1** 1 1 0 0 * 0 0 *	1 * 1 0 1
x1 ≡ 1	**0** 1 1 1 0 * 0 1 1	0 * 1 0 0
x2 ≡ 0	0 1 0 1 1 **0** 0 1 0	0 1 1 1 0
x2 ≡ 1	0 1 0 1 1 **1** 0 1 1	0 0 1 1 1
x3 ≡ 0	1 **1** 1 0 0 * 0 0 *	1 * 1 0 1
x3 ≡ 1	1 **0** 1 0 0 * 1 0 *	1 * 0 0 0

Metoda este eficace la elaborarea testelor cu o singură ieşire şi cu un număr mic de porți logice.

METODA DIFERENȚELOR (DERIVATELOR) BOOLEENE

Este o metodă analitică bazată pe activarea unei căi de propagare a defectului spre ieşirea primară.

Fie dată o funcție booleană de n variabile f(x_1, x_2, …, x_n). Diferența booleană (DB) a funcției *f* în raport cu variabila x_i este funcția:

$$\frac{df}{dx_i} = f(x_1, x_2, ..., x_i, ..., x_n) \oplus f(x_1, x_2, ..., \overline{x_i}, ..., x_n) \quad (1)$$

DB poate fi scrisă şi sub forma:

$$\frac{df}{dx_i} = f(x_1, x_2, ..., 1, ..., x_n) \oplus f(x_1, x_2, ..., 0, ..., x_n) \quad (2)$$

Deoarece DB este rezultatul operației XOR, rezultă că $\frac{df}{dx_i} = 1$ dacă şi numai dacă orice modificare a variabilei x_i conduce la modificarea valorii funcției *f*. Atunci când $\frac{df}{dx_i} = 0$, modificarea variabilei x_i nu va conduce la modificarea valorii funcției *f*.

Această proprietate a derivatei booleene este folosită pentru determinarea condiției de observabilitate (de propagare) a defectului către ieşire.

Condiția de manifestare a defectului **a** pentru un anumit nod din circuit este asigurată prin atribuirea valorii opuse celei implicate în defect, şi anume $\overline{a}$.

Unind aceste două condiții într-o ecuație, vom obține $\overline{a} \cdot \frac{dF}{da} = 1$. Rezolvarea acestei ecuații ne permite să obținem testele pentru detectarea unui defect **a** de tip „blocaj în 0” sau „blocaj în 1”.

La calculul DB se pot utiliza următoarele egalități:

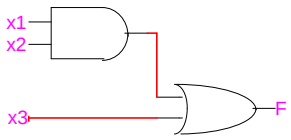
1. $x \oplus \overline{x} = 1$
2. $x \oplus x = 0$
3. $x \oplus 1 = \overline{x}$
4. $x \oplus 0 = x$
5. $\overline{x \oplus y} = xy + \overline{x}\,\overline{y}$
6. $x \oplus y \oplus xy = x + y$

Proprietățile DB:

1. $\frac{d\overline{F}}{dx} = \frac{dF}{dx}$
2. $\frac{dF}{d\overline{x}} = \frac{dF}{dx}$
3. $\frac{dF}{dx_i}[\frac{dF}{dx_s}] = \frac{d}{dx_j}[\frac{dF}{dx_i}]$
4. $\frac{d}{dx}[F \cdot G] = F\frac{dG}{dx} \oplus G\frac{dF}{dx} \oplus \frac{dF}{dx} \cdot \frac{dG}{dx}$
5. $\frac{d}{dx}[F + G] = \overline{F}\frac{dG}{dx} \oplus \overline{G}\frac{dF}{dx} \oplus \frac{dF}{dx} \cdot \frac{dG}{dx}$
6. $\frac{d}{dx}[F \oplus G] = \frac{dF}{dx} \oplus \frac{dG}{dx}$
7. $\frac{dF}{dx_i} = 0$ dacă F nu depinde de x_i
8. $\frac{dF}{dx_i} = 1$ dacă F=x_i
9. $\frac{d}{dx_i}[F \cdot G] = F\frac{dG}{dx_i}$ dacă F nu depinde de x_i
10. $\frac{d}{dx_i}[F + G] = \overline{F}\frac{dG}{dx_i}$ dacă F nu depinde de x_i

Exemplu:

Fie dată funcția booleană F=∑(1,3,5,6,7)=$x_1x_2 + x_3$



Vom determina derivata booleană față de variabila x_1.

$$\frac{dF}{dx_1} = \frac{d(x_1x_2 + x_3)}{dx_1}^{pr.10} = \overline{x}_3\frac{d(x_1x_2)}{dx_1}^{pr.9} = x_2\overline{x}_3\frac{dx_1}{dx_1}^{pr.8} = x_2\overline{x}_3 :$$

Vom determina testul pentru $x_1 \equiv 0$. Deci defectul **a,** în acest caz, va fi $x_1 = 1$, iar condiția de manifestare a defectului $\overline{a}$ va fi $x_1 = 1$. Înlocuim în ecuația $\overline{a} \cdot \frac{dF}{da} = 1$ valorile pentru $\overline{a}$ şi $\frac{dF}{da}$ şi obținem:

$$x_1 \frac{dF}{dx1} = x_1 x_2 \bar{x}_3 = 1$$

Rezolvând ecuația obținem: $x_1 = 1, x_2 = 1, x_3 = 0$.

Înlocuind valorile obținute pentri variabilele de intrare în expresia logică a funcției obținem: $f = x_1 x_2 + x_3 = 1 \cdot 1 + 0 = 1$

Testul rezultat este: (1,1,0;1).

Vom determina testul pentru $x1 \equiv 1$. Deci defectul **a,** în acest caz, va fi $x1 \equiv 1$, iar condiția de manifestare a defectului $\bar{a}$ va fi $x_1 = 0$. Înlocuim în ecuația $\bar{a} \cdot \frac{dF}{da} = 1$ valorile pentru $\bar{a}$ şi $\frac{dF}{da}$ şi obținem:

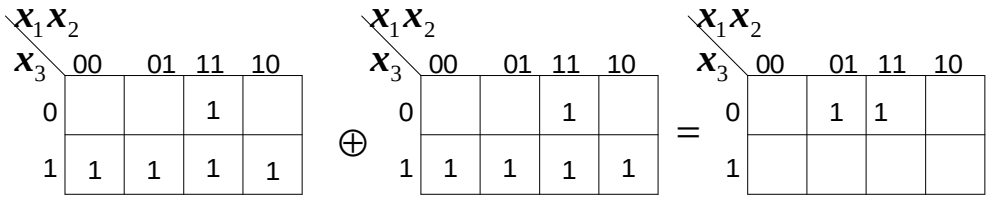
$$\bar{x}_1 \frac{dF}{dx1} = x_1 x_2 \bar{x}_3 = 1$$

Rezolvând ecuația obținem: $x_1 = 0, x_2 = 1, x_3 = 0$.

Înlocuind valorile obținute pentri variabilele de intrare în expresia logică a funcției obținem: $f = x_1 x_2 + x_3 = 0 \cdot 1 + 0 = 0$

Testul rezultat este: (0,1,0;0).

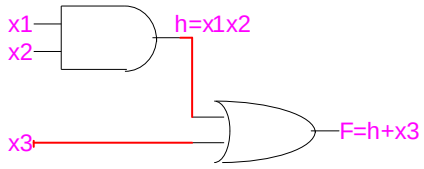
Derivata booleană poate fi calculată şi utilizând metoda grafică bazată pe diagramele Karnaugh. În acest caz $F(x_1,...x_i,...x_n)$, $F(x_1,...\bar{x}_i,...x_n)$ şi $\frac{dF}{dx_i}$ se reprezintă prin câte o diagramă Karnaugh.



$F(x_1, x_2, x_3)$ $F(\bar{x}_1, x_2, x_3)$ $\frac{dF}{dx_1} = x_2 \bar{x}_3$

Generarea testelor prin metoda diferențelor booleene pentru conexiuni interne

La generarea testelor pentru conexiuni interne defecte, se porneşte de la faptul ca funcția de ieşire poate fi scrisă şi sub forma: F(x,h), unde h=h(x)



Aflăm derivata booleană față de nodul h: $\frac{dF(x,h)}{dh} = \frac{dF(h+x_3)}{dh} = \bar{x}_3 \frac{dF(h)}{dh} = \bar{x}_3$.

Vom determina testul pentru $h \equiv 1$. Deci defectul **a,** în acest caz, va fi $h \equiv 1$, iar condiția de manifestare a defectului $\bar{a}$ va fi $h = x_1 x_2 = 0$. Înlocuim în ecuația $\bar{a} \cdot \frac{dF}{da} = 1$ valorile pentru $\bar{a}$ şi $\frac{dF}{da}$ şi obținem:

$$\overline{x_1 x_2} \frac{dF}{dh} = \overline{x_1 x_2} \bar{x}_3 = 1$$
$$(\bar{x}_1 + \bar{x}_2)\bar{x}_3 = 1$$

Rezolvând ecuația şi calculând valoarea funcției obținem:

$$x_1 = 0, x_2 = *, x_3 = 0 \quad f = 0$$
$$x_1 = *, x_2 = 0, x_3 = 0 \quad f = 0 .$$

Testele rezultate sunt: (0,*,0;0) şi (*,0,0;0).

Vom determina testul pentru $h \equiv 0$. Deci defectul **a,** în acest caz, va fi $h \equiv 0$, iar condiția de manifestare a defectului $\bar{a}$ va fi $h = x_1 x_2 = 1$. Înlocuim în ecuația $\bar{a} \cdot \frac{dF}{da} = 1$ valorile pentru $\bar{a}$ şi $\frac{dF}{da}$ şi obținem:

$$x_1 x_2 \frac{dF}{dh} = x_1 x_2 \bar{x}_3 = 1$$

Rezolvând ecuația şi calculând valoarea funcției obținem:

$$x_1 = 1, x_2 = 1, x_3 = 0 \quad f = 1.$$

Testul rezultat este: (1,1,0;1).

Pentru generarea testelor în cazul a 2 eroi simultane se foloseşte derivata booleană cu 2 variabile, definită astfel:

$$\frac{dF}{d(x_i x_j)} = F(x_1, x_2, \dots x_i, \dots x_j, \dots x_n) + F(x_1, x_2, \dots \overline{x_i}, \dots \overline{x_j}, \dots x_n)$$

Observație: $\frac{dF}{d(x_i x_j)} \neq \frac{d}{dx_i}\left(\frac{dF}{dx_j}\right)$

3. TESTAREA CIRCUITELOR LOGICE SECVENȚIALE

3.1. Noțiuni de bază şi definiții

Un circuit logic combinațional realizează o dependență a valorilor binare de ieşire numai de variabilele de intrare curente. În CLC nu se ia în considerare variabila timp, cel puțin teoretic, deoarece ea nu există în suportul formal şi anume în algebra booleană.

Din punct de vedere formal, un CLC reprezintă un 3-tuplu: {X, Y, F}, în care semnificația obiectelor matematice este următoarea:

- *X* – mulțimea variabilelor de intrare;
- *Y* – mulțimea variabilelor de ieşire;
- *F* – funcția de ieşire.

Modelul matematic al CLS reprezintă un automat cu stări finite şi este definit de următorul 5-tuplu: {X, Y, S, F, G}, în care semnificația obiectelor matematice este următoarea:

- *X* – mulțimea variabilelor de intrare;
- *Y* – mulțimea variabilelor de ieşire;
- S – mulțimea stărilor interne;
- F – funcția de ieşire, care exprimă procesul de modificare a ieşirilor în dependență de variabilele de intrare şi starea internă în momentul de timp curent: $\boldsymbol{Y}(t) = \boldsymbol{F}[\boldsymbol{S}(t), \boldsymbol{X}(t)]$;
- G – funcția de tranziție a stărilor, care exprimă procesul de modificare a stărilor în următorul moment de timp în funcție de variabilele de intrare şi de starea internă în momentul de timp curent: $\boldsymbol{S}(t+1) = \boldsymbol{G}[\boldsymbol{S}(t), \boldsymbol{X}(t)]$.

În figura 3.1 este prezentat modelul general al unui circuit secvențial. Conform acestui model, un circuit secvențial este compus din circuite combinaționale şi

bistabile pentru memorarea stării interne. Partea combinațională a circuitului are două mulțimi de variabile de intrare: primare $X(t)$ (aplicate din exterior) şi secundare $S(t)$ (aplicate de la ieşirile bistabilelor). Variabilele secundare de intrare se numesc *variabile de stare,* iar mulțimea variabilelor de stare la momentul de timp t formează *starea curentă* a circuitului $S(t)$.

