UNIVERSITATEA "POLITEHNICA"
TIMIŞOARA
BIBLIOTECA CENTRALĂ

Nr. Inv. ______________
Dulap 39 Lit. A

L EDUCATIEI NATIONALE
A "POLITEHNICA" DIN TIMISOARA
)E AUTOMATICA SI CALCULATOARE

TEZA DE DOCT(RA

"Contribuții la testarea și creșterea fiabilității unui sistem de calcul și sinteza testabilă a echipamentelor de calcul cu fiabilitate sporită"

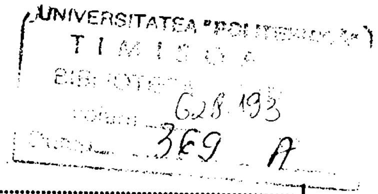
Conducător științific,
Prof.dr.ing.Mircea VLĂDUȚIU

BIBLIOTECA CENTRALĂ
UNIVERSITATEA "POLITEHNICA"
TIMIŞOARA

Doctorand,
Daniela Elena POPESCU

Timișoara, 1998

BUPT


UNIVERSITATEA "POLITEHNICA" TIMIŞOARA BIBLIOTECA Volum 628.193 Dulap 369 A

Cuprins

0. INTRODUCERE 1

0.1 DEPENDABILITATEA CA ŞI ATRIBUT DE EFICIENŢĂ A SISTEMELOR TEHNICE 1
0.2 STRUCTURA DE ÎNSUMARE CA ŞI OBIECTIV CU TESTARE DIFICILĂ 2
0.3 TESTAREA CA ŞI PROPRIETATE DE REALIZARE A STRUCTURILOR DE ÎNSUMARE DEPENDABILE 3
0.4 TESTABILITATEA CA ŞI CARACTERISTICĂ A STRUCTURILOR DE ÎNSUMARE VERIFICABILE 4
0.5 STRUCTURA TEZEI DE DIZERTAŢIE 5

1. ANALIZA FUNCŢIONALĂ A STRUCTURILOR DE ÎNSUMARE ŞI PROBLEMATICA DE TESTARE AFERENTĂ ACESTORA 7

1.1 STRUCTURI DE ÎNSUMARE CU PROPAGAREA SERIALĂ A TRANSPORTULUI 7
1.1.1 Sumator cu propagare în cascadă a transportului 7
1.1.2 Sumator Manchester 8
1.1.3 Sumator cu semnalizarea sfârşitului însumării 9
1.2 STRUCTURI DE ÎNSUMARE CU PROPAGAREA ANTICIPATĂ A TRANSPORTULUI 10
1.2.1 Sumator cu anticiparea transportului 10
1.2.2 Sumatorul cu anticiparea transportului pe blocuri de cifre binare 11
1.2.3 Sumatorul cu anticiparea transportului între blocuri de cifre binare 12
1.2.4 Sumatoare organizate pe superblocuri 13
1.2.4.1 Sumatorul cu anticiparea transportului între blocuri organizat pe două niveluri 13
1.2.4.2 Sumatorul cu anticiparea transportului între blocuri organizat pe două niveluri în variantă îmbunătăţită 15
1.2.4.3 Sumatorul cu anticiparea transportului pe blocuri organizat pe două niveluri 17
1.2.5 Sumator cu anticipare piramidală a transportului 17
1.3 STRUCTURI DE ÎNSUMARE CU OMITEREA TRANSPORTULUI 18
1.3.1 Sumator cu un nivel de omitere a transportului 18
1.3.2 Sumator cu mai multe niveluri de omitere a transportului 19
1.4 STRUCTURI DE ÎNSUMARE CU PROPAGARE SPECIALĂ A TRANSPORTURILOR 20
1.4.1 Sumator cu condiţionarea sumei 20
1.4.2 Sumator cu selecţia sumei 23
1.4.3 Sumator piramidal 27
1.5 PROBLEME ALE TESTĂRII STRUCTURILOR DE ÎNSUMARE 30
1.5.1 Modelarea defectelor 30
1.5.1.1 Modele de întârziere 35
1.5.2 Tipuri de testări 36
1.5.3 Generarea automată a testelor 39
1.5.3.1 Generarea pattern-urilor de test 40
1.5.3.2 Simularea defectelor 44
1.5.3.2.1 Metodologia de simulare 44
1.5.3.2.2 Simularea paralelă a defectelor 45
1.5.3.2.3 Simularea deductivă a defectelor 47
1.5.3.2.4 Simularea concurentă a defectelor 49
1.5.3.2.5 Simularea defectelor de întârziere 50
1.5.3.2.5.1 Detecţia robustă şi nerobustă a căilor ce prezintă defecte de întârziere 51
1.5.3.2.5.2 Simularea robustă şi nerobustă a defectelor de întârziere de pe căile de propagare a semnalelor 52
1.5.3.2.5.3 Codificarea valorilor logice 53
1.5.3.2.5.4 Simularea paralelă a căilor de întârziere 54
1.5.3.2.5.5 Detecţia căilor de întârziere 56
1.5.4 Modalităţi de evaluare a răspunsurilor 57
1.6 CONCLUZII 58

BUPT

2. EFICIENTIZAREA TESTĂRII STRUCTURILOR DE ÎNSUMARE 59

2.1 ANALIZA DE PERFORMANȚĂ LA PROPAGAREA SERIALĂ A TRANSPORTULUI 59
2.1.1 Analiza de performanță la sumatorul RCA 59
2.1.2 Analiza de performanță la sumatorul CCA 60
2.2 ANALIZA DE PERFORMANȚĂ LA PROPAGAREA ANTICIPATĂ A TRANSPORTULUI 61
2.2.1 Analiza de performanță la sumatoarele CLA, RCLA, BCLA 61
2.2.2 Analiza de performanță la sumatoare organizate pe superblocuri 63
2.2.2.1 Analiza de performanță la sumatorul SBCLA 63
2.2.2.2 Analiza de performanță la sumatorul ISBCLA 64
2.2.2.3 Analiza de performanță la sumatorul SRCLA 65
2.2.3 Analiza de performanță la sumatorul PyCLA 66
2.3 ANALIZA DE PERFORMANȚĂ LA PROPAGAREA CU OMITERE A TRANSPORTULUI 67
2.3.1 Analiza de performanță la sumatorul CSkA cu un nivel de omitere 67
2.3.2 Analiza de performanță la sumatorul CSkA multinivel 72
2.4 ANALIZA DE PERFORMANȚĂ LA STRUCTURILE DE ÎNSUMARE CU PROPAGARE SPECIALĂ A TRANSPORTULUI 74
2.5 EFICIENTIZAREA TESTĂRII PRIN C–TESTABILITATE LA STRUCTURILE DE ÎNSUMARE CU PROPAGARE SERIALĂ A TRANSPORTULUI 76
2.5.1 Testarea sumatorului RCA 76
2.5.2 Testarea sumatorului CCA 80
2.6 EFICIENTIZAREA TESTĂRII PRIN C–TESTABILITATE LA STRUCTURILE DE ÎNSUMARE CU PROPAGARE ANTICIPATĂ A TRANSPORTULUI 81
2.6.1 Testarea sumatorului CLA 81
2.6.2 Testarea sumatorului RCLA 84
2.6.3 Testarea sumatorului BCLA 86
2.6.4 Eficientizarea testării prin C testabilitate la structurile de însumare organizate pe supeblocuri 88
2.6.4.1 Testarea sumatorului SBCLA 88
2.6.4.2 Testarea sumatorului ISBCLA 89
2.6.4.3 Testarea sumatorului SRCLA 90
2.6.5 Testarea sumatorului PyCLA 90
2.7 EFICIENTIZAREA TESTĂRII PRIN C–TESTABILITATE LA STRUCTURILE DE ÎNSUMARE CU PROPAGARE CU OMITERE A TRANSPORTULUI 96
2.7.1 Testarea sumatorului CSkA cu un nivel de omitere 96
2.7.2 Testarea sumatorului CSkA cu două niveluri de omitere 99
2.8 EFICIENTIZAREA TESTĂRII PRIN C–TESTABILITATE LA STRUCTURILE DE ÎNSUMARE CU PROPAGARE SPECIALĂ A TRANSPORTULUI 102
2.8.1 Testarea sumatorului LR-CSIA 102
2.9 EVALUARE COMPARATIVĂ PRIN PRISMA INDICATORULUI DE PERFORMABILITATE A STRUCTURILOR DE ÎNSUMARE 104
2.10 CONCLUZII 107

3. RECONFIGURĂRI ALE STRUCTURILOR DE ÎNSUMARE ÎN SCOPUL FACILITĂRII TESTĂRII 108

3.1 RECONFIGURĂRI BIST ON LINE PENTRU TESTAREA STRUCTURILOR DE ÎNSUMARE 109
3.1.1 Aplicații ale codurilor detectoare corectoare de erori 112
3.1.2 Erori multiple la nivelul biților 115
3.1.3 Problematica construirii checker-elor de erori 116
3.1.4 Tehnici de proiectare pentru checker-ele de erori incorporate testabile 117
3.1.4.1 Checker-e de paritate testabile 118
3.1.4.2 Checker-e cu autotestare 118
3.1.4.3 Checker-e dublă calc 120
3.1.4.4 Checker-e de egalitate 121
3.1.4.5 Checker-e M din N 122
3.1.5 Realizarea circuitelor combinaționale autotestabile 123
3.1.6 Autotestarea circuitelor secvențiale 124
3.1.7 Proiectarea unui sistem TSC 125
3.1.8 Reconfigurări BIST ale structurilor de însumare bazate pe controlul parității 126
3.1.8.1 Sumator cu verificarea parității prin dublarea lui CY 128

BUPT

3.1.8.2 Sumator cu suma dependentă de CY 128
3.1.9 Reconfigurări BIST ale structurilor de însumare bazate pe codul sumă de control 130
3.1.10 Reconfigurări BIST ale structurilor de însumare bazate pe coduri combinate 132
3.2 RECONFIGURĂRI BIST OFF LINE PENTRU TESTAREA STRUCTURILOR DE ÎNSUMARE 137
3.2.1 Arhitecturi BIST off-line generice 137
3.2.2 Testarea exhaustivă 139
3.2.3 Testarea pseudoaleatoare 140
3.2.3.1 Generarea aleatoare a testelor 140
3.2.3.2 Reducerea măsurilor de testabilitate la problema generală a probabilităţilor circuitelor logice 143
3.2.3.3 Algoritm de rezolvare a Problemei generalizate a probabilităţilor pentru circuite logice 145
3.2.4 Testarea pseudoexhaustivă 148
3.2.4.1 Pattern-uri de pondere constantă 149
3.2.4.2 Segmentarea logică la nivelul structurilor de însumare 150
3.2.4.2.1 Testare bazată pe partiţionare logică prin activarea căilor la CSkA 150
3.2.4.2.2 Testare bazată pe partiţionare logică prin activarea căilor la structurile de însumare cu anticiparea transportului 152
3.2.4.3 Segmentarea fizică la nivelul structurilor de însumare 154
3.2.4.4 Testarea sumatoarelor cu ajutorul pattern-urilor de pondere constantă 155
3.2.4.5 Identificarea intrărilor semnal de test 157
3.2.4.6 Determinarea setului complet de teste pentru testarea pseudoexhaustivă a sumatoarelor 160
3.3 CONCLUZII 161

4. EVALUAREA REZULTATULUI TESTĂRII UTILIZÂND MEDIUL LDL 163

4.1 EVALUĂRI ALE SUMATOARELOR CU PROPAGAREA SERIALĂ A TRANSPORTULUI 164
4.2 EVALUĂRI ALE SUMATOARELOR CU PROPAGAREA ANTICIPATĂ A TRANSPORTULUI 165
4.2.1 Evaluarea sumatorului CLA 165
4.2.2 Evaluarea sumatorului RCLA 170
4.2.3 Evaluarea sumatorului BCLA 174
4.2.4 Evaluarea sumatoarelor organizate pe superblocuri 174
4.2.4.1 Evaluarea sumatorului SBCLA 174
4.2.4.2 Evaluarea sumatorului ISBCLA 175
4.2.4.3 Evaluarea sumatorului SRCLA 177
4.2.5 Evaluarea sumatorului PyCLA 178
4.2.6 Evaluări comparative ale sumatoarelor cu anticiparea transportului 180
4.3 EVALUĂRI ALE SUMATOARELOR CU PROPAGAREA CU OMITERE A TRANSPORTULUI 182
4.3.1 Evaluarea sumatorului CskA cu un nivel de omitere 182
4.3.2 Evaluarea sumatorului CSkA multinivel 183
4.4 EVALUĂRI ALE SUMATOARELOR CU PROPAGAREA SPECIALĂ A TRANSPORTULUI 184
4.4.1 Evaluarea sumatorului LR-CSIA 184
4.5 CONCLUZII 185

5. CONCLUZII 188

ANEXA 1 191

ANEXA 2 192

BIBLIOGRAFIE 234

BUPT

0. Introducere

0.1 Dependabilitatea ca şi atribut de calitate al sistemelor de calcul

Dependabilitatea (*dependability*) unui sistem exprimă calitatea serviciilor asigurate de acesta [JOHN85]. Conform aprecierilor majoritare din literatură, conceptul de dependabilitate acoperă pe cele de: fiabilitate, mentenabilitate, disponibilitate, siguranţă, performabilitate şi testabilitate.

1. ***Fiabilitatea*** este capacitatea ca un produs să-şi menţină calitatea (totalitatea proprietăţilor lui care-l fac corespunzător pentru utilizare potrivit destinaţiei) pe toată durata de utilizare, în condiţii de exploatare specificate.
2. ***Mentenabilitatea*** permite evaluarea capacităţii de redare a funcţionalităţii unui sistem defectat. Din punct de vedere cantitativ, mentenabilitatea, M(t), este dată de probabilitatea de repunere în stare operaţională a sistemului defect (prin acţiuni de mentenanţă) într-un interval de timp t specificat.
3. ***Disponibilitatea***, reprezintă aptitudinea unui sistem de a-şi îndeplini funcţia specificată, sub aspectele combinate de fiabilitate, mentenabilitate şi de management al acţiunilor de mentenanţă, la un moment dat, sau într-un interval de timp specificat; de fapt, disponibilitatea este afectată de două probabilităţi:
   - probabilitatea funcţionării pe o anumită durată;
   - probabilitatea ca, după apariţia unei căderi, capacităţii de bună funcţionare a sistemului să fie restabilită în decursul unui intrval de timp.
4. ***Siguranţa*** (*Safety*), S(t), este dată de probabilitatea ca un sistem să-şi realizeze corect funcţiile, sau în cazul defectării, să-şi întrerupă funcţionarea într-o manieră prin care să nu afecteze opeararea altor sisteme, sau să compromită siguranţa oamenilor.
5. ***Performabilitatea*** (*Performability*), P(L,t) unui sistem este o funcţie de timp definită ca şi probabilitatea ca la momentul *t* performanţa sistemului, să fie mai mare sau egală cu un anumit nivel L.
6. ***Testabilitatea*** (*Testability*), exprimă abilitatea de testarea a unor atribute interne sistemului.

Sistemele de calcul sunt caracterizate din punctul de vedere al disponibilităţii lor prin coeficientul de disponibilitate, exprimat după cum urmează:

$$A = \frac{MTTF}{(MTTF + MTTR)},$$

unde:

- $MTTF$ este timpul mediu până la defectare (*Mean Time To Failure*) - măsură a *fiabiliăţii*
- $MTTR$ este timpul mediu de reparaţie (*Mean Time To Repair*) - măsură a *mentenabilităţii*.

În acest context, cercetările prezente în domeniul sistemelor de calcul, vizează ansamblul de măsuri ce trebuiesc întreprinse, încă din faza de proiectare, pentru a asigura îmbunătăţirea măsurilor conceptelor mai sus definite, cu implicaţii directe în creşterea gradului de *dependabilitate* ca şi atribut complex al calităţii acestora.