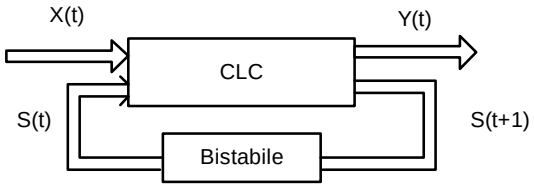


Figura 3.1. Modelul CLS

Un circuit format din m bistabile va avea 2^m stări curente. Ieşirile părții combinaționale ale circuitului sunt formate din două mulțimi. Ieşirile primare $Y(t)$, accesibile exteriorului sunt utilizate pentru gestiunea operațiilor din circuit. Ieşirile secundare se folosesc pentru specificarea stării următoare a circuitului $S(t+1)$. Modelul descris poate fi reprezentat în forma mai multor copii ale aceluiaşi CLS (figura 3.2), astfel încât starea primei copii să servească drept intrare secundară pentru copia a doua, starea celei de-a doua copii să servească drept intrare secundară pentru copia a treia, ş.a.m.d.

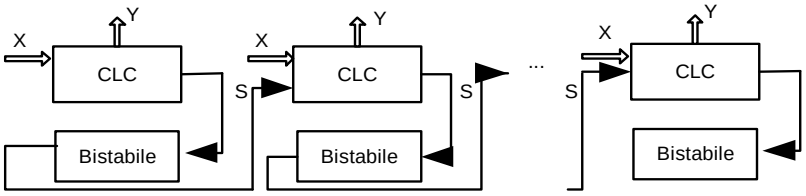


Figura 3.2. Modelul iterativ al CLS

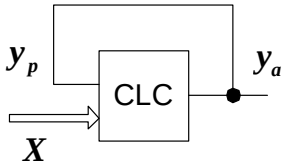
În rezultatul acestei interpretări un CLS poate fi transformat în k CLC, unde k este numărul total de stări ($k=2^m$). CLS, reprezentat astfel poate fi testat, utilizând metodele descrise pentru CLC. Dar, în acest caz, un defect singular va fi prezent în toate cele k copii şi va deveni un defect k-multiplu. Astfel, odată cu creşterea complexității CLS, testarea devine practic imposibilă.

O altă abordare pentru efectuarea testării CLS este metoda verificării funcționale în baza tabelelor de tranziție a stărilor. În acest caz, sarcina de bază este găsirea unei perechi {X(k), Y(k)}, astfel încât, secvența ieşirilor să corespundă Y(k) pentru secvența intrărilor X(k), doar în absența defectelor.

Atât în cazul primei abordări, cât şi în cazul celei de-a doua, apare problema setării inițiale a CLS în mod univoc în 1 sau 0 logic, fiind necunoscută valoarea inițială de la ieşirea circuitului. Pe lângă aceasta, pe parcursul elaborării testelor este necesară aplicarea unor seturi suplimentare de instalare a circuitului într-o valoare sau alta, pentru a fi posibilă detectarea tuturor defectelor. Aceasta măreşte şi mai mult numărul seturilor aplicate la intrarea circuitului, făcând imposibilă testarea circuitelor mari.

Testarea CLS asincrone cu o buclă de reacție

Cel mai simplu circuit secvențial asincron poate fi realizat prin introducerea unei bucle de reacție între ieşirea circuitului combinațional şi una dintre intrări



Testarea acestui tip de circuit poate fi efectuată utilizând metodele cunoscute pentru CLC în cazul întreruperii imaginare a buclei de reacție. Pentru aceasta vom lua în considerație valorile variabilei de stare la începutul şi la sfârşitul acestei bucle în momente succesive de timp.

Variabila de stare în momentul de timp actual este notată prin y_a, iar variabila de stare în momentul de timp precedent - prin y_p. Evident, în acest caz, în expresia logică care descrie funcționarea circuitului va fi prezentă şi variabila y_p.

Etapele de elaborare a testelor pentru CLS cu o buclă de reacție sunt următoarele:

1) Reprezentarea expresiei logice care descrie funcționarea circuitului în forma disjunctivă normală (FDN).

2) Aplicarea setului inițial de instalare univocă a y_a în 0 sau 1 logic, y_p fiind necunoscut.

3) Analizând FDN, se caută un termen pentru care e posibilă instalarea tuturor variabilelor în 1 logic. În acelaşi timp se asigură în ceilalți termeni cel puțin câte un 0 logic. Aceasta va permite detectarea defectelor „blocaj la 0” pentru variabilele ce sunt prezente în formă directă în termenul ales şi, respectiv, detectarea defectelor „blocaj la 1” pentru variabilele ce sunt prezente în formă inversă.

În absența defectelor variabilelor din termenul ales, valoarea y_a va fi egală cu 1 logic, în prezența defectelor, valoarea y_a va fi egală cu 0 logic.

Procedura se repetă pentru fiecare termen.

4) Analizând FDN, se caută acei termeni pentru care e posibilă instalarea în 0 logic a unei singure variabile, în ceilalți termeni fiind prezente două sau mai multe variabile egale cu 0. Aceasta va permite detectarea defectelor „blocaj la 1” pentru variabilele ce sunt prezente în formă directă în termenii aleşi şi, respectiv, detectarea defectelor „blocaj la 0” pentru variabilele ce sunt prezente în formă inversă.

În absența defectelor variabilelor din termenii aleşi, valoarea y_a va fi egală cu 0 logic, în prezența defectelor, valoarea y_a va fi egală cu 1 logic.

Procedura se repetă până când toate variabilele funcției logice vor fi testate astfel.

La elaborarea testelor se vor respecta următoarele reguli:

1) Valoarea y_p din testul curent va coincide cu valoarea y_a din testul precedent.

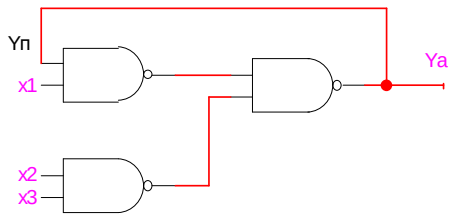
2) Atunci când valoarea y_p din setul curent nu permite asigurarea condițiilor pentru etapele 3 sau 4, se adaugă seturi de instalare a y_a în valoarea necesară.

Spre deosebire de CLC, în cazul testării CLS este importantă ordinea aplicării testelor.

Exemplu de elaborare a testelor pentru CLS cu o buclă de reacție

Fie dată următoarea expresie logică:

$$y_a = \overline{\overline{x_1 y_p} \cdot \overline{x_2 x_3}}$$



Vom reprezenta expresia logică care descrie funcționarea circuitului în forma disjunctivă normală:

$$y_a = \overline{\overline{x_1 y_p} \cdot \overline{x_2 x_3}} = x_1 y_p + x_2 x_3$$

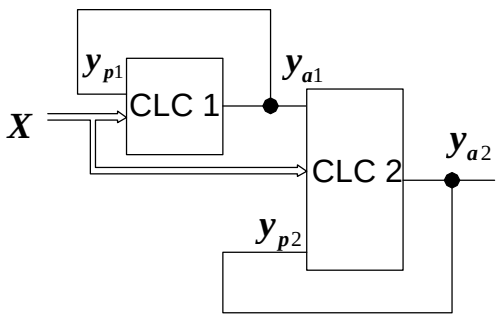
Elaborarea testelor pentru CLS cu o buclă de reacție

Nr	Teste					FDN					Defecte			
	x_1	x_2	x_3	y_p	y_a	$x_1 y_p + x_2 x_3$					x_1	x_2	x_3	y_p
1	0	1	1	*	1	0	*		1	1		≡0	≡0	
2	1	0	*	1	1	1	1		0	*	≡0			≡0
3	0	0	1	1	0	0	1		0	1	≡1	≡1		
4	1	1	0	0	0	1	0		1	0			≡1	≡1

În exemplul analizat setul inițial instalează semnalul $y_a = 1$. Prin alegerea semnalului $x_1 = 0$, acest set poate servi şi drept test pentru detectarea defectelor $x_2 \equiv 1$ şi $x_3 \equiv 1$.

Testarea CLS asincrone cu mai multe bucle de reacție

În CLS asincrone cu mai multe bucle de reacție se pot evidenția câteva subcircuite combinaționale, iar numărul variabilelor de stare depinde de numărul buclelor formate.



Vom utiliza următoarele notații: y_{a1} - variabila de stare internă în momentul de timp actual; y_{a2} - variabila de stare externă în momentul de timp actual; y_{p1} - variabila de stare internă în momentul de timp precedent; y_{p2} - variabila de stare externă în momentul de timp precedent.

Etapele:

1. Se obține FDN a expresiei logice pentru variabila de stare externă y_{a2} și FDN pentru variabila de stare internă y_{a1}.
2. Se analizează condițiile instalării inițiale a circuitului, reieşind din faptul că semnalele y_{p1} și y_{p2} sunt nedefinite. Pentru început se examinează logica funcționării şi posibilitatea instalării în 0 sau 1 logic a subcircuitului intern, apoi a celui extern.
3. Pentru detectarea defectelor de tip „blocaj la 0” şi „blocaj la 1” a intrărilor circuitului se aplică aceeaşi logică ca şi în cazul CLS cu o buclă de reacție.

Pentru a fi posibilă testarea tuturor variabilelor de intrare şi a variabilelor de stare y_{p1} și y_{p2}, este necesară aplicarea mai multor seturi de setare sau resetare a semnalelor y_{a1} și y_{a2}.

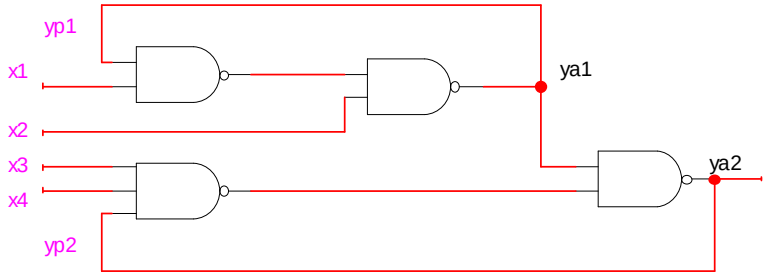
Exemplu de elaborare a testelor pentru CLS cu două bucle de reacție

Fie dată următoarea expresie logică:

$$y_{a2} = \overline{\overline{x_1 y_{p1}} \cdot x_2 \cdot \overline{\bar{x}_3 x_4 y_{p2}}}$$

Evidențiem în această expresie bucla de reacție internă:

$$y_{a1} = \overline{\overline{x_1 y_{p1}} \cdot x_2}$$



$$y_{a2} = \overline{\overline{\overline{x_1 y_{p1}} \cdot x_2} \cdot \overline{\overline{x}_3 x_4 y_{p2}}} = \overline{x_1 y_{p1}} \cdot x_2 + \overline{x}_3 x_4 y_{p2} =$$
$$= (\overline{x}_1 + \overline{y}_{p1})\overline{x}_2 + \overline{x}_3 x_4 y_{p2} = \overline{x}_1 x_2 + x_2 \overline{y}_{p1} + \overline{x}_3 x_4 y_{p2}$$

$$y_{a1} = \overline{\overline{x_1 y_{p1}} \cdot x_2} = x_1 y_{p1} + \overline{x}_2$$

Elaborarea testelor pentru CLS cu două bucle de reacție

Teste								FDN									Defecte					
x_1	x_2	x_3	x_4	y_{p1}	y_{a1}	y_{p2}	y_{a2}	$\overline{x}_1 x_2 + x_2 \overline{y}_{p1} + \overline{x}_3 x_4 y_{p2}$									x_1	x_2	x_3	x_4	y_{p1}	y_{p2}
*	0	1	*	*	1	*	0	*	0		0	*		0	*	*	$y_{a2}=0$					
0	1	1	*	1	0	0	1	1	1		1	0		0	*	0	≡1	≡0				
1	1	1	*	0	0	1	1	0	1		1	1		0	*	1		≡0			≡1	
*	0	0	1	0	1	1	1	*	0		0	1		1	1	1			≡1	≡0		≡0
1	1	1	1	1	1	1	0	0	1		1	0		0	1	1	≡0		≡0		≡0	
0	0	0	1	1	1	0	0	1	0		0	0		1	1	0		≡1				≡1
0	1	*	*	1	0	0	1	1	1		1	0		*	*	0	$y_{a2}=1$					
*	0	0	0	0	1	1	0	*	0		0	1		1	0	1				≡1		

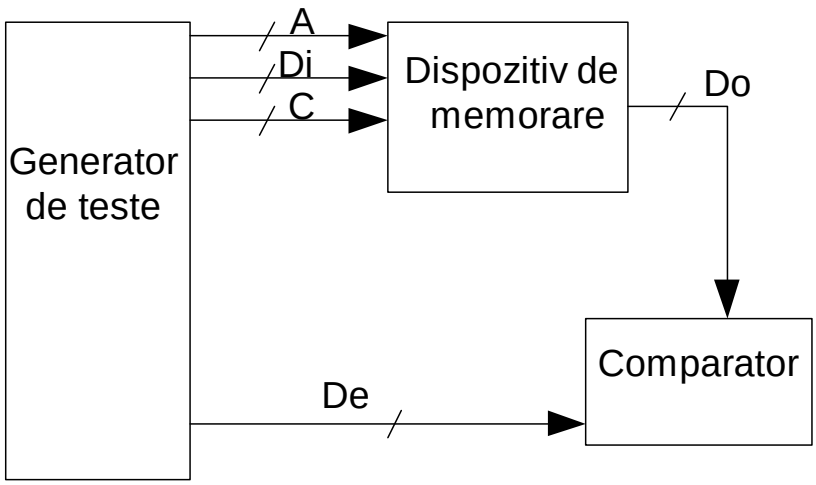
Din cele expuse reiese că metoda elaborării testelor pentru CLS cu bucle de reacție este bazată pe utilizarea metodelor pentru CLC. Se prevede doar întreruperea buclelor de reacție şi adăugarea la variabilele de intrare a variabilelor de stare precedente.

Pentru a creşte testabilitatea CLS de tip LSI (Large Scale Integration) şi VLSI (Very Large Scale Integration) sunt prevăzute un şir de măsuri: proiectarea circuitelor astfel, încât să fie posibilă separarea părții combinaționale de elementele de memorie şi testarea separată a lor; proiectarea circuitelor autotestabile, în care sunt prevăzute mijloace încorporate de generare a testelor şi tehnici de detectare a defectelor.

Testarea funcțională a memoriei

Metodele testării funționale a memoriei se bazează pe compararea variabilelor de ieşire a circuitului testat cu semnale etalon.

Schema de principiu a ecchipamentelor pentru verificarea funcțională a memoriei:



A – adresa locației de memorie la care se face apelul;

Di – datele care se înscriu în locația respectivă;

C – semnale de control care determină regimul de lucru al dispozitivului (înscriere, citire...);

De – datele etalon care sunt comparate cu datele de ieşire Do.

Generatorul de teste este destinat formării secvențelor de test şi a semnalelor etalon în conformitate cu o anumită regulă.

În prezent pentru testarea memoriei se folosesc aşa numitele teste algoritmice funcționale care reprezintă o succesiune de vectori binari-tip aplicați la intrările A şi Di în conformitate cu un algoritm. De regulă se folosesc vectori binari de tipul: 00...0, 11...1, 1010...

Semnalul etalon se generează de regulă la fel algoritmic, sau poate fi utilizată o memorie etalon.

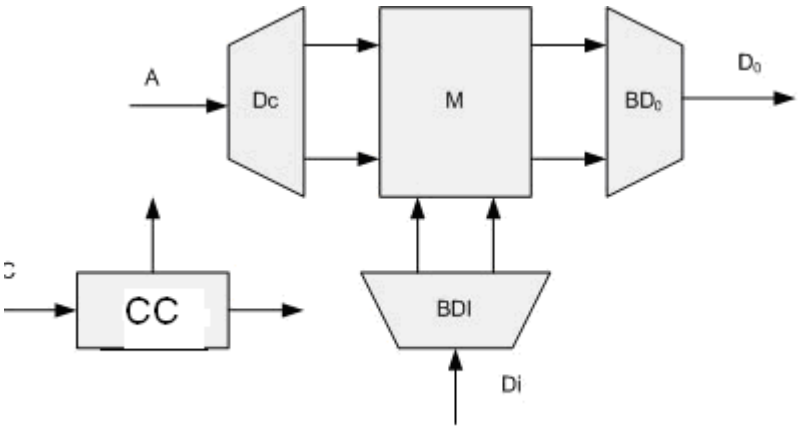
Testele algoritmice trebuie să satisfacă 2 condiții:

1. Să asigure un nivel înalt de verificare;

2. Să fie destul de scurte pentru a asigura o durată acceptabilă a procesului de verificare.

Teste pentru memoria RAM

Structura de principiu a memoriei operative:



DC – decodificator de adrese, care constă din 2 decodificatoare – a rândului şi a coloanei;

CC – circuit de control;

BDi şi BDo – circuitele informaționale de intrare şi de ieşire;

M – matricea de elemente de memorare.

Se presupune că memoria poate fi afectată de următoarele defecte:

Pentru decodificator:

1. Lipsa selecției. (Nu se activează nici o ieşire a DC).

2. Selecție multiplă. (Pentru câteva adrese diferite se activează aceiaşi ieşire).

3. Selecție nedeterminată. (Pentru aceiaşi adresă se activează simultan câteva ieşiri).

Pentru matricea M:

1. Lipsa înscrieii. (Nu se înscrie 0 sau 1 în locația de memorie: ≡1, ≡0).

2. Însriere falsă. (Din cauza legăturilor galvanice, capacitative dintre elemente).

3. Citire falsă. (Din cauza configurației nefavorabile de date din elementele vecine).

Defectele CC, BDi şi BDo reprezintă defecte clasice de tipul blocaj la 0 sau 1, scurtcircuit, şi pot fi detectate în timpul verificării decodificatorului de adrese şi a matricei M.

În dependență de numărul de cicluri „înscriere-citire”, testele algoritmice pentru verificarea memoriei RAM; se clasifică în 3 grupe: **N, N^2 и $N^{3\backslash 2}$,** unde N este numărul de locații ale memoriei.

Testele liniare N se utilizează pentru verificarea prealabilă a memoriei la determinarea defectelor globale.

Exemple.

Testul „înscrierea şi citirea consecutivă”

1. În toate celulele de memorie se înscrie fonul 0.

2. Informația se citeşte şi se compară cu cea etalon.

3. În toate celulele de memorie se înscrie fonul 1.

4. Informația se citeşte şi se compară cu cea etalon.

Lungimea testului – 4N. Testul permite detectarea parțială a lipsei de înscriere în celulele de memorie.

Testul „tabela de şah”

1. În toate celulele de memorie cu adresă pară se înscrie 0, iar în cele impare 1.

2. Informația se citeşte şi se compară cu cea etalon.

3. În toate celulele de memorie cu adresă pară se înscrie 1, iar în cele impare 0.

2. Informația se citeşte şi se compară cu cea etalon.

Lungimea testului este de de 4N. Testul permite detectarea tuturor defectelor de înscriere falsă în celulele de memorie şi parțial a defectelor de citire falsă. La fel permite şi detectarea parțială a lipsei de selecție şi a selecției nedeterminate.

Testul March „Înscriere şi citire în direcție directă şi inversă”

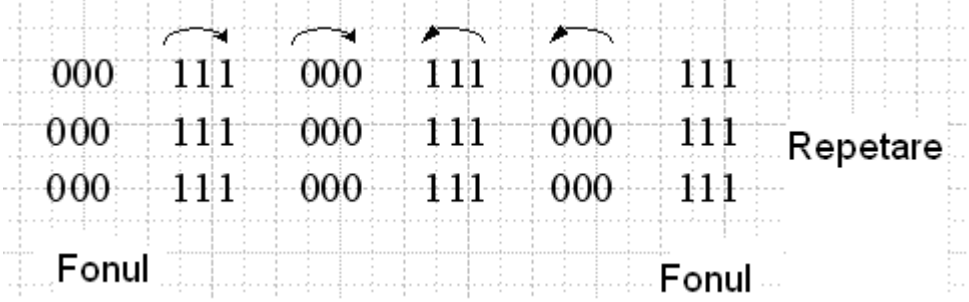
1. Înscriere fonului 0.

2. Citirea, verificarea cu informația etalon şi înscrierea lui 1 logic în direcția $A_0 \rightarrow A_{N-1}$.

3. Citirea, verificarea cu informația etalon şi înscrierea lui 0 logic în direcția $A_0 \rightarrow A_{N-1}$.

4. Repetarea p. 2 şi 3 cu schimbarea direcției de înscriere şi citire: $A_{N-1} \rightarrow A_0$.

5. Inversarea fonului (înscrierea fonului 1) şi repetarea punctelor 2-4.



Lungimea testului este de 10N. Testul permite verificarea tuturor defectelor de înscriere în celulele de memorie şi verificarea parțială a tuturor celorlalte defecte pentru memorie şi decodificator.

Teste de tipul N^2.

Specificul acestor teste este transmiterea şi verificarea informației în pereche între orice 2 celule de memorie a circuitului testat. Aceasta permite depistarea efectivă a defectelor statice şi dinamice a circuitelor RAM. Dar aplicarea acestor teste este limitată din cauza timpului foarte mare de verificare. Ele se utilizează doar în cazul necesității unei verificări foarte drastice a memoriei.

Testul „ping-pong”

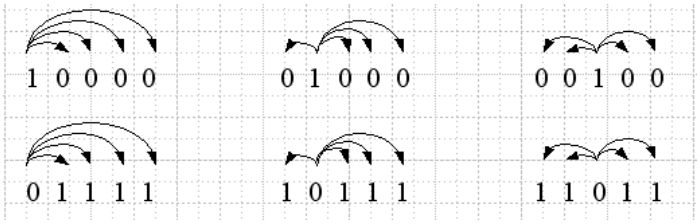
1. În prima adresă de verificare $A_k=A_0$ se înscrie 1, în toate celelalte – 0. $[A_I]= 0 \quad I = 1, N-1$.

2. Consecutiv se citeşte şi se verifică conținutul perechilor de adrese A_kA_i (A_0A_1, A_0A_2, A_0A_3, ..., A_0A_{N-1}) pentru $A_k=A_0$.

3. În adresa verificată se înscrie 0 şi se incrementează k. În noua adresă se înscrie 1. ($A_k=A_1$).

4. Se repetă punctele 2 şi 3 pentru k=0, N-1.

5. Se inversează şi se repetă punctele 2 - 4. (se înscrie 0 pe un fond de 1).



Lungimea testului este $2(2N^2+2N)$. Permite verificarea totală a defectelor memoriei şi decodificatorulu. Pentru N=64K şi durata unui ciclu de citire/înscriere de 200ns durata testului este: $T=2(2N^2+2N)\approx 2000$sec.

Teste de tipul $N^{3/2}$

Au apărut ca rezultat al unui compromis între durata testului şi calitatea verificării pentru circuitele VLSI de memorie. Lungimea acestor teste este cu mult

mai mică decât a celorde tipul N^2 deoarece se analizează doar elementele de memorie vecine.

Testul „colonița în fugă”

1. Înscrierea fonului 0.
2. Înscrierea în coloana curentăa valorii 1 logic.
3. Citirea şi verificarea tuturor celulelor de memorie.
4. Înscrierea lui 0 în celulele din coloană şi trecerea la următoarea coloană.
5. Se repetă punctele 2-4 pentru toate coloanele.
6. Inversarea şi repetarea testării.

000 100 010 001 111 011

000 100 010 001 111 011

000 100 010 001 111 011

Lungimea testului este de $2(N^{3/2}+3N)$.

Pentru N=64K, T≈9 sec.

Permite depistarea totală a defectelor pentru celulele de memorie şi parțială a tuturor defectelor decodificatorului.

Testarea memoriei RAM dinamice (DRAM)

În memoria DRAM informația se păstrează un timp limitat şi e necesară împrospătarea periodică a ei.

Testul „tabelul de şah cu regenerare”

1. În toate celulele de memorie cu adresă pară se înscrie 0, iar în cele impare 1.
2. Pauză egală cu t_{ref}.
3. Informația se citeşte şi se compară cu cea etalon.
4. Inversarea şi repetarea testului.

Testarea memoriei ROM

Cel mai simplu test constă în citirea informației încorporate şi compararea ei cu informația etalon. Uneori ultima locație conține suma de control a informației încorporate, ceea ce permite testarea fără informația etalon.

Teste mai complicate se elaborează în baza testelor algoritmice funcționale luând în considerație tehnologia de programare şi reprogramare a memoriei ROM.

Proiectarea circuitelor testabile

Dificultatea de bază în cazul testării circuitelor secvențiale mari este setarea şi analiza stărilor bistabilelor, numărul cărora este semnificativ în asemenea circuite. De aceea, luând în considerare imposibilitatea practică de elaborare a testelor pentru CLS mari, se impune modificarea circuitelor în scopul simplificării procedurii de testare. Una dintre primele abordări, cunoscută sub denumirea de metoda căii scanate (Scan Path), constă în separarea părții combinaționale de elementele de memorie şi testarea lor separată. Testarea circuitului combinațional poate fi efectuată prin metode deterministe sau nedeterministe. Pentru testarea elementelor de memorie (bistabilelor), acestea se unesc într-un registru unic de deplasare care, în regim de testare, primeşte anumiți vectori-test standard de tipul: 100...0, 011...1 ş.a. Astfel, fiecare element de memorie primeşte toate combinațiile posibile ale variabilei de stare în momentele de timp precedent şi actual.