Pentru a realiza o creştere a disponibilităţii sistemelor de calcul există două direcţii posibile de acţionare:

BUPT

a. Creştere, prin acţiuni specifice fiabilităţii a valorii componentei MTTF
În prezent, există două clase de metode de creştere a fiabilităţii:
- evitarea defectelor (*fault avoidance*) sau netolerarea defectelor *(fault intolerance)*
- tolerarea defectelor *(fault tolerance)*.

b. Micşorare, prin acţiuni specifice mentenabilităţii a valorii componentei MTTR. În acest sens, este preferabilă intervenţia cu precădere la nivelul proiectului tehnologic, în sensul înzestrării schemelor cu mijloace menite să crească testabilitatea proiectului, acţionându-se astfel favorabil în sensul micşorării timpilor medii de detecţie, respectiv de diagnoză a defectelor.

0.2 Structura de însumare ca şi obiectiv cu testare dificilă

Lucrarea aparţine domeniului de calcul, situându-se la joncţiunea domeniului de calcul cu cel al fiabilităţii; mai exact, ţinta cercetărilor este o parte a echipamentului de calcul, şi anume, unitatea de însumare a acestuia.
Interesul meu a fost focalizat în această direcţie din mai multe motive:

1. În primul rând datorită faptului că performanţa de operare a acestor unităţi este hotărâtoare în determinarea performanţei de operare a sistemului de calcul în ansamblu; aceasta, întrucât operaţiile aritmetice de însumare sunt operaţii cu frecvenţă mare cele mai frecvente; ele sunt executate nu numai când sunt comandate în mod explicit de execuţia instrucţiei de adunare, ci ele pot fi implicate pentru calculări de adrese, incrementări de registre, etc. Fac referire în acest sens la rezultatele mix-ului de instrucţiuni DLX obţinute prin programele benchmark SPEC92 [PATT96].
   - 5 programe *benchmark* SPECInt92, care indică pentru programul Espresso un procent de utilizare al adunărilor de 23,8%
   - 5 programe *benchmark* SPECfp92, care indică pentru programele Doduc şi Ear un procent de utilizare al adunărilor de 13,6%.
2. O a doua motivaţie este dată de faptul că din punct de vedere al testării, structurile de însumare se constituie ca şi o clasă aparte a circuitelor combinaţionale, la care pe parcursul procesului de operare, semnalele sunt activate pentru propagare pe două direcţii:
   - O direcţie dată de semnalele de însumare
   - O a doua direcţie stabilită de propagarea semnalelor de transport
3. A treia motivaţie este conferită de importanţa deosebită din punctul de vedere al fiabilităţii sistemelor a realizării unui autocontrol eficient al operării la acest nivel. În acest sens se impun o serie de reconfigurări la nivelul structurilor, pentru care s-au dat soluţii în cadrul lucrării.

Ţinând cont de importanţa la nivelul sistemelor de calcul a funcţiei implementate de aceste structuri în contextul dinamicii tehnologice la care asistăm, se justifică din plin eforturile concentrate în direcţia creşterii performanţei lor operaţionale în conjuncţie cu cea a performanţele lor de testabilitate. Pentru calificarea globală a performanţelor structurilor de însumare în raport cu cele două aspecte mai sus menţionate am introdus un indicator de performabilitate pe care l-am utilizat pentru evaluarea comaparativă a structurilor de însumare.
Astfel, pe baza motivaţiei prezentate, pornind de la o analiză funcţională ce s-a constituit ca şi temelie a investigaţiilor ulterioare, am aplicat o serie de strategii şi tehnici prin care mi-am adus contribuţia la creşterea fiabilităţii şi a capacităţii de testare pentru această parte a calculatoarelor.

BUPT

0.3 Testarea ca şi proprietate de realizare a structurilor de însumare dependabile

În ultimul deceniu complexitatea circuitelor a mărit costul testării. Această creştere nu apare numai din cauza măririi circuitului, ci şi din cauza modalităţii în care a fost proiectat, în sensul că în faza de proiectare accentul a fost pus pe implementarea funcţiilor circuitului neglijându-se modul în care se va face testarea lui ulterioară.
Astfel, indiferent de dimensiunea circuitului ce se testează, dificultăţile legate de testarea lui sunt legate de următorii factori:

- Imposibilitatea iniţializării elementelor de memorare din circuit la o stare cunoscută; pentru atingerea acestui deziderat în trecut au fost concentrate o mare cantitate de eforturi de cercetare spre problema generării secvenţelor de iniţializare, al căror obiectiv era să aducă circuitul într-o stare cunoscută indiferent de starea lui prezentă.
- Imposibilitatea controlării şi observării semnalelor care se află în nodurile interioare ale circuitului ce se testează
- Realizarea funcţiilor logice cu ajutorul tranzistoarelor de trecere care nu pot fi modelate în mod convenţional în porţi echivalente
- Testarea plăcilor realizate în noile tehnologii nu permite utilizarea patului de cuie şi, în general, a metodelor de testare în circuit. Aceasta se datorează geometriei lor micşorate şi a sensibilităţii lor ridicate.

Recent, au evoluat câteva tehnologii de proiectare ale căror obiectiv este să reducă costul generării configuraţiilor binare de test (*patterns*) precum şi pe cel al testării în general. Pentru aceasta se intervine în următoarele direcţii:

1. Realizarea mai accesibilă a nodurilor interne ale circuitelor
2. Transformarea pentru testare a circuitelor secvenţiale în unele combinaţionale şi/sau descompunerea circuitelor complexe în unele mai puţin complexe.
3. Realizarea autotestabilă a circuitelor (*Self-testing*)
4. Reducerea cantităţii de date de test necesare pentru testarea circuitelor

Dar, cu toate că metodologiile de proiectare menite a îmbunătăţi testabilitatea unui circuit sunt foarte atractive, ele afectează negativ libertatea proiectantului, precum şi performanţele circuitului. Din moment ce majoritatea metodelor apelează la hardware adiţional, creşte mărimea fizică a circuitului, având un efect negativ asupra producţiei lui; hardware-ul adiţional introduce întârzieri suplimentare semnalelor, lucru care afectează performanţele circuitului. Apare de asemenea necesitatea pinilor adiţionali de intrare/ieşire, care măresc costul de împachetare.
Prin urmare, având în vedere efectele negative relevate, pentru ca un anumit stil de proiectare pentru testabilitate să fie acceptat, el trebuie:

- Să fie uşor de aplicat şi cu cât mai puţine restricţii care să nu inhibe ingeniozitatea proiectantului.
- Să aibă suportul software adecvat, adică programe care să verifice la un proiect respectarea regulilor de proiectare impuse de un anumit stil
- Să determine o reducere considerabilă a costurilor de generare a experimentului de testare, în condiţiile respectării performanţelor circuitului şi a investiţiei în efort de proiectare

In general, proiectarea pentru testabilitate (*Design for Testability DFT*) implică modificarea circuitului pentru îmbunătăţirea procesului generării *pattern*-urilor de test precum şi a celui de aplicare a acestora. Aceste tehnici de proiectare pentru testabilitate pot fi împărţite în trei clase:

1. metode de circumstanţă (*ad-hoc*)

BUPT

2. metode structurate
3. metode de încorporare a necanismelor de implementare a testării (*Built-in test*)

Prima clasă de astfel de metode, cele de circumstanță (de genul inserării punctelor de test, al utilizării logicii de blocare), nu au evoluat pentru a rezolva testarea unui circuit complex, ci doar pentru a rezolva o anumită problemă de testare.

În contrast, metodele structurate sunt mai formale; ele nu se introduc ulterior, ci sunt încorporate în proiect încă din faza lui de început.

Metodele de testare încorporată încearcă să reducă cantitatea datelor de test care trebuiesc transformate şi procesate, fiind deosebit de utile mai ales în cazul testării circuitelor complexe.

În majoritatea circuitelor cu autoverificare disponibile azi, ieşirile ciruitului sunt codificate printr-un cod detector de erori. Dacă într-un circuit apare o defecțiune, atunci sunt afectate ieşirile acestuia, şi se poate observa această schimbare. În unele cazuri o defecțiune poate afecta mai multe ieşiri, iar codul va apărea ca fiind corupt. În situația cea mai defavorabilă o defecțiune poate afecta aproape toate ieşirile, iar construcția codului va deveni foarte dificilă. Pentru a limita această problemă proiectanții se limitează la circuite cu autoverificare (*self-checking*) în care o defecțiune poate afecta mai multe ieşiri, dar toate schimbările de ieşiri sunt în aceeaşi direcție. Codurile numite detectoare de erori unidirecționale detectează astfel de erori.

0.4 Testabilitatea ca şi caracteristică a structurilor de însumare verificabile

Testul îl putem defini ca fiind un mijloc prin care se determină existența şi se califică anumite atribute la nivelul unui sistem. De exemplu, dacă o structură de însumare este proiectată să execute 1 000 000 adunări/sec., este de dorit să se identifice un test care să poată certifica acestă viteză de execuție. Astfel, putem defini testabilitatea în acest context ca fiind dată de abilitatea de certificare a anumitor atribute la nivelul sistemelor. Ea este un concept abstract ce operează cu o serie de costuri asociate testării.

Prin urmare, testabilitatea este o caracteristică de proiectare ce influențează diferite costuri asociate testării. Complexitatea testării poate fi convertită în costuri asociate procesului de testare, care sunt determinate de costuri de generare a testelor, costuri de simulare a defectelor şi de generare a informațiilor legate de localizarea defectului, costuri ale echipamentelor de test, precum şi costuri legate de procesul de testare în sine, adică de detectare şi localizare a unui defect. Întrucât aceste costuri pot fi mari şi ele pot depăşi chiar costul de proiectare, este foarte important ca ele să fie ținute în limite rezonabile prin procesul de proiectare pentru testabilitate.

Crescând testabilitatea unui circuit în mod automat se reduc anumite funcții ale acestor costuri şi nu neapărat fiecare cost în mod individual. De exemplu, proiectele *Scan* reduc costul generării testelor, dar ele cresc numărul pinilor de I/O, aria de integrare şi timpul de testare.

Cele mai multe tehnici DFT acționează prin diferite modalități în sensul îmbunătățirii controlabilității, observabilității şi predictibilității.

- *Controlabilitatea* se defineşte ca fiind dată de abilitatea de stabilire prin setarea valorilor de intrare ale unui circuit a unei valori specifice la nivelul fiecărui nod intern a circuitului.
- *Observabilitatea* este abilitatea de determinare a valorii de semnal a oricărui nod a circuitului prin controlarea intrărilor circuitului şi prin observarea ieşirilor sale.

BUPT

- *Predictibilitatea* este abilitatea de obţinere a unor valori cunoscute ale ieşirlor ca şi răspuns la stimuli de intrare daţi.

Deseori gradul de controlabilitate şi observabilitate a unui circuit este măsurat cu referire la modalitatea în care se face generarea testelor: aleator, sau în manieră deterministă prin utilizarea unui algoritm de generare a testelor. În general un nod intern al unui circuit are o slabă controlabilitate şi pentru stabilirea stării sale el implică utilizarea unui *pattern* unic de test. Astfel, termenul de *controlabilitate aleatoare* se referă la conceptul controlabilităţii când se utilizează teste aleatoare. În general un nod intern a unui circuit are o slabă controlabilitate aleatoare dacă el implică utilizarea unui *pattern* de test unic pentru determinarea stării sale. Pe de altă parte un nod are o slabă controlabilitate dacă este necesară o secvenţă lungă la intrările sale pentru a-i stabili starea.

Printre metodele DFT de tip ad-hoc se numără tehnica adăugării punctelor de test la nivelul nodurilor interne ale circuitelor, crescând astfel controlabilitatea şi observabilitatea acestor noduri ce au fost detectate în prealabil ca având valori mici ale acestor indicatori. La cealaltă extremă se situează metodele DFT cu caracter structurat, mai general aplicabile în comparaţie cu primele, aici amintim metodele de partiţionare fizică a circuitelor în vederea aplicării unei testări pseudoexhaustive. Tehnicile mai sus amintite se pretează să fie aplicate structurilor de însumare pentru creşterea testabilităţii acestora.

0.5 Structura tezei de dizertaţie

Prezenta teză de doctorat abordează aspectul creşterii testabilităţii şi disponibilităţii structurilor aritmetice de însumare, valorificând avantajele oferite prin utilizarea autocontrolului şi prin exploatarea C-testabilităţii acestora. Lucrarea cuprinde 6 capitole cu două anexe şi o listă bibliografică.

Precizez că în vederea unei prezentări unitare a lucrării, pentru referirea elementelor, structurilor şi tehnicilor uzitate am folosit abrevieri ale unor termeni de specialitate, consacraţi în limba engleză; aceasta deoarece în limba română nu există o terminologie consacrată. Astfel, acolo unde a fost cazul, am dat denumirea tradusă în limba română ce corespunde denumirii consacrate din limba engleză urmată de abrevierea consacrată în literatura de specialitate, ce corespunde denumirii în limba engleză.

După o scurtă introducere, capitolul 1 cuprinde o descriere funcţională a structurilor de însumare împreună cu unele precizări cu privire la terminologia uzitată, precum şi o trecere în revistă a modelelor, metodelor şi tipurilor de testări ce pot fi abordate pentru testarea acestor structuri. Se insistă asupra prezentării procesului de generare automată a testelor, şi asupra etapelor sale. Sinteza originală rezultată constituie punctul de plecare al construcţiei acestei lucrări.

Pe baza analizei funcţionale a structurilor de însumare s-au elaboarat în cadrul capitolului 2 relaţiile de calcul pentru performanţele lor de operare în termenii numărului maxim de porţi traversate de semnalele activate în cadrul procesului de însumare. Tot în cadrul acestui capitol, exploatând proprietatea de repetabilitate spaţială a structurilor de însumare şi aplicând conceptul C-testabilităţii – realizând eventual (acolo unde este cazul) o reconfigurare a acestora în vederea realizării dezideratului C-testabilităţii – în combinaţie cu cel al unei testări minimale la nivelul celulelor structurilor repetabile, s-a reuşit, printr-o analiză minuţioasă a structurilor, obţinerea unor algoritmi de generare automată a seturilor minimale pentru testarea lor completă în raport cu defectele de blocare singulară la 1 sau 0 (sunt prezentaţi în Anexa A). Pentru

BUPT

caracterizarea globală a structurilor de însumare este introdus şi definit indicatorul de performabilitate.

Capitolul 3 abordează problematica specifică autocontrolului bazat pe hardware. În prima parte a acestui capitol se tratează problematica autocontrolului *on-line*, precum şi cea a proiectării circuitelor de verificare cu proprietăţi speciale cum ar fi testabilitate, autotestabilitate şi altele, precum şi a circuitelor cu autoverificare totală, restrângându-se astfel domeniul de investigaţii. Pentru autocontrolul structurilor de însumare am dat soluţii de reconfigurare a structurilor de însumare ce să permită autotestarea lor (reconfigurări prin dublarea semnalelor de transport (CY), precum şi reconfigurări ce forţează dependenţa funcţiilor de însumare de valorile semnalelor de transport generate. Astfel, în a doua parte a acestui capitol este abordată problematica autocontrolului *off-line*, insistându-se asupra identificării numărului de *pattern*-uri necesare aplicării acestor tehnici la nivelul structurilor de însumare.

Capitolul 4 prezintă rezultatele experimentale obţinute, precum şi o analiză comparativă a acestora. Pentru validarea practică a algoritmilor optimali originali de generare automată a seturilor minimale de vectori de test ce realizează testarea completă în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare, a fost necesară realizarea de către autoare a unui mediu de lucru complex – intitulat LDL (*Logic Description Language*) – cu facilităţi pentru testarea circuitelor logice. În Anexa B este prezentată structura mediului, precizându-se şi facilităţile sale. Rezultatele obţinute de autoare sunt prezentate şi comentate în cadrul capitolului, pentru fiecare din structurile de însumare analizate.

Capitolul 5 conţine concluziile şi sinteza contribuţiilor originale aduse de autoare pe parcursul lucrării.

Autoarea tezei mulţumeşte în mod deosebit conducătorului ştiinţific, domnului prof.dr.ing. Mircea Vlăduţiu care, prin îndrumarea competentă într-o direcţie de un puternic interes actual, prin profesionalism în abordarea problemelor specifice şi în validarea soluţiilor alese, şi nu în ultimul rând prin răbdarea şi timpul acordat de-a lungul perioadei de pregătire, a contribuit decisiv la elaborarea tezei.
De asemenea adresează mulţumiri coordonatorilor şi contractorilor programului TEMPUS "EUROQUALROM", pentru oportunitatea de documentare în domeniu ce i-a fost acordată cu ocazia mobilităţii sale la laboratoarele TIMA Grenoble în vara anului 1998.
Autoarea mulţumeşte şi cadrelor didactice din catedra de calculatoare din Oradea şi din Timişoara, precum şi colegilor cu care a colaborat, pentru sugestiile şi recomandările primite.

Dedic această lucrare fiului meu Traian Mihai, dorind să însemne pentru el un îndemn în viaţă.

BUPT

1. Analiza funcțională a structurilor de însumare și problematica de testare aferentă acestora

In cadrul capitolului se abordează prezentarea analizei funcționale în conjucție cu o descriere structurală a structurilor de însumare în virgulă fixă pentru operanzi în reprezentare fără semn. Nu se insistă asupra modului de detectare a depășirilor de capacitate la însumare, întrucât această verificare este independentă de tehnica utilizată în implementarea sumatorului, ea fiind dependentă doar de modul în care se face codificarea operanzilor.

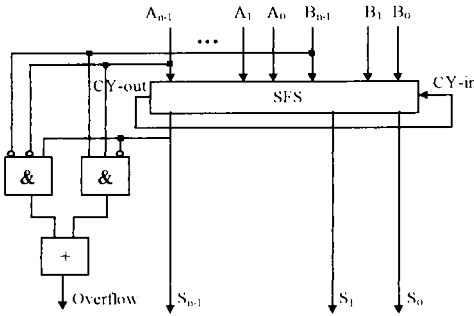


Fig.1. 1

În Fig.1. 1 este prezentat un sumator pentru numere reprezentate în complement de unu, cu detecție a *overflow*-ului. Însumarea se face cu ajutorul blocului sumator pentru operanzi fără semn – *SFS* , iar transportul ce se generează – *CY-out* – este recirculat, constituindu-se ca și transport inițial – *CY-in* – al sumatorului.

Sumatorul pentru însumarea numerelor în complement de doi va fi similar, cu diferența că acesta nu va avea recircularea transportului.

1.1 *Structuri de însumare cu propagarea serială a transportului*

1.1.1 Sumator cu propagare în cascadă a transportului

Sumatorul cu propagare în cascadă a transportului (*Ripple-Carry Adder* - RCA) realizează operația în paralel pentru 2 operanzi, A și B, de *n* cifre binare prin conectarea în lanț a *n* celule sumator complet (*Full Adder* - FA) astfel încât *CY-out* generat de FA al rangului *i* (C_i), devine *CY-in* pentru rangul *i*+*1*.

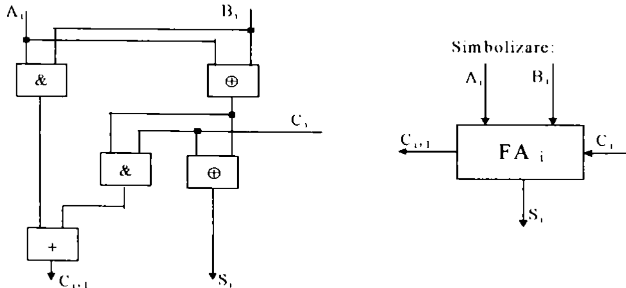


Fig.1. 2

Ecuațiile ce stau la baza implementării celulei sumator complet FA (Fig.1. 2), deduse pe baza tabelului de adevăr, sunt următoarele:

BUPT

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$
$$C_i = A_i \cdot B_i + (A_i + B_i)\cdot C_{i-1} = A_i \cdot B_i + (A_i \oplus B_i)\cdot C_{i-1} \quad (1.1)$$

Deci, CY se propagă prin întregul sumator, începând cu rangul cel mai puțin semnificativ, până la rangul cel mai semnificativ al sumatorului RCA. Registrele de intrare și de ieșire nu trebuie să fie de deplasare, și toți cei *n* biți ai operanzilor pot alimenta în paralel intrările sumatorului. De asemenea nu sunt necesare elemente de întârziere sau de memorare ale CY intermediare.
În Fig.1. 3 este dată schema de principiu a unui sumator RCA.

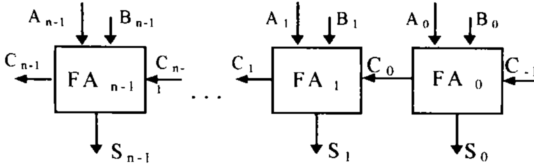


Fig.1. 3

1.1.2 Sumator Manchester

Sumatorul Manchester (*Manchester Adder* - MA) este de fapt un sumator RCA la care calea de propagare a lui CY e formată din switch-uri ce înlocuiesc porțile din această cale. Această cale de propagare a lui CY este mai rapidă decât cea a sumatorului RCA. În Fig.1. 4 este prezentat un singur rang din calea de propagare a lui CY a unui asemenea sumator. Ea constă din 3 switch-uri S_G, S_P și S_O, ce operează după cum urmează:

- Când ambii biți operanzi (A_i, B_i) asociați reangului respectiv sunt 1, se-nchide switch-ul S_G și *CY-out* al rangului este 1 indiferent de valoarea lui *CY-in*.
- Când ambii biți operanzi asociați reangului respectiv sunt 0, se-nchide switch-ul S_O și *CY-out* al rangului este 0 indiferent de valoarea lui *CY-in*.
- Când unul din biții operanzi este 0, celălalt este 1 și *CY-in* este 1 se-nchide switch-ul S_P și *CY-out* al rangului este 1.

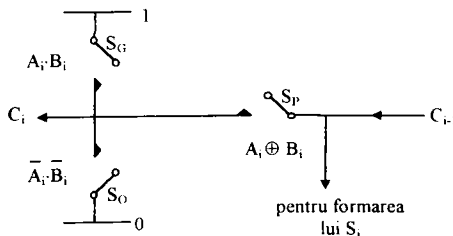


Fig.1. 4

În proiectul original switch-urile S_G și S_P erau realizate din tranzistoare simetrice, dar întrucât la stadiul tehnologic de la momentul respectiv astfel de tranzistoare erau dificil de implementat, sumatorul a fost abandonat până la dezvoltarea VLSI-urilor.

În plus, sumatorul MA prezintă avantajul structurii sale regulate ce favorizează implementarea lui la circuitele VLSI. Astfel, au fost implementate o serie de sumatoare ce combină acestă idee cu tehnica *CY-skip*.

BUPT

1.1.3 Sumator cu semnalizarea sfârşitului însumării

Un sumator proiectat astfel încât timpul de însumare să varieze în funcţie de operanzi, este mai rapid decât sumatorul RCA. Un asemenea sumator necesită logică suplimentară care să detecteze sfârşitul propagării lui CY, astfel încât odată sesizat momentul să se poată opri procesul de însumare.
Un asemenea sumator este cunoscut în literatură sub denumirea de sumator cu semnalizarea sfârşitului însumării (*Carry Completion Adder* - CCA).

CCA se obţine pornind de la celula FA la care se include o logică de detecţie a propagării complete.
Procesul de adunare al oricărui rang este complet când cifrele binare ale operanzilor au fost adunate şi s-a stabilit: fie că nu va fi CY de la acest rang, fie că CY a fost deja produs.

- Prima situaţie, CY=0, există dacă cifrele binare ale ambilor operanzi sunt 0, sau dacă una este 0 şi cealaltă este 1 şi CY=0.
- Cel de-al doilea caz, cazul unui CY=1, apare dacă: fie cifrele binare ale ambilor operanzi sunt 1 (şi apare un CY generat de acest rang), fie când un operand are bitul 0, celălalt are bitul 1 şi CY=1 se va propaga prin acest rang.

Esenţa creşterii în viteză a sumatorului CCA în raport cu sumatorul RCA., constă în observaţia că se poate lua decizia valorii logice a lui *CY-out* indiferent de starea lui *CY-in*, în situaţia când ambii operanzi au cifre binare 0, sau când ambii operanzi au cifrele binare 1.

În concluzie, modificarea ce se impune unui RCA pentru a realiza un CCA constă în adăugarea a două lanţuri CY:

- unul pentru detecţia valorii 0 pentru CY, notat cu 0 - CY
- altul pentru detecţia valorii 1 pentru CY, notat cu 1-CY

Dacă notăm

- C_i^0 - 0 -CY de la rangul i şi
- C_i^1 - 1 - CY de la rangul i,

se obţin următoarele ecuaţii pentru cele două semnale:

$$
\begin{aligned}
C_i^1 &= A_i \cdot B_i + \left(\overline{A_i} \cdot B_i + A_i \cdot \overline{B_i}\right) \cdot C_{i-1}^1 \\
&= A_i \cdot B_i + \left(A_i \oplus B_i\right) \cdot C_{i-1}^1 \qquad (1.2) \\
C_i^0 &= \overline{A_i} \cdot \overline{B_i} + \left(\overline{A_i} \cdot B_i + A_i \cdot \overline{B_i}\right) \cdot \overline{C_{i-1}^0}
\end{aligned}
$$

Aceste ecuaţii stau la baza sintezei celulei de rang i, numită *Carry-complete full adder* - CFA, a sumatorului CCA (Fig.1. 5).

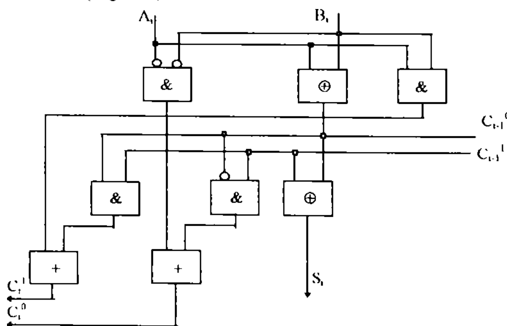


Fig.1. 5

BUPT

În Fig.1. 6 este prezentată proiectarea unui sumator complet de n biţi bazat pe utilizarea celulelor CFA.

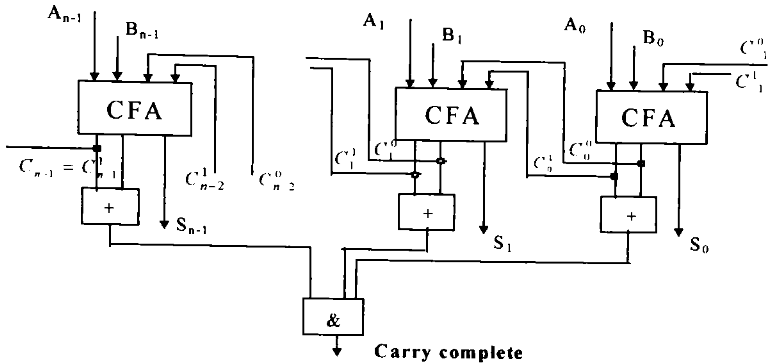


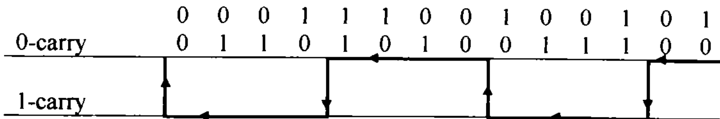
Fig.1. 6

Prin urmare, ecuaţia logică pentru semnalul de sesizare a propagării complete a lui CY (*CY-complete*) este:

$$carry-complete = \left(C_0^{\prime 0} + C_0^{\prime 1}\right)\cdot\left(C_1^{\prime 0} + C_1^{\prime 1}\right)\cdot \ldots \cdot\left(C_{n-1}^{\prime 0} + C_{n-1}^{\prime 1}\right) \qquad (1.3)$$

Esenţa operării acestui sumator constă în faptul că la nivelul fiecărui rang i, ($i = 1,2,\ldots,n-1$) odată determinat faptul că nu există *CY-out*, sau că s-a produs un *CY-out* se vor activa ieşirile $C_i^{\prime 0}$, sau $C_i^{\prime 1}$ ale rangului respectiv.

Operaţia de adunare poate fi privită în felul următor:



Trebuiesc luate măsuri de precauţie pentru a ne asigura că nu există semnale *CY-complete* false. Aceasta se poate realiza fie prin ştergerea la începutul operării a tuturor biţilor de intrare, şi alimentarea simultană a sumatorului pentru toate cifrele binare de intrare, fie prin setarea tuturor $C_i^{\prime 0}$ la 1şi a lui $C_i^{\prime 1}$ la 0 ($i = 0,1,\ldots,n-1$) [SALO87].

1.2 Structuri de însumare cu propagarea anticipată a transportului

1.2.1 Sumator cu anticiparea transportului

Ţinând cont că la nivelul rangului i se generează un transport dacă ambii biţi operanzi ai rangului respectiv sunt 1 ($A_i \cdot B_i = 1$), şi că *CY-in* este propagat peste acel rang dacă doar unul dintre biţi este 1 şi celălalt bit este 0, expresia disjunctivă a transportului C_i obţinut de la rangul i poate fi scrisă sub forma:

$$C_i = A_i \cdot B_i + C_{i-1}\cdot(A_i \oplus B_i) = G_i + C_{i-1}\cdot P_i \qquad (1.4)$$

BUPT

În expresia (1.4), s-a notat cu G_i condiția de generare a transportului în rangul i, iar cu P_i condiția de propagare a transportului peste rangul i. Cu aceste notații (funcțiile auxiliare G_i și P_i) se obține expresia generală a transportului C_i sub forma:

$$C_i = G_i + P_i \cdot G_{i-1} + P_i \cdot P_{i-1} \cdot G_{i-2} + \ldots + P_i \cdot P_{i-1} \cdot P_{i-2} \ldots \cdot P_0 \cdot C_{-1} \quad (1.5)$$

unde $G_i = A_i \cdot B_i$ și $P_i = A_i \oplus B_i$ (1.6)

și C_{-1} este transportul inițial (transportul pentru poziția c.m.p.s.).
Funcția de propagare P_i poate fi înlocuită cu *funcția de transfer* T_i:

$$T_i = A_i + B_i \quad (1.7)$$

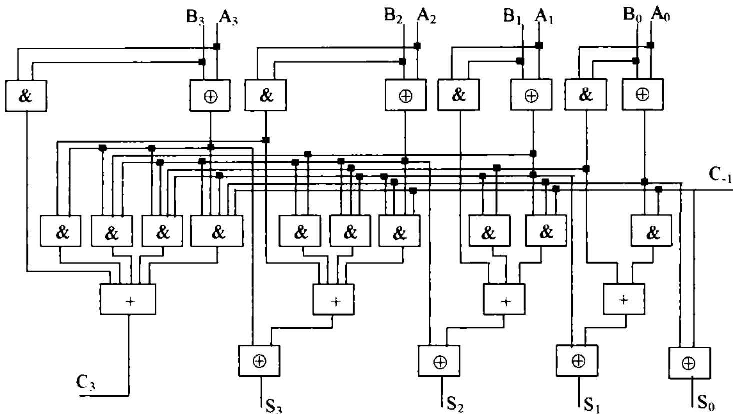


Fig.1. 7

cu observația că utilizarea pentru implementarea sumatorului a funcțiilor P_i permite utilizarea unei părți de circuite comune pentru partea de obținere a lui S_i și a lui C_i.