În prezent există mai multe tehnici de acest gen, toate având drept scop creşterea testabilității circuitelor proiectate. Testabilitatea unui circuit constituie aptitudinea acestuia de detectare şi localizare uşoară a defectelor posibile. Testabilitatea se realizează în faza de proiectare, regulile de proiectare urmărind să crească observabilitatea şi controlabilitatea circuitului respectiv.

Proiectarea circuitelor testabile se bazează pe două metode:

1) Proiectarea circuitelor astfel încât să fie uşor testabile. Această metodă poate introduce întârzieri mari în propagarea semnalelor şi o creştere a numărului de porți logice.

2) Proiectarea circuitelor autotestabile prin introducerea elementelor logice adiționale. Aceasta conduce la o creştere a complexității circuitului, dar reduce numărul de teste necesare pentru diagnoză. Metoda se poate aplica la nivelul de plăci de circuite integrate prin introducerea unor puncte de test suplimentare.

Prima metodă presupune utilizarea tehnicii de proiectare LSSD (Level Sensitive Scan Design) şi este o continuare a metodei căii scanate. Ea este folosită pentru circuitele electronice integrate de tip LSI sau VLSI. Implementarea acestei metode se bazează pe următoarele concepte:

· Toate schimbările de stare a circuitului sunt determinate de nivelul logic al semnalului de test şi nu de fronturile acestuia (Level Sensitive);

· În regim de testare toate elementele de memorie formează un registru unic de deplasare, în care poate fi înscrisă orice stare a circuitului şi care poate fi utilizat pentru testarea părții combinaționale a lui (Scan Design).

În acest scop se utilizează o structură logică special proiectată de tip SRL (Shift Register Latch), a cărui schemă bloc este prezentată în figura 3.10. Bistabilul B_1 este utilizat în regim de funcționare normală şi în regim de testare, bistabilul B_2 este utilizat doar în regim de testare. Bistabilul B_1 în regim de funcționare normală are intrarea de date pe *D*, semnalul de tact al echipamentului pe intrarea *C* şi ieşirea pe L_1. Pe perioada funcționării normale semnalele de tact *A* şi *B* sunt fixate în 0 logic. Memorarea datelor din echipament are loc în momentul trecerii semnalului *C* din 1 în 0 logic. În regim de testare semnalul de tact *C* este blocat în 0 logic. Pentru încărcarea datelor de pe intrarea *SDI* (Scan Data Input) în bistabilul B_1, semnalul de tact *A* este instalat în 1 logic. În momentul trecerii acestui semnal în 0, datele se fixează la ieşirea L_1. În acelaşi timp, semnalul de tact *B* se instalează în 1 logic pentru a efectua transferul informației de pe L_1 în bistabilul B_2. Pentru fixarea datelor, la ieşirea L_2 se efectuează trecerea semnalului de tact *B* în 0 logic. Formarea registrului de deplasare în regim de testare are loc prin conexiunea ieşirii L_2 a unei structuri SRL cu intrarea *SDI* a următoarei structuri SRL.

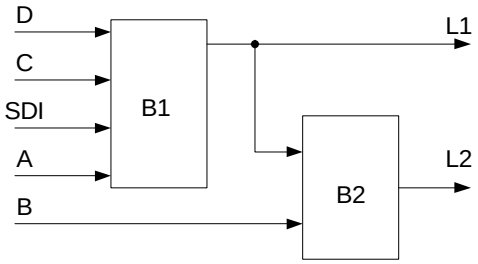


Figura 3.10. Structura SRL

Realizarea unei testări automate în tehnica LSSD presupune următoarele etape:

a) se testează registrele SRL, transferându-se în mod serial o secvență logică cunoscută;

b) se testează blocurile combinaționale, transferându-se la intrarea lor vectorii de test necesari prin intermediul unui registru de deplasare;

c) se citeşte vectorul semnalelor rezultate la ieşirea blocului combinațional, extrăgându-se informația în mod serial prin intermediul registrului de deplasare;

d) se repetă acțiunile de la punctele *b* şi *c* până la expirarea programului de test.

Proiectarea circuitelor LSI şi VLSI autotestabile se bazează pe următoarele principii: secvențele de test sunt generate nemijlocit în circuit; reacțiile la secvențele de test se păstrează la fel în circuit; pentru realizarea autotestării e necesar doar a iniția procedurile de testare şi a analiza rezultatele.

Generarea pseudoaleatoare a vectorilor de test poate fi realizată pe un registru de deplasare cu bucle de reacție, ce extrage informația din diferite puncte ale registrului şi o transportă la intrarea acestuia. Structura unui astfel de registru, numit LFSR (Linear Feedback Shift Register), este prezentată în figura 3.11.

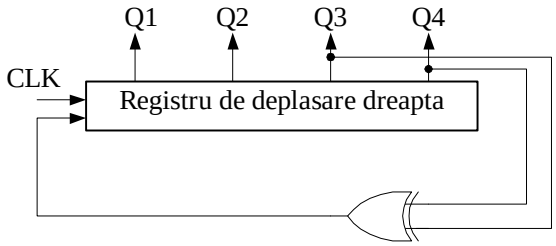


Figura 3.11. Generarea vectorilor de test pe un registru de deplasare

Metoda este simplă, uşor de realizat şi permite obținerea unui număr de 2^{n-1} vectori de test, unde *n* este numărul de bistabile din registru.

Metodele de analiză a rezultatelor aplicării secvenței de test constau fie în numărarea tranzițiilor (numărarea trecerii semnalului din 0 în 1 logic şi invers într-un anumit interval de timp), fie în analiza de semnătură (reacția unică a nodului din circuit la un vector de testare cunoscut). Scopul de bază a acestor metode este comprimarea informației obținute în urma aplicării secvențelor de test.

În figura 3.12 este prezentat analizorul de semnătură realizat în baza registrului de deplasare LFSR pe 16 biți.

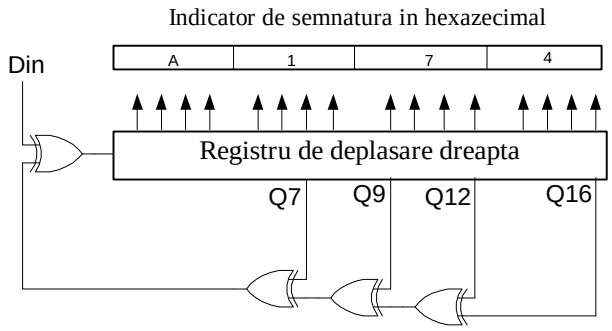


Figura 3.12. Analizor de semnătură

La intrarea registrului de deplasare LFSR se aplică printr-o poartă SAU-EXCLUSIV două surse de informație: şirul de *m* biți rezultați în urma aplicării secvenței de test pentru un nod anumit din circuitul testat şi şirul de semnale de pe bucla de reacție. La terminarea secvenței de test, LSFR conține un cuvânt de 16 biți, care poartă denumirea de *semnătură* a nodului analizat. Orice defect al nodului va cauza o eroare la ieşirea lui, care va determina schimbarea conținutului registrului, iar semnătura rezultată va fi diferită față de cea etalon.

Una dintre cele mai răspândite structuri, care permite realizarea unui şir de funcții necesare pentru implementarea autotestării, este BILBO (Built-In Logic Block Observation). Structura BILBO este realizată în baza unui registru de deplasare cu buclă de reacție şi mai multe porți logice. Prin intermediul a patru semnale de tact structura poate trece în următoarele regimuri: resetarea registrului, funcționare normală, generator de teste şi analizor de semnătură.

CODURI DETECTOARE ŞI CORECTOARE DE ERORI

Metode de protecție a datelor împotriva erorilor tranziente

În sistemele de calcul contemporane cea mai mare parte a erorilor sunt tranziente. Principalele cauze ale erorilor tranziente sunt perturbațiile: zgomotul, interferența semnalelor. Aceste erori, spre deosebire de erorile permanente, sunt cu mult mai greu de depistat din cauza caracterului temporar.

Sistemele fiabile, tolerante la astfel de erori, pot fi construite doar folosind anumite tipuri de **redundanță**.

Se deosebesc următoarele tipuri de redundanță:

1) *Redundanță hardware* - foloseşte mai multe componente decât strictul necesar pentru a implementa un anumit sistem.

2) *Redundanță software* - Se scriu versiuni diferite de software pentru aceeaşi aplicație.

2) *Redundanță temporală* – utilizează un singur dispozitiv pentru a calcula acelaşi lucru în mod repetat, după care rezultatele se compară între ele.

3) *Redundanță informațională* - se realizează prin adăugarea de biți suplimentari la cei originali. Erorile în biți pot fi detectate şi chiar corectate.

Costul redundanței hardware este foarte mare, deoarece este necesară replicarea identică a unui întreg sistem. De exemplu, redundanța modulară triplă, propusă de John von Neumann în 1956, are o eficiență de 33%. Redundanța temporală duce, cel puțin, la dublarea timpului necesar pentru efectuarea calculelor sau pentru transmiterea datelor.

Redundanța informațională oferă metode de protecție a datelor împotriva erorilor, folosind mai puține resurse suplimentare şi implică utilizarea codurilor detectoare şi corectoare de erori.

Metodele de protecție a datelor împotriva erorilor introduse de canalul de transmisiune implică folosirea unui codor în transmițător, a unui decodor în receptor şi a unei strategii de control al erorii.

Strategia de utilizare a codorului şi decodorului:

1. *Detecţia simplă a erorilor* - entitatea care primeşte datele este informată despre blocurile de date recepționate cu erori.

2. *Detecţia şi corectarea erorilor.* Se disting doua cazuri:

1) detectarea blocurilor de date recepționate eronat şi *corectarea erorilor prin retransmiterea* acestor blocuri;

2) *corectarea directă*, la recepție, a erorilor.

Strategia corectării directe a erorilor necesită utilizarea unor coduri corectoare de erori. Celelalte strategii, *de detecţie simplă a erorilor sau de corectare prin retransmitere,* necesită utilizarea unor coduri detectoare de erori. Corectarea erorilor prin retransmitere este mai simplă în ceea ce priveşte complexitatea decodorului, dar necesită doua căi de transmisiune: una pentru a transmite blocurile de date şi alta, în sens invers, pentru a transmite confirmările de recepție (pozitivă, fără erori şi negativă, cu erori). În plus, corectarea erorilor se face cu o anumită întârziere.

Noțiuni de bază din teoria codurilor

Fie lungimea unui cod binar egală cu n biți. Numărul maxim de combinații care pot fi obținute în acest caz este $N=2^n$.

Un asemenea cod, unde toți biții sunt informaționali se numeşte **cod simplu** sau **cod iredundant.**

Codurile simple nu pot fi folosite la detectarea erorilor deoarece orice eroare va duce la apariția unei combinații admisibile în acest cod.

Codurile în care în afară de biți informaționali se folosesc şi biți de control se numesc **coduri redundante**.

În asemenea coduri o parte din combinațiile posibile sunt interzise. Apariția lor permite detectarea erorilor, iar pentru unele coduri şi corectarea lor.

O noțiune importantă în teoria codurilor o reprezintă **distanța Hamming.** Distanța Hamming între două cuvinte de cod este dată de numărul de poziții binare, prin care cele două cuvinte diferă. Evident distanța Hamming se poate defini numai pentru cuvinte de lungimi identice.

Distanța Hamming dintre două cuvinte binare se calculează prin efectuarea adunării modulo 2 dintre biții acestor cuvinte. Spre exemplu, distanța Hamming dintre cuvintele binare 011011 şi 110111 este egală cu trei:

$$\begin{array}{r} 011011 \\ \oplus\ \underline{110111} \\ 101100 \end{array}$$

Se numeşte **distantă minimă (d_{min})** a unui cod cea mai mică distantă Hamming între două cuvinte distincte ale codului.

Determinați Dmin pentru codul 2din5.

0 00011 1 00101 2 00110 3 01001 4 01010 5 01100 6 10001 7 10010 8 10100 9 11000	Din combinatorica cunoaştem ca formula pentru a determina cate combinari de n elemente luate cate *k*. $C_n^k = \frac{n!}{k!(n-k)!}$, $C_5^2 = \frac{5!}{2!(5-2)!}$

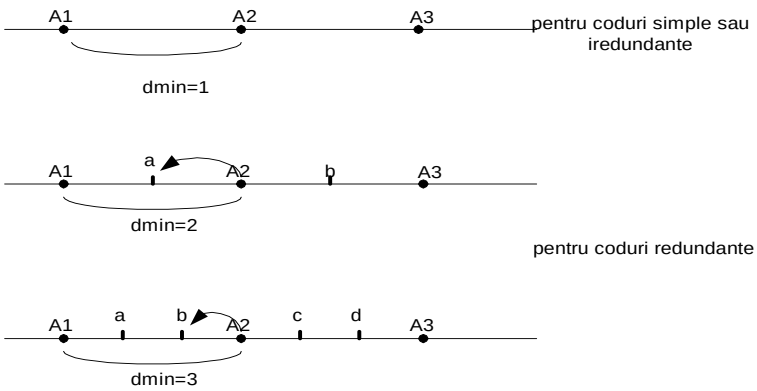
În general, capacitatea codului a detecta şi a corecta erorile este determinată de următoarea relație:

$$\mathbf{d_{min}=r+s+1,}$$

unde **r** este numărul erorilor detectate şi **s** este numărul erorilor corectate.

Deci, un cod poate *detecta* o eroare singulară pentru $d_{min}=2$. Un cod poate *detecta şi corecta* o eroare singulară pentru $d_{min}=3$. Pentru codurile simple $d_{min}=1$.

Interpretarea grafică:



În cazul cînd $d_{min}=2$ orice eroare de rangul 1 va duce la apariția unei combinații interzise, dar nu va fi clar de la care combinație ea a derivate. În cazul cînd $d_{min}=3$

orice eroare de rangul 1 va duce la apariția unei combinații interzise care se va deosebi într-un bit de combinația de la carea derivate şi în doi sau mai mulți biți de la oricare altă combinație. Deci cuvîntul de cod correct poate fi determinat.

Coduri cu paritate

Codurile cu paritate se utilizează la detectarea erorilor la transmiterea datelor (protocolul COM, RS) , în circuitele de memorare (memorie cu paritate), în operațiile aritmetice, ş. a.

Ele sunt bazate pe o metodă foarte simplă de codare, care constă în completarea cuvintelor de cod cu un simbol de control, numit bit de paritate. Pentru un cod cu paritate cu soț (sau fără soț), bitul suplimentar este fixat astfel, ca numărul total de unități binare din cuvântul de cod să fie totdeauna par (sau impar).

Un cod cu paritate are $d_{min}=2$ şi este capabil a detecta erorile care apar în număr impar într-un cuvânt de cod. Dacă un bit se schimbă din 0 în 1 sau invers, paritatea se schimbă şi eroarea poate fi detectată prin verificarea parității. Această paritate simplă nu poate însă corecta bitul eronat.

Lungimea cuvintelor de cod este de $n=n_0+1$, unde n_0 este numărul simbolurilor informaționale.

Exemplu:

cuvîntul de cod	
01000001	0
01100001	1
01001110	0

O astfel de codificare se numeşte metoda paritatii simple.

O creştere a capacității de detectare se poate obține prin **metoda parității încrucişate,** care se aplică unor blocuri de date. În schema cea mai simplă, informația este organizată într-o matrice bidimensională. Biții de la finele unei linii sunt biți de paritate pe acea linie (paritate transversală), biții de la baza unei coloane sunt biți de paritate pe acea coloană (paritate longitudinală). O eroare pe un bit, situată oriunde în matricea utilizată, afectează corectitudinea parității atât pe o linie cât şi pe o coloană.

01000001	0
01100001	1
01001110	0
01101110	1

Paritatea longitudinala si transversala

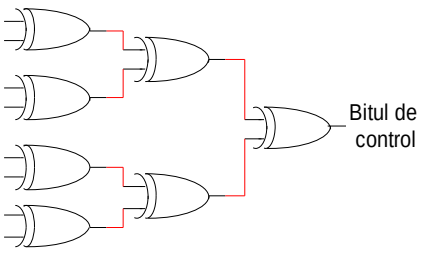
La recepție se calculează încă o dată paritatea transversală şi longitudinală a informației transmise. Comparând parităţile recepționate cu cele calculate, se poate afirma că blocul de informație a fost transmis: *fără erori*, dacă parităţile recepționate şi calculate coincid; *cu erori*, dacă cel puțin pe o coloană şi/sau pe o linie parităţile recepționate şi calculate nu coincid.

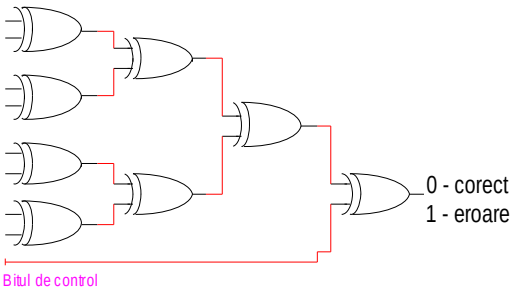
Circuite de codare si de decodare pentru coduri cu paritate

Codorul este un sumator de biți modulo-2. Acesta generează un 0 dacă numărul de unităţi binare din cuvântul format din simboluri informaționale este cu soț şi un 1 dacă numărul de unităţi binare a acestui cuvânt este fără soț. Ieşirea sumatorului este semnalul de paritate.

Decodorul regenerează bitul de paritate din biții care constituie partea de date în cuvântul recepționat si îl compară cu bitul de paritate primit. Dacă biții de paritate: cel evaluat şi cel primit coincid ieşirea porții SAU-EXCLUSUV din ultimul nivel este un 0, ceea ce indică absența erorii. Lipsa coincidenței produce în acelaşi punct al schemei un 1 ceea ce indică o eroare.

Erorile pe doi biți nu pot fi detectate de un cod cu paritate, deoarece paritatea nu este modificată prin inversarea a doi biți. Toate erorile de trei biți într-un cuvânt, desigur o situație mult mai rară, sunt detectate fără a putea discerne dacă este vorba de una sau trei erori în biții cuvântului de cod.





Pentru paritate cu soț.

Coduri Hamming

Un cod Hamming *(n,n_0)* este format din cuvinte de cod cu lungimea de *n* simboluri, dintre care n_0 simboluri sunt informaționale şi $k=n-n_0$ simboluri sunt de control.

Codul Hamming corector de o eroare are $d_{min}=3$.

Dacă vom considera că la transmiterea unui cuvânt de *n* simboluri poate apărea o singură eroare, atunci numărul variantelor posibile va fi egal cu *n*+1, în care o combinație va reprezenta cuvântul de cod transmis fără erori, iar celelalte *n* combinații - toate variantele posibile de erori singulare. Deci, pentru a corecta presupusa eroare, utilizând *k* simboluri de control, este necesar a descrie *n+1* evenimente distincte: $\boldsymbol{n}+1=\boldsymbol{n}_0+\boldsymbol{k}+1\le 2^k$ (4.1)

Din relație rezultă:

1) Lungimea cuvântului de cod: $\boldsymbol{n}\le 2^k-1$
2) Numărul simbolurilor informaționale: $\boldsymbol{n}_0\le 2^k-\boldsymbol{k}-1$

Utilizând relația (4.1), vom calcula numărul simbolurilor de control *k*, fiind cunoscut numărul simbolurilor informaționale n_0.

n_0	1	2-4	5-11	12-26	27-57
k	2	3	4	5	6
n	3	5-7	9-15	17-31	33-63

Detectarea erorilor în codurile Hamming se bazează pe acelaşi principiu ca şi în codurile cu paritate. Deosebirea constă în următoarele:

- *pozițiile de control sunt date de puteri ale lui 2;*
- *fiecare simbol de control răspunde pentru paritatea anumitor simboluri informaționale.*

Simbolurile de control se plasează pe pozițiile 1, 2, 4, 8, 16, ... deoarece la reprezentarea binară a acestor numere este prezentă doar o singură unitate şi astfel se simplifică procedura de calcul a acestor simboluri. Fiecare simbol de control reprezintă paritatea unui set de simboluri informaționale. Poziția simbolului de control determină secvența biților, care *verifică* şi *sare* alternativ. Poziția 1 verifică

un bit şi sare 1 bit (1, 3, 5, 7, 9 etc.). Poziția 2 verifică 2 biți şi sare 2 biți (2, 3, 6, 7, 10, 11 etc.). Poziția 4 verifică 4 biți şi sare 4 biți (4, 5, 6, 7, 12, 13, 14, 15 etc.). Poziția 8 verifică 8 biți şi sare 8 biți (8-15, 24-31 etc.).

După transmiterea cuvintelor de cod, se calculează încă o dată paritatea seturilor de biți, formându-se *sindromul erorii*. Dacă cuvintele au fost transmise fără erori, sindromul este nul. În cazul apariției unei erori, sindromul va indica poziția ei în cuvântul transmis, iar simbolul eronat va fi corectat prin inversare. În final, din cuvântul de cod vor fi extrase doar simbolurile informaționale.

Vom examina codul Hamming (12,8). Un cuvânt de cod are $n=12$ simboluri, dintre care $n_0=8$ simboluri informaționale şi $k=n-n_0=4$ simboluri de control.

Vom nota prin d_i simbolurile informaționale şi prin c_i simbolurile de control.

Se va genera un cuvânt de cod $\boldsymbol{V}$

$$\boldsymbol{V}=\{\boldsymbol{d}_{12},\boldsymbol{d}_{11},\boldsymbol{d}_{10},\boldsymbol{d}_9,\boldsymbol{c}_8,\boldsymbol{d}_7,\boldsymbol{d}_6,\boldsymbol{d}_5,\boldsymbol{c}_4,\boldsymbol{d}_3,\boldsymbol{c}_2,\boldsymbol{c}_1\}$$

Calculul simbolurilor de control:

$$\boldsymbol{c}_1=\boldsymbol{d}_3\oplus\boldsymbol{d}_5\oplus\boldsymbol{d}_7\oplus\boldsymbol{d}_9\oplus\boldsymbol{d}_{11}$$
$$\boldsymbol{c}_2=\boldsymbol{d}_3\oplus\boldsymbol{d}_6\oplus\boldsymbol{d}_7\oplus\boldsymbol{d}_{10}\oplus\boldsymbol{d}_{11}$$
$$\boldsymbol{c}_4=\boldsymbol{d}_5\oplus\boldsymbol{d}_6\oplus\boldsymbol{d}_7\oplus\boldsymbol{d}_{12}$$
$$\boldsymbol{c}_8=\boldsymbol{d}_9\oplus\boldsymbol{d}_{10}\oplus\boldsymbol{d}_{11}\oplus\boldsymbol{d}_{12}$$

La recepționarea cuvântului de cod $\boldsymbol{V}'=\{\boldsymbol{d}'_{12},\boldsymbol{d}'_{11},\boldsymbol{d}'_{10},\boldsymbol{d}'_9,\boldsymbol{c}'_8,\boldsymbol{d}'_7,\boldsymbol{d}'_6,\boldsymbol{d}'_5,\boldsymbol{c}'_4,\boldsymbol{d}'_3,\boldsymbol{c}'_2,\boldsymbol{c}'_1\}$, se va calcula sindromul erorii conform următoarelor relații:

$$\boldsymbol{s}_1=\boldsymbol{c}'_1\oplus\boldsymbol{d}'_3\oplus\boldsymbol{d}'_5\oplus\boldsymbol{d}'_7\oplus\boldsymbol{d}'_9\oplus\boldsymbol{d}'_{11}$$
$$\boldsymbol{s}_2=\boldsymbol{c}'_2\oplus\boldsymbol{d}'_3\oplus\boldsymbol{d}'_6\oplus\boldsymbol{d}'_7\oplus\boldsymbol{d}'_{10}\oplus\boldsymbol{d}'_{11}$$
$$\boldsymbol{s}_4=\boldsymbol{c}'_4\oplus\boldsymbol{d}'_5\oplus\boldsymbol{d}'_6\oplus\boldsymbol{d}'_7\oplus\boldsymbol{d}'_{12}$$
$$\boldsymbol{s}_8=\boldsymbol{c}'_8\oplus\boldsymbol{d}'_9\oplus\boldsymbol{d}'_{10}\oplus\boldsymbol{d}'_{11}\oplus\boldsymbol{d}'_{12}$$

Sindromul erorii $(\boldsymbol{s}_8,\boldsymbol{s}_4,\boldsymbol{s}_2,\boldsymbol{s}_1)$ va fi nul, dacă cuvântul a fost transmis fără erori. În cazul apariției unei erori, sindromul va indica poziția simbolului eronat.