Un sumator la care funcțiile CY ale fiecărui rang sunt implementate pe baza ecuațiilor (1.5) și (1.6) este numit sumator cu anticiparea transportului (*Carry-Lookahead Adder* - CLA) în formă pură.
Un asemenea sumator pentru 4 ranguri este prezentat în Fig.1. 7.

La-nceputurile dezvoltării tehnologiei VLSI, se credea că sumatoarele CLA, datorită lipsei lor de regularitate și a numărului mare de conexiuni implicate, sunt nepotrivite acestei tehnologii. Lucrarea [BREN82] insistă asupra infirmării acestui lucru. Au urmat și alte rezultate utile în această direcție: [BAYO83], [NGAI84], [HAN87], [WEI90].

1.2.2 Sumatorul cu anticiparea transportului pe blocuri de cifre binare

În vederea reducerii dificultăților inerente datorate fan-in-ului și fan-out-ului mare implicat de sumatoarele CLA în formă pură, se împarte sumatorul de n biți în N $(1 < N < n)$ blocuri de m cifre binare fiecare, astfel încât transporturile la nivelul blocurilor să se anticipeze, iar transporturile dintre blocuri să se propage în serie. În Fig.1. 8 este dată organizarea de principiu a unui asemenea sumator, numit sumator cu anticiparea transportului pe blocuri de cifre binare (*Ripple-Block Carry-Lookahead Adder* - RCLA). Fiecare sumator CLA din figură este un CLA complet de forma celui din Fig.1. 7.

BUPT

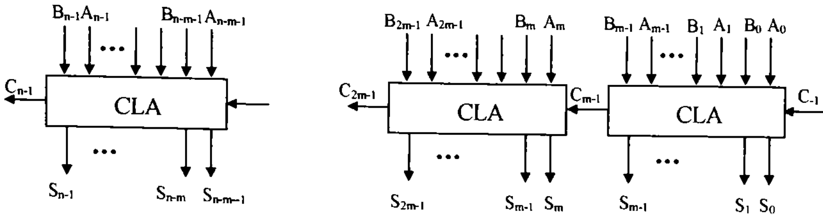


Fig.1. 8

Ecuațiile implementate de acest sumator sunt următoarele:

$$
\begin{aligned}
&C_0 = G_0 + P_0 \cdot C_{-1} \\
&C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_{-1} \\
&\dots \\
&C_{m-1} = G_{m-1} + P_{m-1} \cdot G_{m-2} + \dots + P_{m-1} \cdot P_{m-2} \cdot \dots \cdot P_0 \cdot C_{-1} \\
&C_m = G_m + P_m \cdot C_{m-1} \\
&C_{m+1} = G_{m+1} + P_{m+1} \cdot G_m + P_{m+1} \cdot P_m \cdot C_{m-1} \\
&\dots \\
&C_{2m-1} = G_{2m-1} + P_{2m-1} \cdot G_{2m-2} + \dots + P_{2m-1} \cdot P_{2m-2} \cdot \dots \cdot P_m \cdot C_{m-1} \\
&\dots \\
&C_{(N-1)m} = G_{(N-1)m} + P_{(N-1)m} \cdot C_{(N-1)m-1} \\
&C_{(N-1)m+1} = G_{(N-1)m+1} + P_{(N-1)m+1} \cdot G_{(N-1)m} + P_{(N-1)m+1} \cdot P_{(N-1)m} \cdot C_{(N-1)m-1} \\
&\dots \\
&C_{Nm-1} = G_{Nm-1} + P_{Nm-1} \cdot G_{Nm-2} + \dots + P_{Nm-1} \cdot P_{Nm-2} \cdot \dots \cdot P_m \cdot C_{(N-1)m-1}
\end{aligned}
\tag{1.8}
$$

1.2.3 Sumatorul cu anticiparea transportului între blocuri de cifre binare

O altă manieră practică de proiectare a sumatorului CLA este de a propaga serie transporturile la nivelul blocurilor și de a le anticipa între blocuri. Acest tip de sumator este eficient doar dacă n este suficient de mare și m este suficient de mic pentru ca pierderea de viteză datorită propagării serie a transportului la nivelul blocurilor să fie compensată de câștigul în viteză dat de anticiparea transporturilor între blocuri.

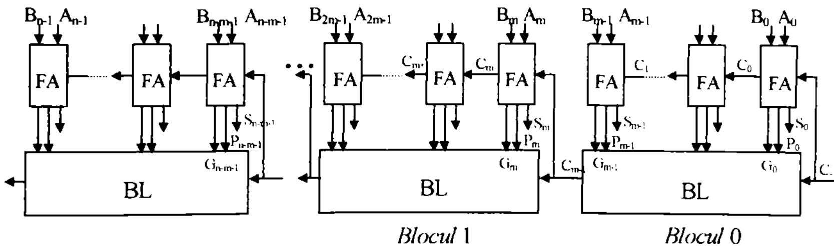


Fig.1. 9

În Fig.1. 9 este dată organizarea de principiu a unei părți a unui asemenea sumator, numit sumator cu anticiparea transportului între blocuri de cifre binare (*Block Carry Lookahead Adder* - BCLA). Unitatea *CY-lookahead* la nivel de bloc – BL – din

BUPT

Fig.1. 9 este construită prin implementarea ecuațiilor de anticipare de la nivelul unui singur bloc, iar în Fig.1. 10 este prezentată schema detaliată corespunzătoare.

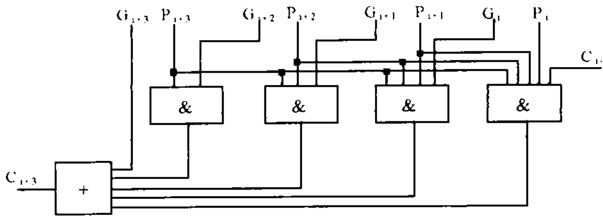


Fig.1. 10

1.2.4 Sumatoare organizate pe superblocuri

1.2.4.1 Sumatorul cu anticiparea transportului între blocuri organizat pe două niveluri

Prin utilizarea a două niveluri de anticipare a transporturilor la sumatorul BCLA se obține un alt tip de sumator bazat pe tehnica *lookahead*. Funcția celui de-al doilea nivel de anticipare a transporturilor a acestui nou sumator este de reducere a timpului de propagare serie a transporturilor dintre unitățile *block-lookahead* -BL-.
Acest principiu de creştere a numărului de niveluri de anticipare poate fi extins, obținându-se astfel sumatoare cu mai mult de două astfel de niveluri, dar întârzierile ce apar datorită anticipărilor suplimentare introduse fac ca aceste sumatoare să fie mai puțin performante comparativ cu primele.

Vom considera un sumator BCLA cu M blocuri de m cifre binare ce sunt grupate într-un superbloc, şi la care al doilea nivel de anticipare acționează la nivelul superblocului. Un asemenea sumator este referit prin termenul de sumator cu anticiparea transportului între blocuri organizat pe două niveluri (*SuperBlock Carry-Lookahead Adder* - SBCLA).
Ieşirile CY ale acestui sumator sunt obținute pe baza semnalelor auxiliare de propagare P_i^j şi de generare G_i^j ale primului nivel, semnale ce sunt similare semnalelor P şi G. Acestea sunt urmatoarele:

$$P_0^{m-1} = P_{m-1} \cdot P_{m-2} \cdot \ldots \cdot P_0$$
$$P_m^{2m-1} = P_{2m-1} \cdot P_{2m-2} \cdot \ldots \cdot P_m$$
$$\ldots$$
$$P_m^{Nm-1} = P_{Nm-1} \cdot P_{Nm-2} \cdot \ldots \cdot P_{(N-1)m} \qquad (1.9)$$
$$G_0^{m-1} = G_{m-1} + P_{m-1} \cdot G_{m-2} + \ldots + P_{m-1} \cdot P_{m-2} \cdot \ldots P_1 \cdot G_0$$
$$G_0^{2m-1} = G_{2m-1} + P_{2m-1} \cdot G_{2m-2} + \ldots + P_{2m-1} \cdot P_{2m-2} \cdot \ldots P_{m+1} \cdot G_m$$
$$\ldots$$
$$G_{(N-1)m}^{Nm-1} = G_{Nm-1} + P_{Nm-1} \cdot G_{Nm-2} + \ldots + P_{Nm-1} \cdot P_{Nm-2} \cdot \ldots \cdot P_{(N-1)m+1} \cdot G_{(N-1)m}$$

unde $N = n/m$

Prin urmare, semnalele P_i^j sunt expresia condițiilor de propagare a unui CY de la rangul i la rangul j, iar semnalele G_i^j sunt expresia generării unui CY la nivelul

BUPT

oricăruia dintre rangurile cuprinse între rangul i şi rangul j, CY ce se propagă prin celelalte ranguri ale blocului.

Semnalele CY ale blocurilor ($C_{m-1}, C_{2m-1}, \ldots, C_{(M-1)m-1}, C_{Mm}$, ş.a.) se obţin prin anticipare la nivelul blocului, ca şi la BCLA, şi semnalele CY la nivelul superblocurilor ($C_{Mm-1}, C_{2Mm-1}, C_{3Mm-1}$, ş.a.) se obţin la nivelul unităţilor *superbloc lookahead* – SBL – , ele având următoarele ecuaţii:

$$C_{Mm-1} = G_{(M-1)m}^{Mm-1} + P_{(M-1)m}^{Mm-1} \cdot G_{(M-2)m}^{(M-1)m-1} + \ldots + P_{(M-1)m}^{Mm-1} \cdot P_{(M-2)m}^{(M-1)m-1} \cdot \ldots \cdot P_0^{m} \cdot C_{-1}$$

$$C_{2Mm-1} = G_{(2M-1)m}^{2Mm-1} + P_{(2M-1)m}^{2Mm-1} \cdot G_{(2M-2)m}^{(2M-1)m-1} + \ldots + P_{(2M-1)m}^{2Mm-1} \cdot P_{(2M-2)m}^{(2M-1)m-1} + \\ + \cdot \ldots \cdot P_{Mm}^{(M+1)m} \cdot C_{Mm-1} \qquad (1.10)$$

$$\ldots$$

$$C_{Nm-1} = G_{(N-1)m}^{Nm-1} + P_{(N-1)m}^{Nm-1} \cdot G_{(N-2)m}^{(N-1)m-1} + \ldots + P_{(N-1)m}^{Nm-1} \cdot P_{(N-2)m}^{(N-1)m-1} \cdot \ldots \cdot P_{(N-M)m}^{(N-M+1)m-1} \cdot C_{-1}$$

În Fig.1. 11 este dată organizarea de principiu a unui asemenea sumator pe 32 biţi, cu $m=4$ şi $M=4$, iar în Fig.1. 12 sunt date detaliile primului nivel de *lookahead*.

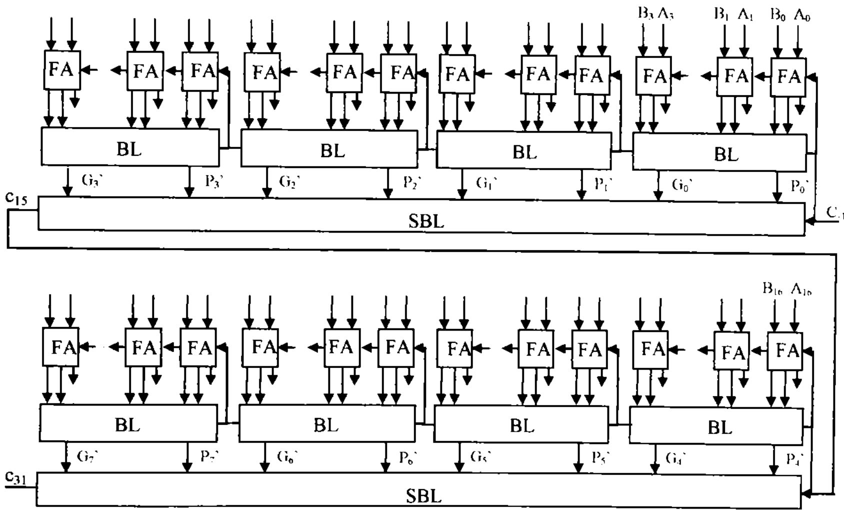


Fig.1. 11

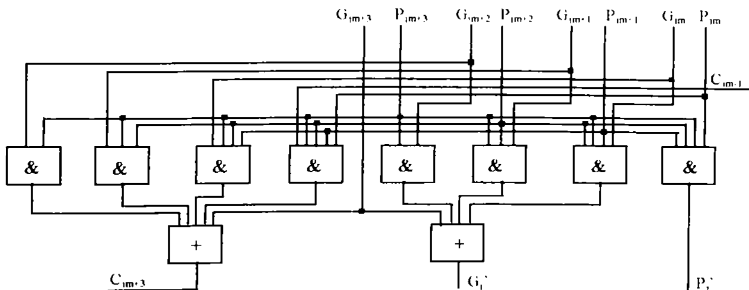


Fig.1. 12

BUPT

1.2.4.2 Sumatorul cu anticiparea transportului între blocuri organizat pe două niveluri în variantă îmbunătățită

La sumatoarele SBCLA de dimensiuni mari, timpul consumat de propagarea serie a transporturilor prin unitățile *block-lookahead* poate fi relativ mare. Prin eliminarea acestei propagări serie se obține un sumator mai rapid.