Să analizăm un exemplu concret. Fie că este necesar a forma un cuvânt de cod pentru simbolurile informaționale {1,1,0,0,1,1,1,0}. Le vom plasa în felul următor:

Poz	d_{12}	d_{11}	d_{10}	d_9	c_8	d_7	d_6	d_5	c_4	d_3	c_2	c_1
V	1	1	0	0		1	1	1		0		

Vom calcula simbolurile de control:

$$\boldsymbol{c}_1=\boldsymbol{d}_3\oplus\boldsymbol{d}_5\oplus\boldsymbol{d}_7\oplus\boldsymbol{d}_9\oplus\boldsymbol{d}_{11}=0\oplus1\oplus1\oplus0\oplus1=1$$

$$c_2 = d_3 \oplus d_6 \oplus d_7 \oplus d_{10} \oplus d_{11} = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$$

$$c_4 = d_5 \oplus d_6 \oplus d_7 \oplus d_{12} = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$c_8 = d_9 \oplus d_{10} \oplus d_{11} \oplus d_{12} = 0 \oplus 0 \oplus 1 \oplus 1 = 0$$

Se va obține următorul cuvânt de cod:

Poz	d_{12}	d_{11}	d_{10}	d_9	c_8	d_7	d_6	d_5	c_4	d_3	c_2	c_1
V	1	1	0	0	0	1	1	1	0	0	1	1

Să presupunem, că la transmitere a fost recepționat cuvântul de cod

Poz	d_{12}'	d_{11}'	d_{10}'	d_9'	c_8'	d_7'	d_6'	d_5'	c_4'	d_3'	c_2'	c_1'
V'	1	1	1	0	0	1	1	1	0	0	1	1

Pentru a determina dacă cuvântul de cod a fost transmis corect, se va calcula sindromul erorii

$$s_1 = c_1' \oplus d_3' \oplus d_5' \oplus d_7' \oplus d_9' \oplus d_{11}' = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$$
$$s_2 = c_2' \oplus d_3' \oplus d_6' \oplus d_7' \oplus d_{10}' \oplus d_{11}' = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$
$$s_4 = c_4' \oplus d_5' \oplus d_6' \oplus d_7' \oplus d_{12}' = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$
$$s_8 = c_8' \oplus d_9' \oplus d_{10}' \oplus d_{11}' \oplus d_{12}' = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

Sindromul (s_8, s_4, s_2, s_1)=(1,0,1,0) indică că eroarea a avut loc în poziția 10 (bitul d_{10}). Acest bit poate fi corectat prin inversare.

Să presupunem, că la transmitere a fost recepționat un cuvânt de cod cu 2 erori:

Poz	d_{12}'	d_{11}'	d_{10}'	d_9'	c_8'	d_7'	d_6'	d_5'	c_4'	d_3'	c_2'	c_1'
V'	1	1	0	0	0	0	1	1	0	0	0	1

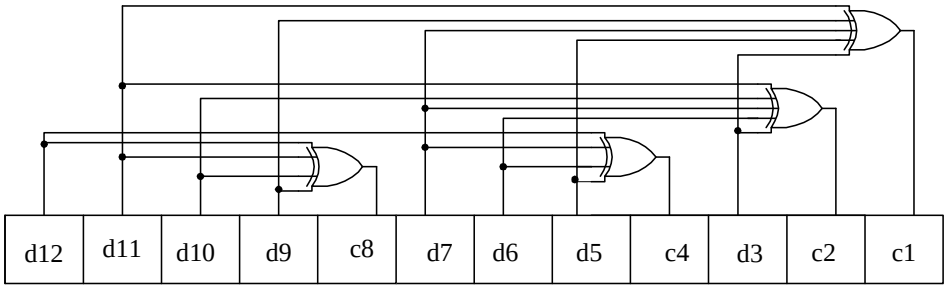
$$s_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$$
$$s_2 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0$$
$$s_4 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$s_8 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 0$$

Sindromul indică eroare în poziția 4, pe când erorile au avut loc în pozițiile 2 şi 6.

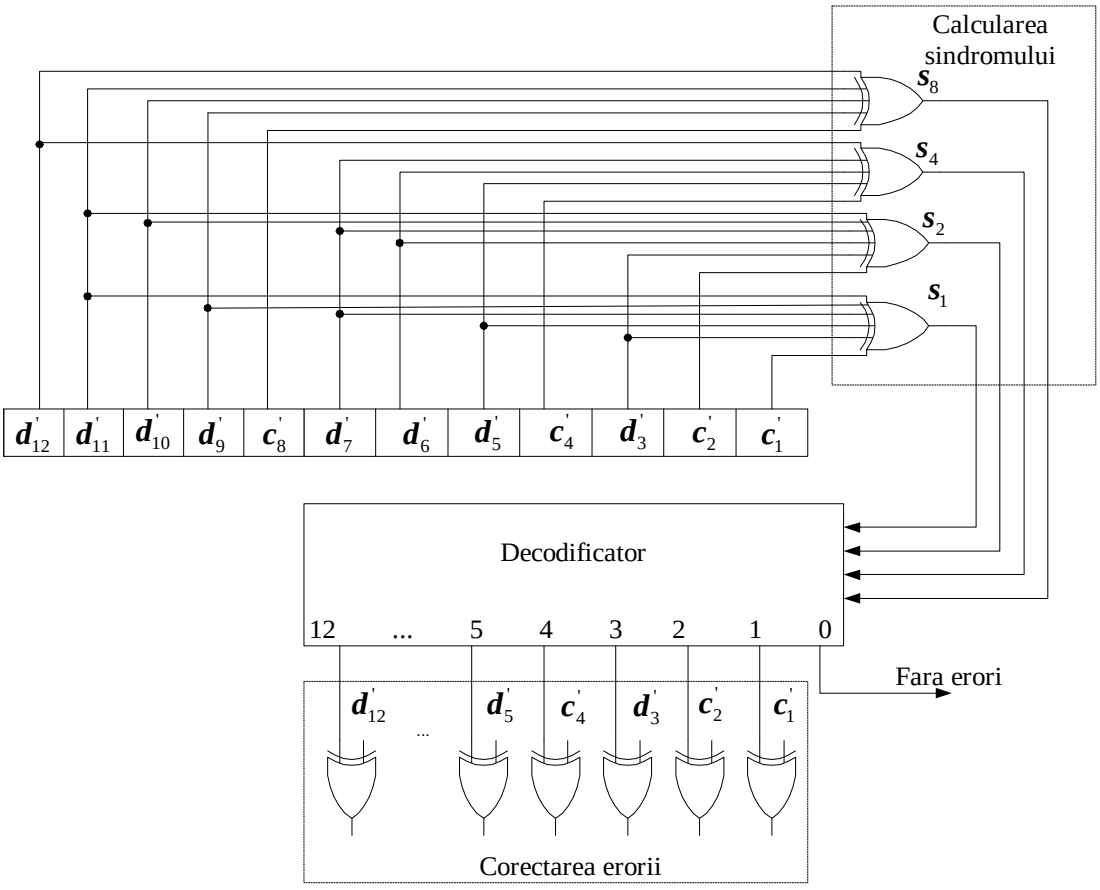
Codorul şi decodorul Hamming detector şi corector de o eroare

Codorul este format dintr-un registru, care conține *n* circuite basculante bistabile, în care se stochează simbolurile informaționale în pozițiile corespunzătoare din cuvântul de cod şi o serie de sumatoare modulo 2, care calculează simbolurile de

control. Simbolurile de control, astfel calculate, se stochează apoi în pozițiile, în care acestea intervin în cuvântul de cod.



Decodorul Hamming este format dintr-un registru cu *n* circuite basculante bistabile, în care va fi memorat cuvântul recepționat, *k* sumatoare modulo 2, care calculează componentele sindromului şi un decodificator binar cu *k* intrări şi *n+1* ieşiri. La intrările decodificatorului vor fi aplicați cei *k* biți ai sindromului. În dependență de semnalul activ de la ieşirea decodificatorului se va stabili dacă cuvântul de cod a fost transmis fără erori sau, în cazul unei erori, se va determina poziția acesteia. Corectarea prin inversare a simbolului eronat se va efectua prin intermediul a *n* sumatoare modulo 2.



Codul Hamming detector de două erori şi corector de o eroare

În scopul corecției unei erori şi a detectării erorilor duble trebuie adăugat un simbol suplimentar, numit de *control al parității*, prin care se decide dacă în cuvântul recepționat sunt două erori sau o eroare.

Structura cuvântului de cod în acest caz este de forma:

$$\boldsymbol{V} = \{\boldsymbol{d}_{12}, \boldsymbol{d}_{11}, \boldsymbol{d}_{10}, \boldsymbol{d}_9, \boldsymbol{c}_8, \boldsymbol{d}_7, \boldsymbol{d}_6, \boldsymbol{d}_5, \boldsymbol{c}_4, \boldsymbol{d}_3, \boldsymbol{c}_2, \boldsymbol{c}_1, \boldsymbol{c}_0\}$$

Simbolul suplimentar de control al parității c_0 determină paritatea tuturor simbolurilor din cuvântul de cod şi este calculat conform relației: $\boldsymbol{c}_0 = \boldsymbol{d}_{12} \oplus \boldsymbol{d}_{11} \oplus \boldsymbol{d}_{10} \oplus \boldsymbol{d}_9 \oplus \boldsymbol{c}_8 \oplus \boldsymbol{d}_7 \oplus \boldsymbol{d}_6 \oplus \boldsymbol{d}_5 \oplus \boldsymbol{c}_4 \oplus \boldsymbol{d}_3 \oplus \boldsymbol{c}_2 \oplus \boldsymbol{c}_1$

Sindromul erorii va include un simbol suplimentar $\boldsymbol{s}_0$ şi va avea următoarea formă: $\boldsymbol{s} = \{\boldsymbol{s}_8, \boldsymbol{s}_4, \boldsymbol{s}_2, \boldsymbol{s}_1, \boldsymbol{s}_0\}$

Biții $\boldsymbol{s}_8, \boldsymbol{s}_4, \boldsymbol{s}_2$ şi $\boldsymbol{s}_1$ vor fi calculați conform relațiilor respective, iar bitul $\boldsymbol{s}_0$ conform următoarei relații:

$$\boldsymbol{s}_0 = \boldsymbol{d}'_{12} \oplus \boldsymbol{d}'_{11} \oplus \boldsymbol{d}'_{10} \oplus \boldsymbol{d}'_9 \oplus \boldsymbol{c}'_8 \oplus \boldsymbol{d}'_7 \oplus \boldsymbol{d}'_6 \oplus \boldsymbol{d}'_5 \oplus \boldsymbol{c}'_4 \oplus \boldsymbol{d}'_3 \oplus \boldsymbol{c}'_2 \oplus \boldsymbol{c}'_1 \oplus \boldsymbol{c}'_0$$

Criteriile de detectare a erorilor:

$\boldsymbol{s} = \{\boldsymbol{s}_8, \boldsymbol{s}_4, \boldsymbol{s}_2, \boldsymbol{s}_1, \boldsymbol{s}_0\}$	c'_0	Erori
0	0	Erori nu sunt
$\neq 0$	1	1 eroare
$\neq 0$	0	2 erori
0	1	Eroare in c'_0

Avantaje: 1. Costul de implementare este relativ mic. Pentru o magistrală de date de 64 biți, algoritmul dat necesită 8 biți de control, la fel caşi în cazul utilizării codului cu paritate.

2. Timpul de reținere este proporțional cu $\log_2 \boldsymbol{N}$ şi poate necesita încă un ciclu de citire latentă în dependență d estructura circuitului de control.

Neajunsuri: Nu este posibil de detectat erori de rang impar, care pot fi confundate cu erori de rangul1 1.

Coduri ciclice (CRC- Cyclic Redundancy Code)

Codurile ciclice se utilizează pe larg la transmiterea datelor datorită eficienței înalte de detectare şi corectare a erorilor. Ele au o redundanță relativ mică iar dispozitivele de codare şi decodare se pot realiza simplu, în baza registrelor de delasare cu reacție.

La fel ca şi codurile Hamming, codurile ciclice sunt sistematice liniare- bloc. Fiecare cuvânt de cod este codat şi decodat aparte (coduri bloc), cuvintele de cod constau din n_0 biți informaționali şi k biți de control (coduri sistematice).

Codurile ciclice se notează (n, n_0).

Cuvintele de cod se pot scrie sub forma unei matrice de formare cu n coloane, în care n_0 rânduri sunt liniar independente. Fiecare rând din matrice poate fi obținut prin deplasarea ciclică la stânga a unui cuvânt de cod inițial.

0001011
0010110
0101100
1011000
0110001
1100010
1000101

Elementele cuvântului de cod $d = (d_{n-1}, d_{n-2}, ..., d_0)$ pot fi considerate drept coeficienți a unui polinom cu variabila fictivă x.

$$d(x) = d_{n-1} \cdot x^{n-1} + d_{n-2} \cdot x^{n-2} + ... + d_0 \cdot x^0$$

Exemplu: $01011 = x^3 + x + 1$ (Polinom de ordinul 3). Ordinul polinomului se determină după puterea cea mai mare a lui x cu coeficientul 1.

Aritmetica polinomială

Aritmetica polinomială, utilizată pentru efectuarea operațiilor matematice asupra codurilor ciclice este o aritmetică unde nu se ia în considerație transportul.

Adunarea şi scăderea.

Reg. de adunare:	Reg. de scădere:	Exemple:		
$0+0=0$ $0+1=1$ $1+0=1$ $1+1=0^*$	$0-0=0$ $0-1=1^*$ $1-0=1$ $1-1=0$	$+\begin{array}{r}1010\\ \underline{1111}\end{array}\oplus$ 0101	$-\begin{array}{r}1010\\ \underline{1111}\end{array}\oplus$ 0101	10101101 00101000 +,− 11101011 01101110

În aritmetica polinomială $x+x=0$, deoarece $x+x=(1+1)\cdot x=0\cdot x=0$.

Înmulţirea

Înmulțirea polinomului la x^i coincide cu deplasarea la stânga cu i biți.

$x^3\cdot(x^3+x+1)=x^6+x^4+x^3$

$1000\cdot(1011)=1011000$

Înmulțirea unui polinom la altul constă din 2 etape: 1. Determinarea produselor intermediare conform regulilor aritmeticii clasice. 2. Adunarea produselor intermediare conform regulilor aritmeticii polinomiale.

$(x^4+x^2+x+1)\cdot(x^3+x+1)=x^7+1$

			1	0	1	1	1
				1	0	1	1
			1	0	1	1	1
		1	0	1	1	1	⊕
1	0	1	1	1		⊕	
1	0	0	0	0	0	0	1

Împărţirea

Împărțirea se efectuiază conform regulilor aritmeticii clasice cu excepția operațiilor de scădere, care se înlocuesc cu adunarea mod 2. Împărțirea se efectuiază până când ordinul restului nu va fi mai mic decât ordinul împărțitorului.

1	1	1	1	0	0	0	1	0	0	1
1	0	0	1				1	1	1	0
	1	1	0	0						
	1	0	0	1						
		1	0	1	0					
		1	0	0	1					
				1	1	0	- restul			

Deplasarea ciclică la st'nga eechivalentă cu înmulțirea la x şi împărțirea la x^n+1.

$1101 \Rightarrow x^3+x^2+1$

$x^n+1=x^4+1$

$x\cdot(x^3+x^2+1)=x^4+x^3+x$

$x^4+x^3+x \quad \big| \underline{x^4+1}$

$\underline{x^4 \qquad +1}$

$x^3+x+1 \Rightarrow 1011$

Polinoame generatoare

În codurile ciclice fiecare cuvânt de cod este multiplu al unui şi acelui polinom d eordin n-n_0=k, numit polinom generator g(x).

g(x) este un polinom primitiv (nu poate fi reprezentat în formă de produs a altor polinoame de ordin mai mic). În orice g(x) coeficientul lui x_0 este egal cu 1. Este o condiție de formare a polinomului primitiv. Fiecare cuvânt de cod valid se împarte fără rest doar la g(x). Orice alt cuvânt interzis se va împărți cu rest. Valoarea restului permite detectarea şi corectarea erorilor.

$x+1$	11	3
x^2+x+1	111	7
x^3+x+1	1011	11
x^3+x^2+1	1101	13
x^4+x+1	10011	19

Formarea cuvântului de cod (codarea)

Etapele:

1. Se efectuiază înmulțirea $\boldsymbol{x^k d(x)}$, unde k=n-n_0 este numărul biților de control, care coincide cu ordinul lui g(x). Operația este echivalentă cu completarea a k zerouri din dreapta.

2. La produsul $\boldsymbol{x^k d(x)}$ se adună restul r(x), obținut în urma împărțirii $\boldsymbol{x^k d(x)}$ la g(x). $\boldsymbol{F(x) = x^k d(x) + r(x)}$.

Explicație:

$$\frac{\boldsymbol{x^k d(x)}}{\boldsymbol{g(x)}} = \boldsymbol{Q(x)} + \frac{\boldsymbol{r(x)}}{\boldsymbol{g(x)}} \quad \Big| \boldsymbol{inmultim\ la\ g(x)}$$

$$\boldsymbol{x^k d(x) = Q(x) g(x) + r(x)}$$

Aici $\boldsymbol{F(x) = Q(x) g(x)}$ este un cuvânt de cod valid fiind multiplu al lui g(x).

$\boldsymbol{F(x) = Q(x) g(x) = x^k d(x) + r(x)}$, operațiile de adunare şi scădere fiind identice.

Numărul biților de control k se determină în felul următor:

Pentru corectarea unei erori$(\boldsymbol{d_{min} = 3})$: $\boldsymbol{k = [\log_2(n+1)]}$, $\boldsymbol{n = 2^k - 1}$, $\boldsymbol{n + 1 \le 2^k}$

n=7 k=3 n_0=4
n=15 k=4 n_0=11
n=31 k=5 n_0=26

Pentru $d > 3$

d- impar: $k = [\frac{d-1}{2} \log(n+1)]$, d- par: $k = [\frac{d-2}{2} \log_2(n+1)]$

Pentru alegerea g(x), ordinul polinomului trebuie să fie mai mare sau egal cu k., numărul unităților din g(x) – mai mare sau egal cu d_{min}.

Exemplu. Să se obțină un cuvânt de cod pentru transmiterea a 7 biți, cu corectarea unei erori singulare.

1. Se determină k. $k = \log_2(7+1) = 3$
2. se determină n_0. $n_0 = n - k = 4$

Deci obținem un cod ciclic (7,4).

3. Alegem g(x). Ordinul polinomului ≥ 3 (k)

Numărul unităților ≥ 3 (d_{min})

$g(x) = x^3 + x + 1 \rightarrow 1011$

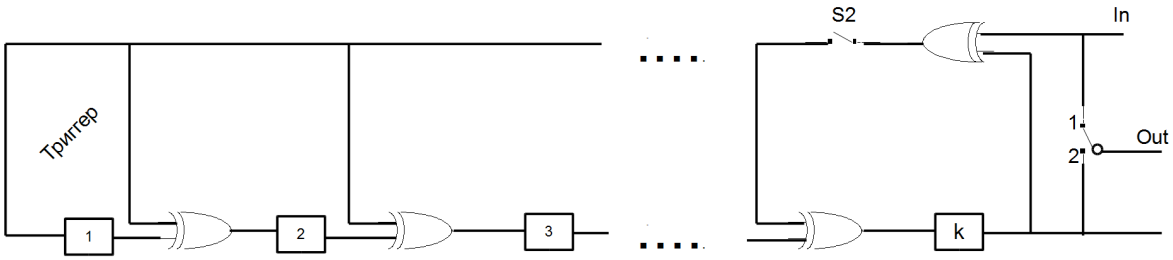
4. $d(x) = 1101$ blocul de date cu polinomul corespunzător $x^3 + x + 1$
5. Înmulțim la x^k 1000 (1101) = 1101000
6. Efectuăm împărțirea la g(x)

```
1101000 | 1011
1011    | 1111
------
 1100
 1011
------
  1110
  1011
------
  1010
  1011
------
     1   –  r(x)
```

7. Formăm cuvântul de cod: F(x) = 1101001

Obținerea restului se efectuiază pe un registru de k-biți cu deplasare stânga şi linii de reacție cu elemente XOR.

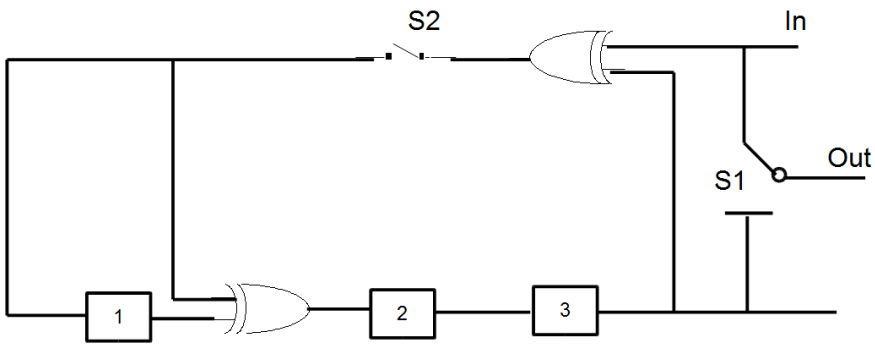
Circuitul de codare pentru un polinom generator de ordin k cu coeficienți nenuli:



Numărul elementelor XOR este egal cu numărul unităților din g(x) fără cea mai semnificativă, deoarece cei mai semnificativi biți din d(x) şi g(x) sunt întotdeauna egali cu 1 iar $1 \oplus 1 = 0$.

La început întrerupătorul S1 se află în poziția 1, S2 e închis. Cuvântul format din n_0 biți informaționali este transferat spre ieşire şi, în acelaşi timp, în registru. Timp de n_0 impulsuri de ceas se formează restul r(x), care şi reprezintă biții de control. După aceasta S2 sw deschide, S1 trece în poziția 2 şi biții de control sunt transferați spre ieşire timp de k impulsuri de ceas.

Pentru $g(x) = x^3 + x + 1$ $\overline{1101}$



	CLK	In	Rg 1 2 3	Out	
	0	-	0 0 0	-	x_{n-1}
n_0	1	1	1 1 0	1	
	2	1	1 0 1	1	
	3	0	1 0 0	0	
	4	1	1 0 0	1	
K	5	-	- 1 0	0	x_0
	6	-	- - 1	0	
	7	-	- - -	1	

Detectarea şi corectarea erorilor

1.Împărțirea cuvântului de cod recepționat F´(x) la g(x), formând sindromul erorii s(x). Dacă s(x)=0 – cuv'ntul a fost transmis fără erori. Dacă s(x)≠0 – a avut loc o eroare.

2. Pentru a corecta eroarea, numărul unităților în s(x) trebuie să fie egal cu numărul erorilor. Dacă numărul unităților este mai mare, se efectuiază deplasarea ciclică cu o poziție la stănga a F´(x) şi se repetă p.1 până când condiția devine adevărată.

3. Se efectuiază adunarea modulo2 dintre ultimul deâmpărțit şi ultimului rest.

4. Se efectuiază deplasarea inversă.

Exemplu.

F(x) = 1101001

Cuvăntul transmis F’(x) = 0101001

1)

```
0101001   | 1011
1011      |------
-------
   101 – r(x)     w = 2  2>1
```

2) $F''(x) = \overleftarrow{F'}^{1}(x)$

```
1010010 | 1011
1011    |------
-------
  1010
  1011
-------
     1  -  r(x) w = 1  1= 1
```

3)

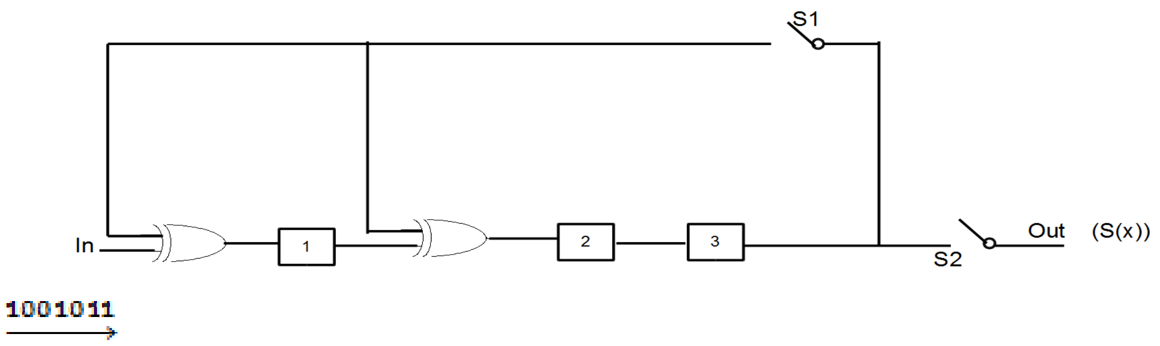
$F''(x) = 1010010 \quad \oplus$

$F_{cor}(x) \quad \dfrac{1}{1010011}$

4) $F(x) = \overrightarrow{F_{cor}}^{1}(x)$

$F(x) = 1101001$

Circuitul de calcul al sindromului:



S2 inițial e închis, s2 - deschis. După 7 impulsuri de ceas, registrul va conține sindromul erorii. S2 se deschide iar s1 se închide.