Astfel, sumatorul SBCLA îmbunătățit (*Improved SuperBlock Carry-Lookahead Adder* - ISBCLA) - se obține pornind de la un sumator SBCLA la care toate CY-urile de bloc ale nivelului superbloc sunt produse simultan, așa cum se arată în Fig.1. 13. Fig.1. 14 și Fig.1. 15 prezintă detalii ale unităților *lookahead.*

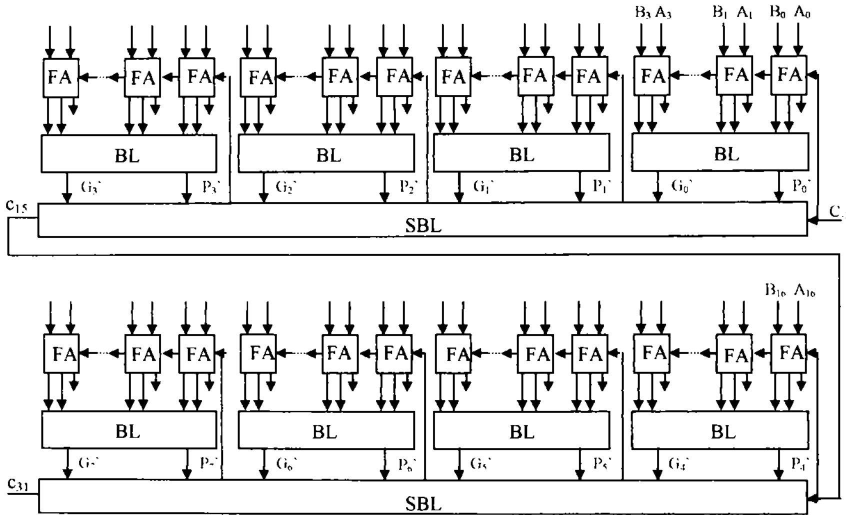


Fig.1. 13

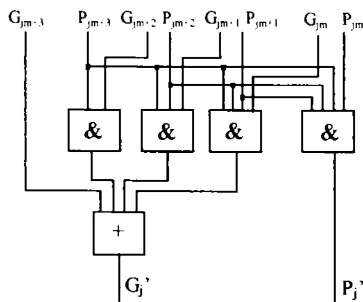


Fig.1. 14

Semnalele CY ale blocurilor se generează pe baza semnalelor P_i^j și G_i^j, după cum urmează:

$$C'_{m-1} = G_0^{\prime m-1} + P_0^{m-1} \cdot C'_{-1}$$

BUPT

$$C_{2m-1} = G_0^{2m-1} + P_0^{2m-1} \cdot C_{m-1} =$$
$$= G_m^{2m-1} + P_m^{2m-1} \cdot G_0^{m-1} + P_m^{2m-1} \cdot P_0^{m-1} \cdot C_{-1}$$

...

$$C_{Mm-1} = G_{(M-1)m}^{Mm-1} + P_{(M-1)m}^{Mm-1} \cdot C_{(M-1)m-1} =$$
$$= G_{(M-1)m}^{Mm-1} + P_{(M-1)m}^{Mm-1} \cdot G_{(M-2)m}^{(M-1)m-1} + \ldots + P_{(M-1)m}^{Mm-1} \cdot G_{(M-2)m}^{(M-1)m-1} \cdot \ldots \cdot P_0^{m-1} \cdot C_{-1}$$

$$C_{(M+1)m-1} = G_{(M)m}^{(M+1)m-1} + P_{(M)m}^{(M+1)m-1} \cdot C_{(M)m-1}$$
$$C_{(M+2)m-1} = G_{(M+1)m}^{(M+2)m-1} + P_{(M+1)m}^{(M+2)m-1} \cdot C_{(M)m-1} =$$
$$= G_{(M+1)m}^{(M+2)m-1} + P_{(M+1)m}^{(M+2)m-1} \cdot G_{(M+1)m}^{(M+2)m-1} + P_{(M+1)m}^{(M+2)m-1} \cdot P_{(M+1)m}^{(M+2)m-1} \cdot C_{Mm-1}$$

...

$$C_{2Mm-1} = G_{(2M-1)m}^{2Mm-1} + P_{(2M-1)m}^{2Mm-1} \cdot C_{(2M-1)m-1} =$$
$$= G_{(2M-1)m}^{2Mm-1} + P_{(2M-1)m}^{2Mm-1} \cdot G_{(2M-21)m}^{(2M-1)m-1} + \ldots + P_{(2M-1)m}^{2Mm-1} \cdot P_{(2M-2)m}^{(2M-1)m-1} \cdot \ldots \cdot P_{Mm}^{(M+1)m-1} C_{Mm-1} \qquad (1.11)$$

...

...

$$C_{(N-M+1)m-1} = G_{(N-M)m}^{(N-M+1)m-1} + P_{(N-M)m}^{(N-M+1)m-1} \cdot C_{(N-M)m-1}$$
$$C_{(N-M+2)m-1} = G_{(N-M+1)m}^{(N-M+2)m-1} + P_{(N-M+1)m}^{(N-M+2)m-1} \cdot C_{(N-M)m-1}$$
$$= G_{(N-M+1)m}^{(N-M+2)m-1} + P_{(N-M+1)m}^{(N-M+2)m-1} \cdot G_{(N-M)m}^{(N-M+1)m-1} +$$
$$+ P_{(N-M+1)m}^{(N-M+2)m-1} \cdot P_{(N-M)m}^{(N-M+1)m-1} C_{(N-M)m-1}$$

...

$$C_{Nm-1} = G_{(N-1)m}^{Nm-1} + P_{(N-1)m}^{Nm-1} \cdot C_{(N-1)m-1}$$
$$= G_{(N-1)m}^{Nm-1} + P_{(N-1)m}^{Nm-1} \cdot G_{(N-2)m}^{(N-1)m-1} + \ldots +$$
$$+ P_{(N-1)m}^{Nm-1} \cdot P_{(N-2)m}^{(N-1)m-1} \cdot \ldots \cdot P_{(N-M)}^{(N-M+1)m-1} C_{(N-1)m-1}$$

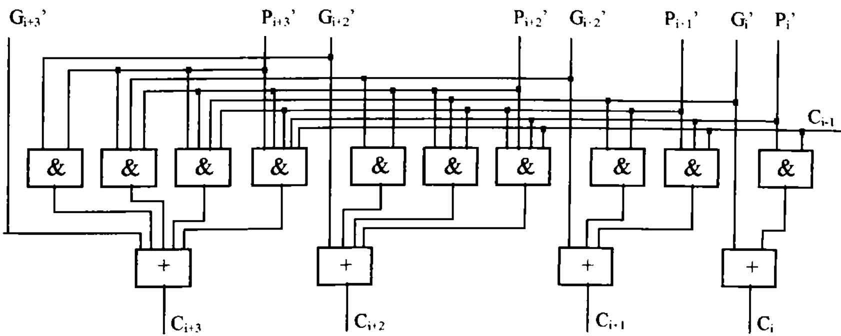


Fig.1. 15

BUPT

1.2.4.3 Sumatorul cu anticiparea transportului pe blocuri organizat pe două niveluri

Acest tip de sumator se obține prin adăugarea unui al doilea nivel de *lookahead* la sumatorul RCLA, obținându-se astfel o structură numită Sumator cu anticiparea transportului pe blocuri organizat pe două niveluri (*SuperBlock-Ripple Carry-Lookahead Adder* - SRCLA) cu performanțe de viteză mai mari datorită faptului că este redus timpul de propagare a transportului între blocuri. Organizarea sumatorului SRCLA este prezentată în Fig.1. 16, unde blocurile CLA sunt de 4 cifre binare, iar blocurile BPG sunt de fapt blocuri ce realizează generarea semnalelor P' şi G'. Detaliile logice ale unităților lookahead de bloc şi superbloc sunt aceleaşi ca ale sumatorului ISBCLA.

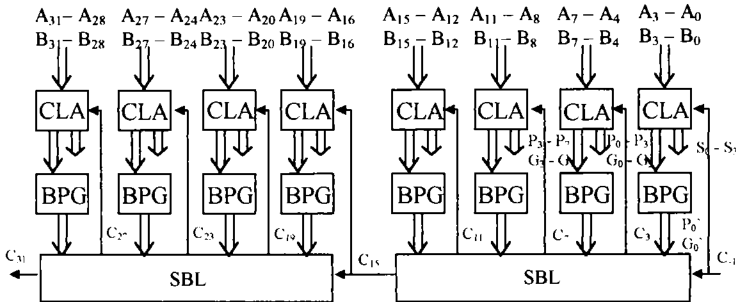


Fig.1. 16

1.2.5 Sumator cu anticipare piramidală a transportului

Conform ecuațiilor de implementare a structurii de însumare CLA, sinteza unui asemenea sumator pentru operare pe n biți, reclamă un fan-in de valoare $n+1$ atât pentru poarta OR cât şi pentru poarta AND cea mai puțin semnificativă. De asemenea, semnalul P_{n-1} trebuie să comande n porți AND. Aceste aspecte precum şi faptul că structura circuitului este destul de neregulată implicând conexiuni lungi fac impracticabilă utilizarea sumatorului CLA pentru valori mari ale lui n.

Totuşi, pornind de la conceptul *carry-lokahead*, se poate construi un sumator cu anticipare piramidală a transportului (Pyramidal CLA – PyCLA), cu $\log_2 n$ niveluri logice (mult mai puține decât cele $2n$ impuse de un sumator RCA) ce are o structură simplă şi regulată. La baza implementării acestui sumator se află ideea construirii pe paşi a semnalelor P şi G. Ecuațiile implementării sunt următoarele:

$$C_0 = g_0 + C_{-1} \cdot p_0$$
$$C_{01} = G_{01} + C_{-1} \cdot P_{01} \qquad (1.12)$$

unde G_{01} indică generarea unui CY de către blocul primilor 2 biți, iar P_{01} reprezintă CY propagat prin acest bloc. Ecuațiile pentru P şi G sunt următoarele:

$$G_{01} = g_1 + p_1 \cdot g_0$$
$$P_{01} = p_1 \cdot p_0 \qquad (1.13)$$

Generalizând pentru $\forall j$, cu $i < j, j+1 < k$, avem următoarele ecuații recursive:

$$C_k = G_{ik} + P_{ik} \cdot C_{k-1}$$
$$G_{ik} = G_{j-1,k} + P_{j-1,k} G_{ij} \qquad (1.14)$$

BUPT

$$P_{ik} = P_{ij} \cdot P_{j+1,k}$$

Aceste ecuații exprimă faptul că se generează un CY la nivelul blocului ce conține biții de la i la k (inclusiv) când:

- el este generat de partea superioară a blocului ce este compusă de biții $(j+1,k)$, sau
- el este generat de partea inferioară a blocului, compusă de biții (i,j) fiind propagat în continuare peste partea superioară a blocului.

Ecuațiile sunt valabile și pentru $i \le j < k$ dacă se setează $G_{ij} = g_i$ și $P_{ij} = p_i$.

Fig.1. 17 prezintă structura unui asemenea sumator PyCLA ce realizează însumarea pe 8 biți. Biții numerelor ce se însumează parcurg arborele de sus în jos, la bază se combină semnalele generate cu C_{-1}, și rezultatele parcurg arborele de jos în sus generând semnalele CY și biții sumă calculați.

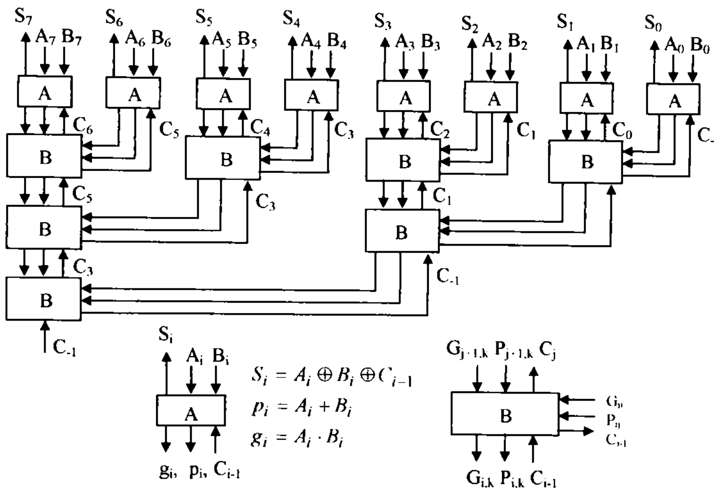


Fig.1. 17

1.3 Structuri de însumare cu omitere a transportului

1.3.1 Sumator cu un nivel de omitere a transportului

Această structură de însumare, numită sumator cu un nivel de omitere a transportului (*Carry-Skip Adder* CSkA), se obține pornind de la un RCA la care rangurile sunt grupate în blocuri (nu neapărat de aceeași dimensiune), ce se echipează cu o logică de omitere (*CY-skip*) care detectează dacă *CY-in* al blocului poate omite blocul respectiv, fiind trecut astfel, direct, ca și *CY-in* la blocul următor.

Ideea de bază a sumatorului CSkA constă în faptul că, la nivelul rangului i, dacă $A_i = 1$, sau dacă $B_i = 1$, *CY-in* va omite acest rang. Pe de altă parte, singura situație când rangul i are un *CY-in*=1 și nu generează *CY-out* este când atât $A_i = 0$, cât și $B_i = 0$.

Prin urmare, un CY va omite blocul i de lungime m dacă:

$$(A_i + B_i)\cdot(A_{i+1} + B_{i+1})\cdot\ldots\cdot(A_{i+m-1} + B_{i+m-1}) = P_i \cdot P_{i+1} \cdot\ldots\cdot P_{i+m-1} = 1 \qquad (1.15)$$

(T_i este semnalul *CY transfer* al rangului i).

BUPT

Blocul i furnizează un CY-*in* blocului $i+1$, în două situaţii:

- dacă C_{i-1} omite blocul i
- blocul i produce C_{i+m-1}

În aceste condiţii, logica CY-*skip* a fiecărui bloc constă din următoarele:

- câte o poartă OR pentru obţinerea fiecărui semnal T_i
- o poartă AND pentru combinarea semnalelor T_i
- o poartă OR pentru selectarea sursei pentru CY

Aceste condiţii sunt date de ecuaţia:

$$block\ CY-out = T_i \cdot T_{i+1} \cdot \ldots \cdot T_{i+m-1} \cdot C_{i-1} + C_{i+m-1} \quad (1.16)$$

În Fig.1. 18 este prezentată schema de principiu a CskA cu un nivel de omitere.

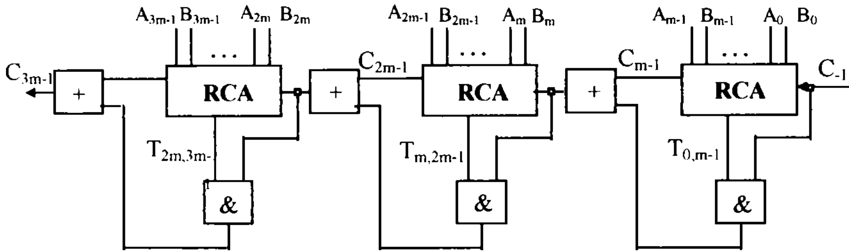


Fig.1. 18

Sumatorul CSkA este utilizabil numai dacă semnalele CY-*out* pot fi şterse uşor la-nceputul fiecărei operaţii (de exemplu prin pre-încărcare în CMOS).

1.3.2 Sumator cu mai multe niveluri de omitere a transportului

Pentru obţinerea unor îmbunătăţiri ale timpului de operare, se poate aplica pentru sumatorul CSkA obţinut aceleaşi principii de modificare pe care le-am folosit pentru dezvoltarea sumatorului CSkA pornind de la sumatorul RCA. În felul acesta se obţine un sumator CSkA pe două niveluri.

În acest scop, blocurile skip se grupează în superblocuri (*superblock*), ce se prevăd fiecare în parte cu logică CY-*skip*.

Presupunem că avem o dimensiune m pentru blocuri şi o dimensiune M pentru superblocuri. Un CY-*in* pentru superblocul i va omite superblocul i, dacă:

$$(A_i + B_i)\cdot(A_{i+1} + B_{i+1})\cdot \ldots \cdot(A_{i+mM-1} + B_{i+mM-1}) = T_i \cdot T_{i+1} \cdot \ldots \cdot T_{i+mM-1} = 1 \quad (1.17)$$

Ţinând cont de logica *skip* ce este deja disponibilă la nivelul fiecărui bloc, şi definind:

$$K_j = P_j \cdot P_{j+1} \cdot \ldots \cdot P_{j+m-1} \quad (1.18)$$

condiţia de omitere a superblocului i poate fi scrisă:

$$K_i \cdot K_{i+1} \cdot \ldots \cdot K_{i+M-1} = 1 \quad (1.19)$$

La nivelul unui superbloc va apărea un CY-*out* dacă apare una din următoarele 3 condiţii:

(i) Dacă CY-*in* omite superblocul

BUPT

(ii) Dacă *CY-out* al blocului *M-2* din superbloc omite blocul M-1 al aceluiaşi superbloc.

(iii) Este produs un CY -out la nivelul blocului M-1.

Deci, la nivelul fiecărui superbloc este necesară o logică adiţională ce realizează implementarea următoarei ecuaţii:

$$Super-block\ CY-out = C_{i+mM-1} + K_{i+M-1} \cdot C_{i+(M-1)m-1} + K_i \cdot K_{i+1} \cdot \ldots \cdot K_{i+M-1} \cdot C_{i-1} \qquad (120)$$

Aceasta duce la organizarea unui superbloc pe două niveluri ca în Fig.1. 19.

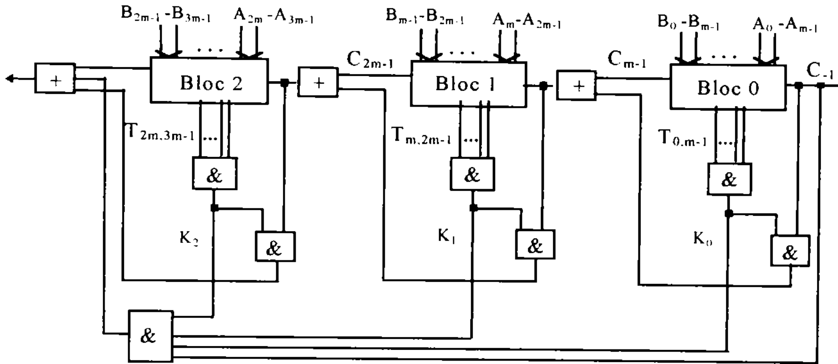


Fig.1. 19

1.4 Structuri de însumare cu propagare specială a transporturilor

1.4.1 Sumator cu condiţionarea sumei

Principiul de bază al proiectării sumatorului cu condiţionarea sumei (*Conditional-Sum Adder* - CdSumA), tratat în [ERCE90] [KORN94] se bazează pe generarea într-un singur pas a tuturor posibilităţilor de biţi sumă şi a biţilor CY, iar apoi biţii CY determinaţi sunt utilizaţi pentru selectarea biţilor corecţi ai sumei; această selecţie se face într-o serie de alţi paşi de însumare, la fiecare pas realizându-se o dublare a numărului de biţi sumă corect determinaţi.

Astfel, în primul pas se face:

- o determinare a unui set de biţi sumă şi CY sub presupunerea că *CY-in* a fiecărui rang este 0.
- o determinare a unui alt set de biţi sumă şi CY sub presupunerea că *CY-in* a fiecărui rang este 1.

În pasul al doilea se face o împerechere a rangurilor şi se iau în considerare variantele posibile de CY între cele două ranguri ale unei astfel de perechi. Se obţine astfel un set nou de biţi sumă şi CY pentru perechile de ranguri formate.

În pasul al treilea, se grupează rangurile câte 4 şi se iau în considerare variantele posibile de CY ce pot apărea între cele două perechi ale grupurilor formate. Din nou se generează un nou set de biţi sumă şi CY pentru fiecare grup de 4 biţi.

BUPT

Acest proces se continuă până se ajunge ca dimensiunea grupurilor formate să fie egală cu dimensiunea sumatorului, moment la care este disponibil rezultatul corect al însumării.
Prin urmare, întrucât la fiecare pas se face o dublare a dimensiunii grupurilor şi a numărului corect de biţi sumă determinaţi, pentru realizarea unei însumări a n cifre binare sunt necesari $(1+\log_2 n)$ paşi.

În Tabel 1. 1 este ilustrată această procedură de însumare pentru operanzi cu 8 cifre binare.

i		7		6		5		4		3		2		1		0	
A			0		1		0		1		0		1		0		1
B			1		0		0		0		0		0		1		1
		C	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S
j=0	C=0	0	1	0	1	0	0	0	1	0	0	0	1	0	1	1	0
	C=1	1	0	1	0	0	1	1	0	0	1	1	0	1	0		
j=1	C=0	0	1		1	0	0		1	0	0		1	1	0		0
	C=1	1	0		0	0	1		0	0	1		0				
j=2	C=0	0	1		1		0		1	0	1		0		0		0
	C=1	0	1		1		1		0								
j=3	C=0	0	1		1		0		1		1		0		0		0

Tabel 1. 1

În Fig.1. 20 este prezentată organizarea la nivel înalt a unui asemenea sumator pe 8 ranguri, unde s-a notat cu:

- ${}_jS_i^k$ şi ${}_jC_i^k$ valorile pasului j pentru bitul sumă şi CY ale rangului i, sub presupunerea că *CY-in* pentru rangul respectiv este k, unde k poate fi 0 sau 1.
- UN_i unităţile de însumare ale nivelului i

Detalierea blocurilor din figură se face după cum urmează:

Nivelul 0: Logica acestui nivel este formată din FA, fiecare din modulele acestui nivel având detaliile logice reprezentate în Fig.1. 21. Ecuaţiile logice implementate de nivel fiind următoarele:

$$
\begin{aligned}
&{}_0C_i^0 = A_i \cdot B_i \qquad i = 1,2,\ldots,n-1\\
&{}_0C_i^1 = A_i \cdot \overline{B_i} + \overline{A_i} \cdot B_i + A_i \cdot B_i = A_i + B_i\\
&{}_0S_i^0 = A_i \oplus B_i\\
&{}_0S_i^1 = \overline{A_i} \cdot \overline{B_i} + A_i \cdot B_i = \overline{{}_0S_i^0}
\end{aligned}
$$

Nivelul 1: În Fig.1. 22 este prezentată partea de logică a unui modul al nivelului 1 ce producce perechile de semnale (C_1, S_1) şi $\left(C_{i+1}^0, S_{i+1}^0\right)$, $i = 0,2,4,etc.$; logica producerii perechilor $\left(C_{i+1}^1, S_{i+1}^1\right)$, $i = 0,2,4,etc.$ este similară. Ecuaţiile nivelului 1 sunt:

$$
\begin{aligned}
&C_1 = {}_0C_1^0 \cdot \overline{C_0} + {}_0C_1^1 \cdot C_0\\
&S_1 = {}_0S_1^0 \cdot \overline{C_0} + {}_0C_1^1 \cdot C_0\\
&{}_1S_{i+1}^0 = {}_0S_{i+1}^0 \cdot \overline{{}_0C_i^0} + {}_0S_{i+1}^1 \cdot {}_0C_i^0 \qquad i\ 2, 4, 6, \ldots, n\text{-}2\\
&{}_1S_{i+1}^1 = {}_0S_{i+1}^0 \cdot \overline{{}_0C_i^1} + {}_0S_{i+1}^1 \cdot {}_0C_i^1\\
&{}_1C_{i+1}^0 = {}_0C_{i+1}^0 \cdot \overline{{}_0C_i^0} + {}_0C_{i+1}^1 \cdot {}_0C_i^0\\
&{}_1C_{i+1}^1 = {}_0C_{i+1}^0 \cdot {}_0C_i^1 + {}_0C_{i+1}^1 \cdot {}_0C_i^1
\end{aligned}
$$

BUPT

Nivelul 2: Ecuațiile și logica acestui nivel sunt similare cu cele ale nivelului 1, cu diferența că numărul ecuațiilor și cantitatea de logică implicată este mult mai mare. În Fig.1. 23 este prezentată jumătate din logica nivelului 2; cealaltă jumătate fiind similară. Ecuațiile nivelului 2 sunt:

$$_2S^0_{i+1} = {_0S^0_{i+1}} \cdot \overline{_1C^0_i} + {_0S^1_{i+1}} \cdot {_1C^0_i} \qquad i = 5, 9, 13, \ldots, n-4$$
$$_2S^1_{i+1} = {_1S^0_{i+1}} \cdot \overline{_1C^1_i} + {_1S^1_{i+1}} \cdot {_1C^1_i}$$
$$_2S^0_{i+2} = {_1S^0_{i+2}} \cdot \overline{_1C^0_i} + {_1S^1_{i+2}} \cdot {_1C^0_i}$$
$$_2S^1_{i+2} = {_1S^0_{i+2}} \cdot \overline{_1C^1_i} + {_1S^1_{i+2}} \cdot {_1C^1_i}$$
$$_2C^0_{i+2} = {_0C^0_{i+2}} \cdot \overline{_1C^0_i} + {_1C^1_{i+2}} \cdot {_1C^0_i}$$
$$_2C^1_{i+2} = {_1C^0_{i+2}} \cdot {_1C^1_i} + {_1C^1_{i+2}} \cdot {_1C^1_i}$$

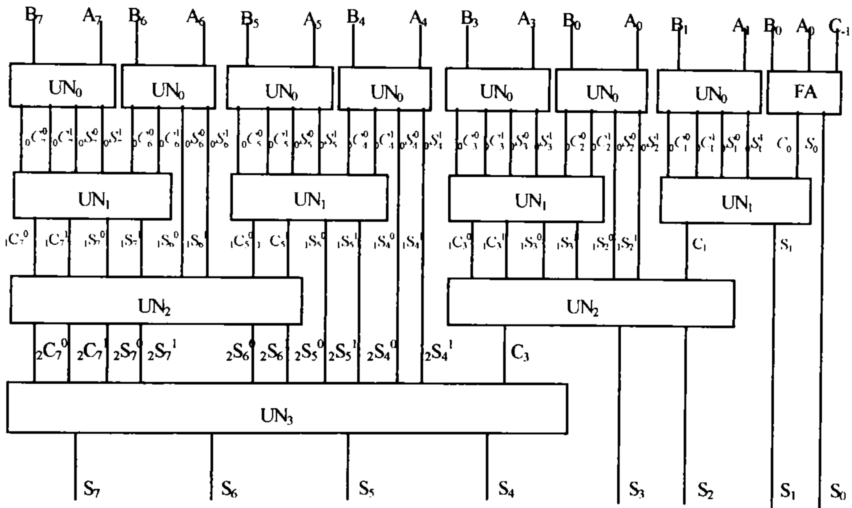


Fig.1. 20

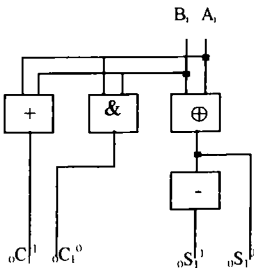


Fig.1. 21

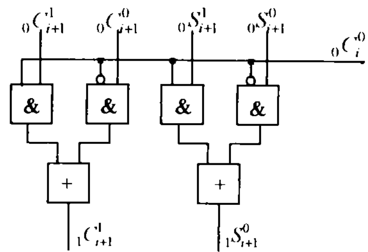


Fig.1. 22

Ca și în cazul precedent, unitatea cea mai din dreapta este un caz special al logicii generale: ${_1C^0_i} = {_1C^1_i} = C_i$.

BUPT

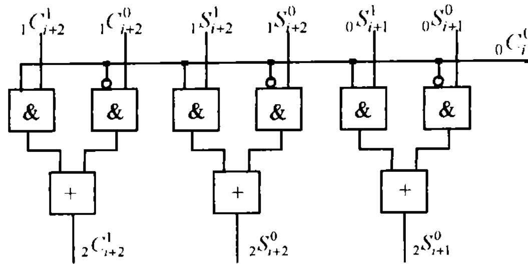


Fig.1. 23

Nivel k: În general, ecuaţiile implementate la nivelul k, pentru $i = 2^k + k - 1$, $2 \cdot 2^k + k - 1, 3 \cdot 2^k + k - 1, \ldots, n - 2^k$, sunt:

$${}_kS^0_{i+1} = {}_0S^0_{i+1} \cdot \overline{{}_{k-1}C^0_i} + {}_0S^1_{i+1} \cdot {}_{k-1}C^0_i$$

$${}_kS^1_{i+1} = {}_0S^0_{i+1} \cdot \overline{{}_{k-1}C^1_i} + {}_0S^1_{i+1} \cdot {}_{k-1}C^1_i$$

$${}_kS^0_{i+2} = {}_1S^0_{i+1} \cdot \overline{{}_{k-1}C^0_i} + {}_1S^1_{i+2} \cdot {}_{k-1}C^0_i$$

$${}_kS^1_{i+2} = {}_1S^0_{i+2} \cdot \overline{{}_{k-1}C^1_i} + {}_1S^1_{i+2} \cdot {}_{k-1}C^1_i$$

...

$${}_kS^0_{i+2^{k-1}} = {}_{k-1}S^0_{i+2^{k-1}} \cdot \overline{{}_{k-1}C^0_i} + {}_{k-1}S^1_{i+2^{k-1}} \cdot {}_{k-1}C^0_i$$

$${}_kS^1_{i+2^{k-1}} = {}_{k-1}S^0_{i+2^{k-1}} \cdot \overline{{}_{k-1}C^1_i} + {}_{k-1}S^1_{i+2^{k-1}} \cdot {}_{k-1}C^1_i$$

$${}_kC^0_{i+2^{k-1}} = {}_{k-1}C^0_{i+2^{k-1}} \cdot \overline{{}_{k-1}C^0_i} + {}_{k-1}C^1_{i+2^{k-1}} \cdot {}_{k-1}C^0_i$$

$${}_kC^1_{i+2^{k-1}} = {}_{k-1}C^0_{i+2^{k-1}} \cdot \overline{{}_{k-1}C^1_i} + {}_{k-1}C^1_{i+2^{k-1}} \cdot {}_{k-1}C^1_i$$

1.4.2 Sumator cu selecţia sumei

Sumatorul cu selecţia sumei (*Carry-Select Adder* - CSlA), este bazat pe acelaşi principiu ca şi sumatorul CdSumA, adică pe calculul mai multor seturi de biţi sumă şi alegerea dintre acestea a unuia în funcţie de valorile CY determinate.

Prin urmare, ca şi generalizare a sumatorului CdSumA, sumatorul CSlA constă într-un arbore de selecţie în care dimensiunea grupului creşte cu un factor *k (k*$\geq$*2)* la fiecare pas, şi care prin urmare constă din $(1 + \log_m n)$ niveluri [LEIG92].

În Fig.1. 24 este dată schema de principiu a sumatorului CSlA.
Sumatorul este împărţit în mai multe blocuri de m biţi, ieşirea fiecăruia dintre aceştia fiind un set de biţi sumă, produşi sub presupunerea că *CY-in* a blocului este 1, şi un alt set de biţi produşi sub presupunerea că *CY-in* este 0. După determinarea în cadrul blocului de determinare a lui CY şi selecţie a sumei – DCY şi SS – a stării semnalului *CY-in* a blocului se alege unul din aceste două seturi ca fiind corect.
La nivelul fiecărui modul de însumare logica detaliată a modulului este determinată de tehnica de însumare folosită: CRA, CLA, [PATT94] etc. De asemenea, pentru determinarea lui *CY-in* al fiecărui bloc, se utilizează oricare din tehnicile deja prezentate.

BUPT

În cele ce urmează vom prezenta un sumator CSlA ce utilizează pe cât posibil CLA, întrucât acesta este cel mai performant pentru asemenea sumatoare. Pe de altă parte sunt posibile şi alte variante de utilizare a lui CLA, de genul: CLA cu propagare serială între blocuri, respectiv cu propagare serială la nivelul blocurilor [VITI95].
Sumatorul la care se face referire în continuare are structura organizatorică din Fig.1. 25 şi el se numeşte *Lookahead-Ripple Carry-Select Adder* – LR-CSlA.
Fiecare din cele m blocuri poate fi văzut ca şi constând din:

- o unitate CY,
- o unitate de generare a sumei – GS ,
- o unitate de selecţie a sumei.- SS

În realitate cele 3 părţi nu sunt neaparat complet separate.

Unitatea CY produce un singur semnal CY pentru întregul bloc. Unitatea de generare a sumei produce doi biţi sumă pentru fiecare poziţie de bit a sumei: una pentru fiecare posibilitate de *CY-in* a blocului. CY şi ambele tipuri de biţi sumă sunt produşi folosind tehnica *lookahead*, cu un grad de concurenţă maxim posibil, şi odată ce *CY-in* în bloc a fost determinat, se face selecţia la nivelul unităţii de selecţie a sumei a biţilor corecţi ai sumei.

Replicarea circuitelor impusă de logica detaliată mai sus, este mai mică decât pare la prima vedere. Aceasta rezultă din examinarea ecuaţiilor logice ce stau la baza implementării sumatorului.
Pentru aceasta considerăm următoarele notaţii:

- Fie S_j^i bitul j al sumei calculat sub presupunerea că *CY-in* a blocului ce conţine rangul j este i, unde i este fie 0, fie 1.
- Fie $C_j^{\prime i}$, CY din rangul j, sub presupunerea că CY-în blocul ce conţine rangul j este i, unde i este fie 0, fie 1.