In	CLK	Rg 1	Rg 2	Rg 3	
1001011	0	0	0	0	
100101	1	1	0	0	
10010	2	1	1	0	
1001	3	0	1	1	
100	4	0	1	1	
10	5	1	1	1	
1	6	1	0	1	
-	7	0	0	0	S(x)

Proprietățile polinomului generator

g(x) = CRC

CRC – 1 11 x+1 Bit de paritate k=1

CRC – 8 (9 biți)

CRC – 16 (17 bişi) 16, 12, 5, 0 - XMODEM, Bluetooth

CRC – 32 (33 biți) 32, 26, 23, 22, 16, 12, 11, 10, 8, 7, 5, 4, 2, 1, 0 - Ethernet, MPEG

CRC – 128 MD5 Message Digest Algorithm 5 (în criptografie)

CRC – 160 SHA Secure Hash Algorithm (în criptografie)

Proprietățile

1.g(x)va detecta toate erorile singulare dacă cel puțin 2 coeficienți sunt nenuli.

$E(x) = x^i$ g(x) = x+1 $\frac{x^i}{x+1}$ va da întotdeauna rest

2.Erorile duble vor fi detectate în cazul când g(x) are cel puțin 2 termeni şi nu-l divide pe

$x^{i-j} + 1$ (11, 101, 1001,…)

$E(x) = x^i + x^j = x^j(x^{i-j} + 1), \quad i > j \text{ и } i - j < k$

3.Dacă g(x)are un număr par de termeni , codul va detecta toate modelele de erori cu număr impar. Această propriettae asigură capacitatea de a detecta jumătate din toate modelele posibile de erori.

4.g(x)detectează pachete de erori de lungime $l = n - n_0$, $(l \le k)$.

Polinomul corespunzător unui pachet de erori de lungime *l* se poate scrie ca $x^i p(x)$, unde $p(x)$ are ordinul maxim $l-1<n-n_0$ şi nueste divizibil prin g(x)de ordin $n-n_0$.

Exemplu. Să se obțină un cuvânt de cod pentru transmiterea a 15 biți cu capacitatea de detectare şi corectare a unei erori singulare.

$d_{min} = 3$

1)$k = \log_2(n+1) = 4$

2)$n_0 = n - k = 15 - 4 = 11$

3)$g(x) = 10011$

4)$d(x) = 00000000101$

5)$x^4 d(x) = 000000001010000$

6)

```
000000001010000 | 10011
  10011         -------
  -----           11100
    11100         10011
    10011         -----
    -----          1111
     1111
```

7)$F(x) = 000000001011111$

$F_e(x) = 010000001011111$

8) $F_e(x)/g(x) = S(x)$

```
010000001011111 | 10011
 10011
 -----
  11001
  10011
  -----
   10100
   10011
   -----
    11111
    10011
    -----
     11001
     10011
     -----
      10101
      10011
      -----
       1101
```

$S(x) = 1101 \quad W = 3 \quad S = 1$

9) $\overleftarrow{F_e(x)}^{1}$

```
100000010111110 | 10011
 10011
 -----
  11001
  10011
  -----
   10100
   10011
   -----
    11111
    10011
    -----
     11001
     10011
     -----
      10101
      10011
      -----
       11010
       10011
       -----
        1001
```

$S(x) = 1001 \quad W = 2 \quad S = 1$

10) $\overleftarrow{F_e(x)}^{2}$

```
000000101111101 | 10011
 10011
 -----
  11001
  10011
  -----
    001
```

$S(x) = 0001 \quad W = 1 \quad S = 1$

11)

```
000000101111101  ⊕
1
-----------------
000000101111100
```

12) F(x) = $\overrightarrow{F_e(x)}^{2}$ 000000001011111

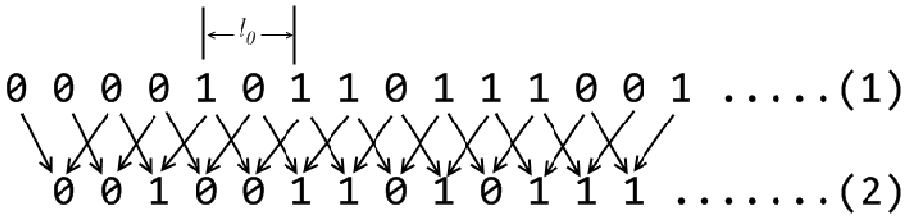
Coduri convoluționale (recurente)

Codurile convoluționale sunt coduri continue, spre deosebire de codurile bloc. Operațiile de codare şi decodare au loc în mod continuu.

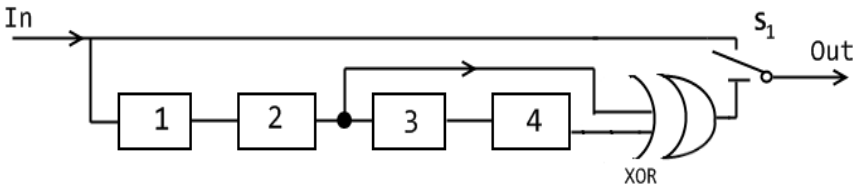
Se notează (n_0/n).

Cel mai simplu cod convoluțional este codul (1/2), unde după fiecare simbol informațional urmează unul de control. Într-un asemenea cod n_0=k=n/2. Redundanța codului este de 50%.

Simbolurile de control se formează prin adunarea modulo 2 dintre două simboluri informaționale situate la o distanță l_0 unul de altul.



Circuitul de codare:



Comutatorul S1 transmite la ieşire un symbol informațional, apoi unul de control, formând şirul:

1 0 0 0 1 1 1 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 (3)

↑inf. ↑ de control

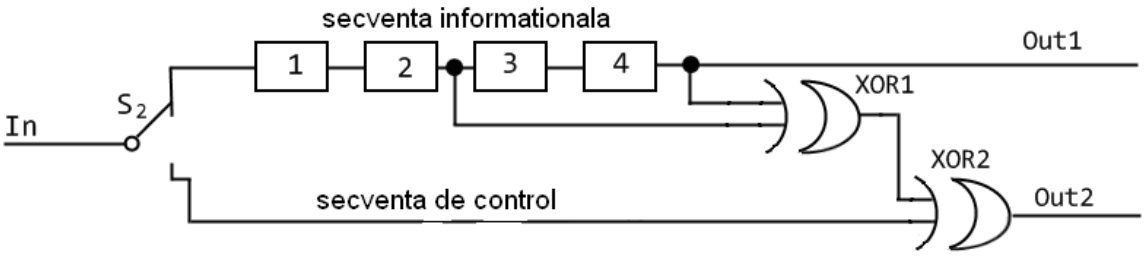
Formarea simbolurilor de control în circuitul de codare:

Perioada T_i	***Rg***				***XOR***
	1	***2***	***3***	***4***	
1	*1*	*0*	*0*	*0*	*0*
2	*0*	*1*	*0*	*0*	*0*
3	*1*	*0*	*1*	*0*	*1*
4	*1*	*1*	*0*	*1*	*0*
5	*0*	*1*	*1*	*0*	*0*
6	*1*	*0*	*1*	*1*	*1*
7	*1*	*1*	*0*	*1*	*1*
8	*1*	*1*	*1*	*0*	*0*
9	*0*	*1*	*1*	*1*	*1*
10	*0*	*0*	*1*	*1*	*0*
11	*1*	*0*	*0*	*1*	*1*

Circuitil de decodare constă din două părți:

1. Circuitul de formare a secvenței de corectare
2. Circuitul de corectare

Circuitul de formare a secvenței de corectare:



Comutatorul S2 funcționează sincron şi sinfaz cu comutatorul S1. Secvența (3) este aplicată laintrarea circuitului, iar comutatorul S2 o împarte în secvență informațională şi secvență de control.

Secvența informațională este transmisă la registrul de deplasarea, iar cea de control – la elementul XOR2. Deoarece structura registrului este similară cu cea din circuitul de codare, la elementul XOR1 se formează încă o dată secvența biților de control. În lipsa erorilor ,la ieşirea OUT2 se formează o secvență de corectare formată din zerouri, iar la OUT1 se formează secvența simbolurilor informaționale.

În cazul erorilor, secvența de corectare de la OUT2 va conține unități în anumite poziții.

Codul cercetat permite corectarea pachetelor de erori cu lungime $\boldsymbol{l \le 2l_o = 4}$.

Presupunem că au fost eronați 4 bițidin cuvântul de cod. 2 biț ieronați vor fi infomaționali şi 2 – de contol.

Începem analiza cu momentul când la intrarea circuitului de decodare apare primul bit eronat, după care urmează tot pachetul cu lungime $2l_0$=4. În registru, până la acest moment, se conțin biții corecți. Deaceea, primii l_0=2 paşi de deplasare în registru vor forma la OUT2 poziția erorilor în biții de control.

În continuare, secvența de control va conține biți neeronați.

Următorii 2 paşi formează la fel la OUT2 două unități, dar acum din cauza biților informaționali eronați din prima jumătate a registrului (1,2). Următorii 2 paşi vor forma la OUT2 unități, dar acum din cauza aceloraşi biți eronați aflați în pozițiile 3 şi 4 ale registrului.

Secvența de la OUT2 va conține:

1. Unități în pozițiile eronate a biților de control (paşii 1 şi 2);
2. Unități în pozițiile eronate a biților informaționali cu o deplasare de 2 biți (paşii 3 şi 4);
3. Unități în pozițiile eronate a biților informaționali cu o deplasare de 4 biți (paşii 5 şi 6);

Să presupunem că a avut loc un pachet de erori:

0**1101**00000000 (unitățile arată pozițiile unde au avut loc erori)

În acest caz secvența recepționată va fi:

1 **1** **1** 0 **0** 1 1 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 (4)

Comutatorul S2 va împărți secvența (4) în secvența informațională (5) şi cea de control (6).

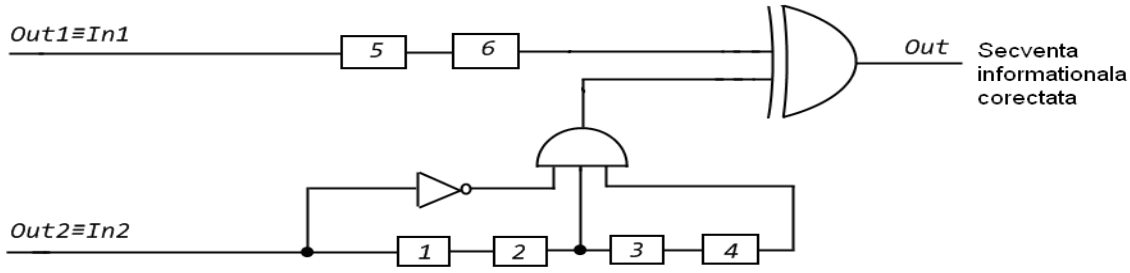
In$_1$ 1**10**10111001 (5)

In$_2$ **1**0100110101 (6)

Funcționarea circuitului de formare a secvenței de corectare:

$In_1(5)$	Rg				XOR	$In_2(6)$	Out_2 (secvența de corectare)
	1	2	3	4			
1	1	0	0	0	0	1	1
1	1	1	0	0	0	0	0
0	0	1	1	0	1	1	0
1	1	0	1	1	1	0	1
0	0	1	0	1	1	0	1
1	1	0	1	0	0	1	1
1	1	1	0	1	0	1	1
1	1	1	1	0	0	0	0
0	0	1	1	1	1	1	0
0	0	0	1	1	0	0	0
1	1	0	0	1	1	1	0

Operația de corectare automată a erorilor se efectuiază în circuitul de corectare:



La elementul ŞI se aplică:

Out2 1 0 0 1 1 1 1 0 0 0 0

1. secv. de corectare inversată..... 0 1 1 0 0 0 0 **1 1** 1 1 . .
2. secv. de corectare deplasata cu 2 biți..... . . 1 0 0 1 1 **1 1** 0 0 .. .
3. secv. de corectare deplasata cu 4 biți..... . . . . 1 0 0 **1 1** 1 1 . .

poarta ŞI 0 0 0 **1 1** 0 0

Unitatea la ieşirea porții ŞI este semnalul de corectare a erorii. Pentru corectarea biților eronați, secvența informațională se deplasează cu 2 poziții (celulele 5 şi 6) şi se adună modulo 2 cu secvența de corectare.

0001100000000 ⊕
11010111001
10110111001

Secvența informațională tece prin 6 celule $3l_0=6$ ale registrului de deplasare. Pe lângă aceasta pentru corectarea tuturor biților eronați este necesar un interval de protecție (6 biți neeronați cu lungimea de $6l_0+1=13$ biți).

Pentru creşterea capacității de corectare şi a pachetului de erori corectate se măreşte n (se utilizează coduri n-1)/n.

Unul din neajunsurile codurilor convoluționale este necesitatea de a avea un interval de timp cu simboluri neeronate, după ce este recepționat pachetul de erori.

Ex. Sa se efectuieze corectarea secv. inf. pentru şirul 10101111000001110011 00 (00**1111**0000...). Secv. inf: 11110001010