În aceste condiţii, biţii sumă sunt exprimaţi de:

$$S_j^1 = (A_j + B_j) \oplus C_{j-1}^{\prime 1} \quad j = 0,1,\ldots,n-1$$
$$S_j^0 = (A_j + B_j) \oplus C_{j-1}^{\prime 0} \tag{1.21}$$

CY-urile din ecuaţiile precedente sunt generate prin *lookahead.*

Fie T_j, P_j şi G_j funcţiile de CY-transfer, CY-propagat şi CY- generat (adică $A_j + B_j$, $A_j \oplus B_j$ şi $A_j \cdot B_j$). Atunci pentru un sumator de n biţi cu blocuri de m biţi, ecuaţiile logice ale CY-urilor cerute pentru a produce biţii blocului sumă ai blocului i, $i = 0,1,\ldots,\frac{n}{m}-1$ sunt (de observat că $C_{i-1}^{\prime 1} = 1$ şi $C_{i-1}^{\prime 0} = 0$):

$$\begin{aligned}
C_i^{\prime 1} &= G_i + P_i \cdot C_{i-1}^{\prime 1} = \\
&= G_i + P_i \\
&= A_i + B_i = T_i \\
C_i^{\prime 0} &= G_i + P_i \cdot C_{i-1}^{\prime 0} = G_i \\
C_{i+1}^{\prime 1} &= G_{i+1} + P_{i+1} \cdot G_i + P_{i+1} \cdot P_i \cdot C_{i-1}^{\prime 1} = \\
&= G_{i+1} + P_{i+1} \cdot G_i + P_{i+1} \cdot P_i = \\
&= G_{i+1} + P_{i+1} \cdot (G_i + P_i) = \\
&= G_{i+1} + P_{i+1} \cdot T_i \\
C_{i+1}^{\prime 0} &= G_{i+1} + P_{i+1} \cdot G_i + P_{i+1} \cdot P_i \cdot C_{i-1}^{\prime 0} = \\
&= G_{i+1} + P_{i+1} \cdot G_i
\end{aligned} \tag{1.22}$$

BUPT

$$\bullet\bullet\bullet$$

$$C^1_{i+m-2} = G_{i+m-2} + P_{i+m-2} \cdot G_{i+m-3} + \ldots + P_{i+m-2} \cdot P_{i+m-3} \cdot \ldots P_{i+1} \cdot P_i \cdot C^1_{i-1}$$
$$= G_{i+m-2} + P_{i+m-2} \cdot G_{i+m-3} + \ldots + P_{i+m-2} \cdot P_{i+m-3} \cdot \ldots P_{i+1} \cdot T_i$$
$$C^0_{i+m-2} = G_{i+m-2} + P_{i+m-2} \cdot G_{i+m-3} + \ldots + P_{i+m-2} \cdot P_{i+m-3} \cdot \ldots P_{i+1} \cdot P_i \cdot C^0_{i-1}$$
$$= G_{i+m-2} + P_{i+m-2} \cdot G_{i+m-3} + \ldots + P_{i+m-2} \cdot P_{i+m-3} \cdot \ldots P_{i+1} \cdot G_i$$

CY-in ale blocurilor sunt produse utilizând funcțiile *bloc-lookahead*, așa cum sunt definite de următoarele ecuații logice ($N = \frac{n}{m}$):

$$P_0^{m-1} = P_{m-1} \cdot P_{m-2} \cdot \ldots \cdot P_1 \cdot T_0$$
$$P_m^{2m-1} = P_{2m-1} \cdot P_{2m-2} \cdot \ldots \cdot P_{m+1} \cdot T_m$$
$$\bullet\bullet\bullet$$
$$P_{(N-1)m}^{Nm-1} = P_{Nm-1} \cdot P_{Nm-2} \cdot \ldots \cdot P_{(N-1)m+1} \cdot T_{(N-1)m},\ N = \frac{n}{m}$$

$$G_0^{m-1} = G_{m-1} + P_{m-1} \cdot G_{m-2} \cdot \ldots \cdot P_{m-1} \cdot P_{m-2} \cdot \ldots \cdot P_1 \cdot G_0$$
$$G_m^{2m-1} = G_{2m-1} + P_{2m-1} \cdot G_{2m-2} \cdot \ldots \cdot P_{2m-1} \cdot P_{2m-2} \cdot \ldots \cdot P_{m+1} \cdot G_m$$

$$\bullet\bullet\bullet \qquad (1.23)$$

$$G_{(N-1)m}^{Nm-1} = G_{Nm-1} + P_{Nm-1} \cdot G_{Nm-2} + \ldots + P_{Nm-1} \cdot P_{Nm-2} \cdot P_{(N-1)m+1} \cdot G_{(N-1)m}$$

$$C_{m-1} = G_0^{m-1} + P_0^{m-1} \cdot C_{-1}$$
$$C_{2m-1} = G_{2m-1}^{2m-1} + P_m^{2m-1} \cdot G_0^{m-1} + P_m^{2m-1} \cdot P_m^{m-1} \cdot C_{-1}$$

$$\bullet\bullet\bullet$$

$$C_{Nm-1} = G_{(N-1)m}^{Nm-1} + P_{(N-1)m}^{Nm-1} \cdot G_{(N-2)m}^{(N-1)m-1} + \ldots + P_{(N-1)m}^{Nm-1} \cdot P_{(N-2)m}^{(N-1)m-1} \cdot \ldots \cdot P_0^{m-1} \cdot C_{-1}$$

Logica de selecție a fiecărui bloc de m biți, i, implementează următoarele ecuații:

$$S_i = C_i \cdot S_i^1 + \overline{C_i} \cdot S_i^0$$
$$S_{i+1} = C_i \cdot S_{i+1}^1 + \overline{C_i} \cdot S_{i+1}^0$$
$$\bullet\bullet\bullet \qquad (1.24)$$
$$S_{i+m-1} = C_i \cdot S_{i+m+1}^1 + \overline{C_i} \cdot S_{i+m-1}^0$$

De observat că pentru orice bloc i, condițiile $C_{i-1}^1 = 1$ și $C_{i-1}^0 = 0$ permit simplificarea ecuațiilor relevante la:

$$S_i^1 = \overline{A_i \oplus B_i} = \overline{P_i}$$
$$S_i^0 = A_i \oplus B_i = P_i \qquad (1.25)$$

BUPT

BUPT

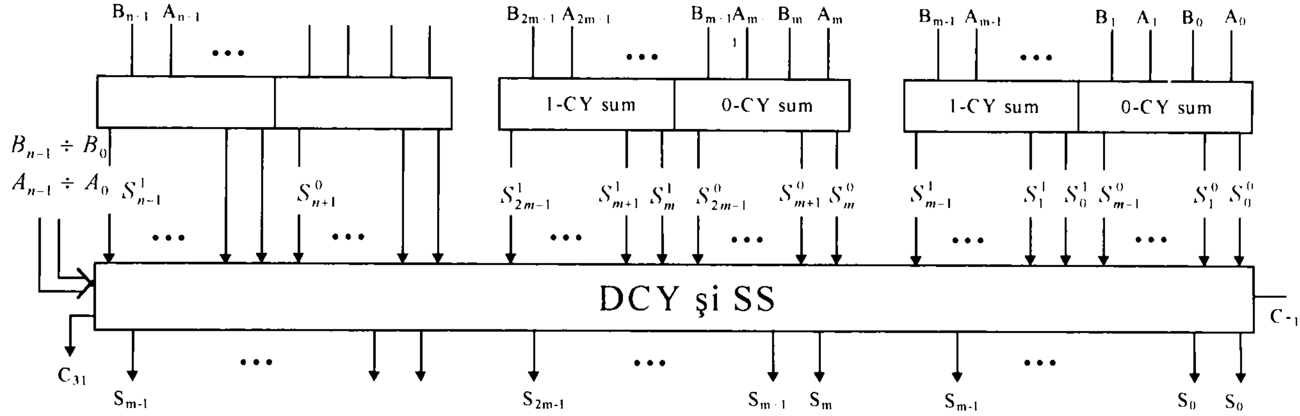


Fig.1. 24

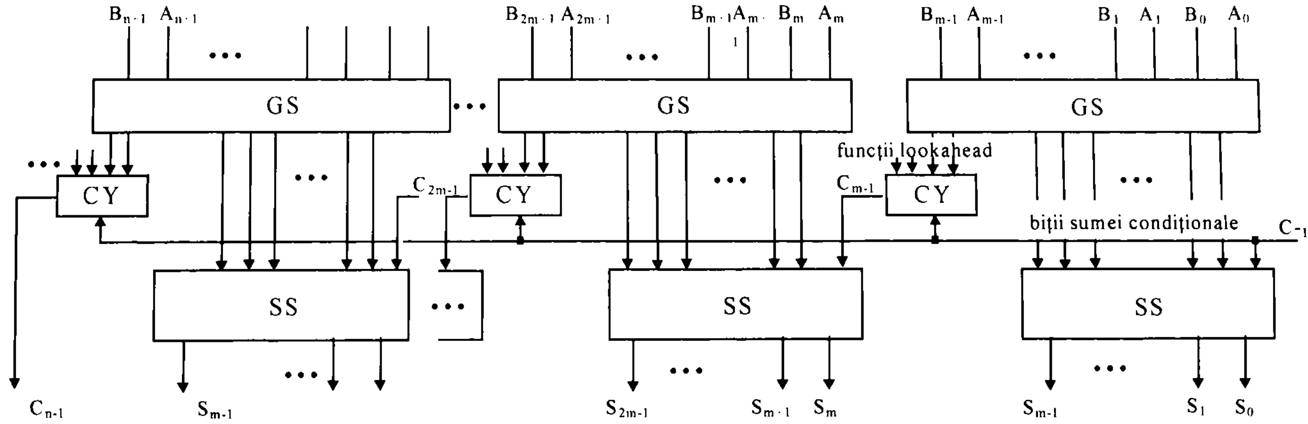


Fig.1. 25

1.4.3 Sumator piramidal

Sumatorul piramidal *(Pyramidal Adder* - PyA) are şi el o structură ce seamănă cu cea a sumatorului CdSumA. El constă din două părţi:

- o parte formată din semi-sumatoare ce produce biţii sumă parţială şi CY parţial pentru toate rangurile sumatorului;
- o a doua parte ce are o structură piramidală având baza ancorată în prima parte, şi care realizează asimilarea transporturilor parţiale cu sumele parţiale prin blocurile ACY din Fig.1. 26; procesul de asimilare constă într-un număr de paşi prin care CY-urile sunt propagate peste anumite distanţe limitate, fiind pe urmă asimilate. Fiecare astfel de pas duce la formarea unor noi sume parţiale şi a unor noi transporturi parţiale.
  Pentru fiecare pas, distanţa maximă pe care se face propagarea transporturilor este dublă faţă de cea a pasului anterior.
  Prin urmare, la nivelul fiecărui pas se face o înjumătăţire a numărului transporturilor ce rămân a fi asimilate şi astfel procesul realizării unei adunări complete durează $(1+\log_2 n)$ paşi.

Ca şi sumatorul CdSumA, sumatorul PyA se pretează la *pipelining* [TING93] [TOMA93] [KUMA94].

Pornind de la ecuaţiile:

$$S_i = (A_i \oplus B_i) \oplus C_{i-1}$$
$$C_i = A_i \cdot B_i + (\overline{A_i} \cdot B_i + A_i \cdot \overline{B_i}) \cdot C_{i-1} \tag{1.26}$$

se observă că fiind dat un bit sumă parţială PS şi un bit CY parţial PC, procesul de asimilare poate fi exprimat prin: $PS \oplus PC$. Dacă considerăm mai mulţi biţi sumă parţială: PS_j, PS_{j-1}, ..., PS_i şi bitul CY parţial PC_{i-1} ce trebuie propagat şi asimilat la aceşti biţi sumă parţială, asimilarea la nivelul rangului i poate fi exprimată prin $PS_i \oplus PC_{i-1}$. La nivelul rangului $(i+1)$, există CY pentru asimilare numai dacă s-a produs un CY prin asimilare la nivelul rangului i.

Prin urmare, producerea unui CY în pasul i poate fi exprimată prin $PS_i \cdot PC_{i-1}$ iar asimilarea la nivelul rangului $(i+1)$ prin $PS_{i+1} \oplus (PS_i \cdot PC_{i-1})$. Raţionamente simailare arată că asimilarea pentru rangurile rămase este dată de relaţiile: $PS_{i+2} \oplus (PS_{i+1} \cdot PS_i \cdot PC_{i-1})$, $PS_{i+3} \oplus (PS_{i+2} \cdot PS_{i+1} \cdot PS_i \cdot PC_{i-1})$, , $PS_j \oplus (PS_{j-1} \cdot PS_{j-2} \cdot \ldots \cdot PS_i \cdot PC_{i-1})$. Aceste relaţii stau la baza sintezei sumatorului PyA, ale cărui detalii de implementare sunt date mai jos.

Fie C_i^j şi S_i^j bitul CY şi bitul sumă al rangului i la sfârşitul pasului j. În aceste condiţii, procesul de adunare se desfăşoară în următorii paşi:

<u>Pasul 0</u>: Producerea sumelor parţiale şi a CY-urilor parţile la nivelul rangurilor 1, ... , n-1: $S_i^0 = A_i \oplus B_i$ şi $C_i^0 = A_i \cdot B_i$, I=1, 2, ..., n-1. Întrucât C_{-1} al rangului 0 este cunoscut, avem: $S_0^0 = S_0 = (A_0 \oplus B_0) \oplus C_{-1}$ şi $C_0^0 = C_0 = A_0 \cdot B_0 + (A_0 \oplus B_0) \oplus C_{-1}$.

<u>Pasul 1</u>: Rangurile se grupează câte două, şi CY-urile sunt propagate şi asimilate la nivelulul fiecărui asemenea grup. Ecuaţiile logice relevante ale pasului 2 sunt:

BUPT

$$S_{j+1}^{1} = S_{j+1}^{0} \oplus C_{j}^{0} \qquad j = 0, 2, 4, \ldots, n\text{-}2$$
$$C_{j+1}^{1} = C_{j+1}^{0} + S_{j+1}^{0} \cdot C_{j}^{0}$$
$$S_{j}^{1} = S_{j}^{0} \qquad \text{pentru celelalte valori } j$$

Pasul 2: Rangurile se grupează câte patru, şi CY-urile sunt propagate şi asimilate la nivelulul fiecărui asemenea grup. Ecuaţiile logice relevante ale pasului 3 sunt:

$$S_{j+1}^{2} = S_{j+1}^{1} \oplus C_{j}^{1} \qquad j = 1, 5, 9, \ldots, n\text{-}3$$
$$S_{j+2}^{2} = S_{j+2}^{1} \oplus S_{j+1}^{1} \cdot C_{j}^{1}$$
$$C_{j+2}^{2} = C_{j+2}^{1} + S_{j+2}^{1} \cdot S_{j+1}^{1} \cdot C_{j}^{1}$$
$$S_{j}^{2} = S_{j}^{1} \qquad \text{pentru celelalte valori } j$$

Pasul 3: Se repetă aceeaşi procedură cu grupări de câte 8 ranguri::

$$S_{j+1}^{3} = S_{j+1}^{2} \oplus C_{j}^{2} \qquad j = 3, 11, 19, \ldots, n\text{-}5$$
$$S_{j+2}^{3} = S_{j+2}^{2} \oplus S_{j+1}^{2} \cdot C_{j}^{2}$$
$$S_{j+3}^{3} = S_{j+3}^{2} \oplus S_{j+2}^{2} \cdot S_{j+1}^{2} \cdot C_{j}^{2}$$
$$S_{j+4}^{3} = S_{j+4}^{2} \oplus S_{j+3}^{2} \cdot S_{j+2}^{2} \cdot S_{j+1}^{2} \cdot C_{j}^{2}$$
$$C_{j+4}^{3} = C_{j+4}^{2} + S_{j+4}^{2} \cdot S_{j+3}^{2} \cdot S_{j+2}^{2} \cdot S_{j+1}^{2} \cdot C_{j}^{2}$$
$$S_{j}^{3} = S_{j}^{2} \qquad \text{pentru celelalte valori j}$$

Pasul k: În general, dimensiunea grupărilor în pasul k este 2^k, şi asimilarea se exprimă prin următoarele ecuaţii $\left(j = 2^k - 1, 3, 2^k - 1, 5, 2^k - 1, \ldots, n - 1 - 2^{k-1}\right)$:

$$S_{j+1}^{k} = S_{j+1}^{k-1} \oplus C_{j}^{k-1}$$
$$S_{j+2}^{k} = S_{j+2}^{k-1} \oplus S_{j+1}^{k-1} \cdot C_{j}^{k-1}$$
$$\ldots$$
$$S_{j+2^{k-1}}^{k} = S_{j+2^{k-1}}^{k-1} \oplus S_{j+2^{k-1}-1}^{k-1} \cdot S_{j+2^{k-1}-2}^{k-1} \cdot \ldots \cdot S_{j+1}^{k-1} \cdot C_{j}^{k-1}$$
$$C_{j+2^{k-1}}^{k} = C_{j+2^{k-1}}^{k-1} + S_{j+2^{k-1}}^{k-1} \cdot S_{j+2^{k-1}-1}^{k-1} \cdot \ldots \cdot S_{j+1}^{k-1} \cdot C_{j}^{k-1}$$
$$S_{j}^{3} = S_{j}^{2} \qquad \text{pentru celelalte valori j}$$

În Tabel 1. 2 este arătat un exemplu de adunare piramidală, ce corespunde adunării cu sumatorul CdSumA exemplificată în Tabel 1. 1.

I	7	6	5	4	3	2	1	0
A	0	1	0	1	0	1	0	1
B	1	0	0	0	0	0	1	1
S_i^0	1	0	0	1	0	1	1	0
C_i^0	0	0	0	0	0	0	0	1
S_i^1	1	1	0	1	0	1	0	0
C_i^1	0		0		0		1	
S_i^2	1	1	0	1	1	0	0	0
C_i^2	0				1			
S_i^3	1	1	0	1	1	0	0	0
C_i^3	0							

Tabel 1. 2

Organizarea la nivel înalt a unui sumator piramidal pe 8 biţi este arătată în Fig. 1. 26 şi detaliile logice ale câtorva din modulele sumatorului sunt prezentate în Fig. 1. 27.

BUPT

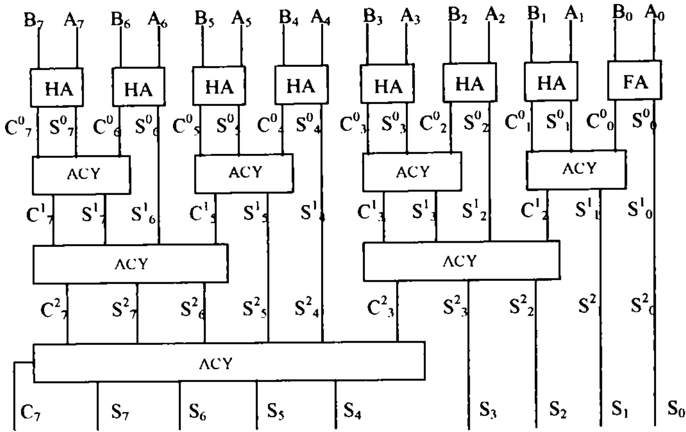


Fig.1. 26

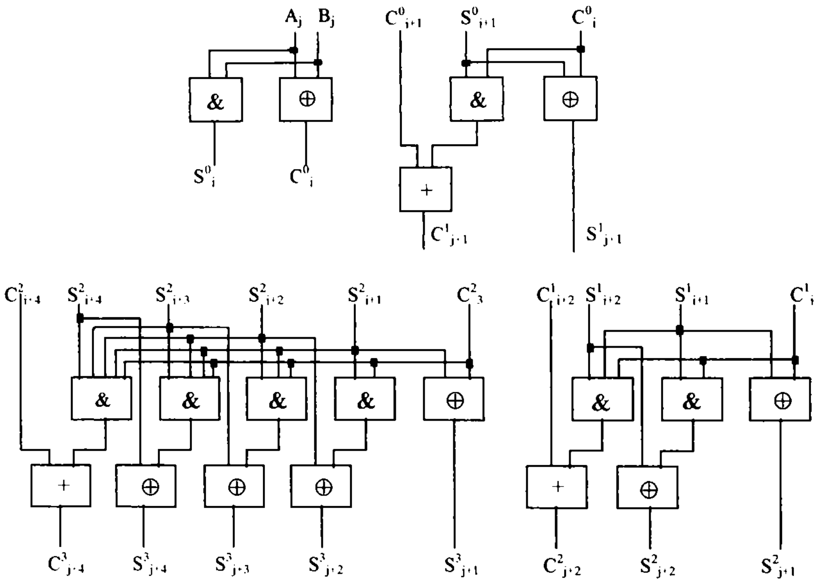


Fig.1. 27

În [POPE97B] sunt date detaliile de implementare ale sumatoarelor cu salvarea semnalelor CY (*Carry-Save Adder*) foarte eficiente pentru implementarea dispozitivelor de înmulţire secvenţială.

BUPT

1.5 Probleme ale testării structurilor de însumare

Pe măsură ce s-au îmbunătăţit performanţele dispozitivelor, a crescut dramatic densitatea de integrare a circuitelor şi costul dispozitivelor a scăzut. Odată cu aceste dezvoltări, a devenit tot mai importantă fiabilitatea circuitelor. Întrucât la circuitele VLSI, densitatea de integrare a crescut mult mai rapid decât numărul terminalelor de acces, s-au deteriorat posibilităţile de generare a *pattern*-urilor de test şi de procesare a simulării defectelor. Dificultatea apărută, poate fi contracarată prin dezvoltarea unor algoritmi de generare a *pattern*-urilor de test mai rapizi şi mai eficienţi, sau prin utilizarea unor tehnici de proiectare pentru creşterea testabilităţii [BHAT89].

1.5.1 Modelarea defectelor

Porţile logice sunt realizate cu tranzistoare, care, funcţie de tehnologia de realizare, pot fi de tipul *bipolar*, sau de tip *tranzistoare metal oxid semiconductor cu efect de câmp* (MOSFET, sau simplu MOS). Exemple de familii logice bazate pe tranzistoare bipolare sunt familia TTL, respectiv familia ECL. Referitor la cele bazate pe MOSFET, există familiile p-MOS, n-MOS, şi CMOS .
Cu toate că TTL şi ECL sunt importante în aplicaţiile de viteză, dimensiunea lor de integrare este limitată de căldura ce se generează prin consumul de putere în interiorul acestora, şi de dimensiunea mare a porţilor din aceste familii. În contrast, familiile de circuite MOS sunt potrivite pentru integrarea circuitelor LSI, VLSI. Majoritatea circuitelor LSI şi VLSI de azi, sunt implementate cu MOS-uri. Utilizarea circuitelor MOS în implementarea circuitelor LSI, şi VLSI a ridicat noi probleme de testare [DAMP87].

Prin *eroare* (observabilă) se denumeşte o instanţă a unei operări incorecte a unităţii ce se testează (*Circuit under test* - CUT). Acest concept are înţelesuri diferite în contextul unor niveluri diferite. De exemplu, la nivelul programelor de diagnoză o eroare observată poate fi dată de un rezultat diferit al unei operaţii aritmetice, în timp ce pentru un echipament de testare automat, o eroare înseamnă de obicei o valoare binară incorectă.

Cauzele erorilor observate pot fi: erorile de proiectare, erorile de fabricaţie, defectele de fabricaţie şi căderile fizice.

Exemple de erori de proiectare:
- specificaţii incorecte sau inconsistenete;
- mapare incorectă între diferite niveluri de proiectare;
- violarea regulilor de proiectare

Erorile apărute pe durata fabricaţiei includ:
- componente greşite (*wrong components*);
- cablare greşită
- scurtcircuite cauzate de realizarea unor lipiri greşite.

Defectele de fabricaţie nu sunt atribuite în mod direct unei erori umane; ele mai degrabă provin de la un proces de fabricaţie imperfect. Localizarea acurată a acestor tipuri de erori este importantă pentru îmbunătăţirea domeniului de fabricaţie.
Căderile fizice apar pe parcursul duratei de viaţă a sistemului, şi ele se datorează uzurii

BUPT

şi/sau factorilor de mediu. De exemplu, conectorii de aluminiu din interiorul împachetării unui circuit integrat se subţiază cu timpul şi se pot întrerupe datorită migrării electronilor şi coroziunii. Factorii de mediu: temperatura, umiditatea şi vibraţiile accelerează îmbătrânirea componentelor. Radiaţiile cosmice şi particulele α pot produce căderi la chip-uri ce conţin RAM-uri de mare densitate. Căderile fizice ce apar imediat după fabricare sunt denumite căderi infantile (*infancy failures*).

Erorile de fabricaţie, defectele de fabricaţie şi căderile fizice sunt referite în mod colectiv prin termenul de *physical faults* - PF.

Acestea pot fi clasificate [VLĂD89B] în funcţie de stabilitatea lor în timp în felul următor:

- *permanente*, adică prezente tot timpul după ce au apărut şi ele nu-şi schimbă natura pe tot parcursul procesului de testare;
- *intermitente*, adică există doar pe parcursul anumitor intervale;
- *tranziente*, a căror apariţie la un moment dat este determinată de o schimbare temporală a unui factor de mediu.

Întrucât multe defecte intermitente devin în timp defecte permanente, este foarte importantă, din punct de vedere fiabilistic, detecţia lor timpurie. Dar, întrucât un asemenea defect poate dispare în momentul aplicării unui test, nu există măsuri fiabilistice pentru detectarea apariţiei lor.

PF pot fi clasificate ca fiind de tip *logic* şi de tip *parametric*. Un defect logic, este un defect ce cauzează schimbarea funcţiei logice a unui element de circuit, sau schimbarea valorii unui semnal de intrare; defectul parametric, alterează mărimea unui parametru de circuit, cauzând modificarea unui factor ca şi viteza circuitului, curentul, sau tensiunea.

În general un defect este detectat prin observarea unei erori cauzate de defect.

PF legate de întârzierile semnalelor în circuit, care, şi ele afectează funcţionarea circuitului, sunt numite defecte *de întârziere*. De obicei, defectele de întârziere, afectează numai timpul de operare a circuitului, ceea ce poate produce hazard şi curse critice în circuit [UNGE95].

În Fig.1. 19, se arată schema unui invertor implementat cu un tranzistor de tip n-p-n. Când intrarea x este la tensiune înaltă, ieşirea z este la tensiune joasă; când x este la tensiune joasă, z este la tensiune înaltă. O întrerupere la colector, sau bază ar cauza ca ieşirea z să fie permanent la nivel ridicat de tensiune, adică să fie pusă pe 1 (s-a-1).

Pe de altă parte, un scurtcircuit între colector şi emitor, ar cauza ca z să fie permanent la nivel de tensiune coborât, adică să fie pusă pe 0 (s-a-0). Aceste defecte sunt numite defecte de *punere pe* 0, *sau 1* (*stuck-at faults*).

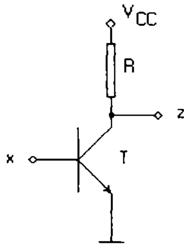


Fig.1. 19

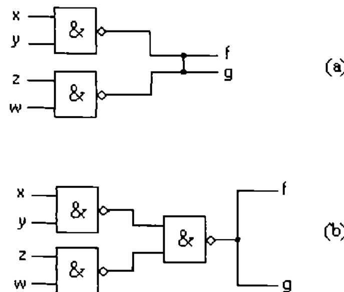


Fig.1. 20

BUPT

Defectele în care două linii sunt puse în scurtcircuit, sunt numite defecte de *scurtcircuit* (*bridging faults*). Efectul scurtcircuitului va fi determinat de tehnologia de realizare a circuitului. În general va domina fie valoare 0L, fie valoare 1L.
Dacă două linii sunt în scurtcircuit şi este dominant nivelul de tensiune inferior, amândouă linii se înlocuiesc prin poarta AND a celor două linii, aşa cum este arătat în Fig.1. 20. Acest efect este acelaşi cu cel al lui AND *cablat* utilizat la porţile TTL.
Dacă domină nivelul de tensiune superior, ambele linii sunt înlocuite de porţi OR între valorile celor două linii. Porţile familiei ECL prezintă avantajul că legând împreună ieşirile unor porţi ECL, se realizează funcţia OR între valorile acestor ieşiri.
Deci, în porţile logice implementate cu *ECL*, defectul de scurtcircuit între linii de semnal, cauzează realizarea funcţiei OR între valorile acestor linii de semnal.

Pentru cele mai multe scopuri practice, defectele logice sunt modelate cu succes, prin utilizarea *modelului de punere pe 0 (1) (stuck-at fault)*, respectiv a celui de *scurtcircuit*. Cu toate acestea, nu toate defectele pot fi modelate prin aceste defecte clasice.

Pentru exemplificare vom considera următoarele exemple. În Fig.1. 21 se arată o poartă CMOS de tip NAND, ce are 2 intrări. Ieşirea circuitului este la nivel de tensiune scăzut, dacă şi numai dacă ambele intrări x_1 şi x_2 sunt la nivel ridicat.

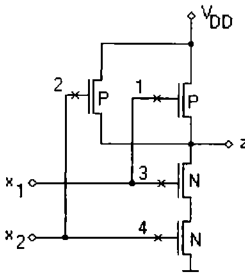


Fig.1. 21

În figură sunt indicate 4 defecte de întrerupere posibile (numerotate de la 1 la 4). Primul defect, numerotat cu 1, este cauzat de lipsa sau întreruperea canalului p. În prezenţa acestui defect, când intrarea x_1 este la nivel scăzut şi intrarea x_2 este la nivel ridicat, ieşirea z, devine în mod nedorit, o stare de impedanţă ridicată, ce reţine valoarea ei logică anterioară.
Durata de timp cât este reţinută această stare este funcţie de curentul de zăvorâre ce se scurge prin nod. În Tabel 1. 3 este reprezentat tabelul de adevăr pentru toate situaţiile ce se analizează, pentru poarta NAND CMOS.

Prin urmare, aceste defecte de întrerupere cauzează transformarea circuitelor combinaţionale în circuite secvenţiale, şi deci ele nu pot fi modelate ca şi defecte clasice [SHENG94].

x_1 x_2	z normal	z întrerup. la 1	z întrerup. la 2	z întrerup. la 3, sau 4
0 0	1	1	1	1
0 1	1	Stare anterioară	1	1
1 0	1	1	Stare anterioară	1
1 1	0	0	0	Stare anterioară

Tabel 1. 3

De asemenea nu pot fi modelate prin modelele clasice, defectele de tip *legătură* (*crosspoint*), ce apar în cazul matricilor logice programabile (PLA). Aceste defecte de legătură, se referă de fapt la defectele: *lipsă dispozitiv* (*missing device*), respectiv *extra dispozitiv* (*extra device*), ce sunt semnalate la PLA-uri.

Pentru a asigura funcţionarea unui sistem, trebuie să se poată detecta prezenţa unui defect în sistem, să se poată face localizarea şi izolarea componentei defecte. Prima

BUPT

procedură este numită *procedură de detecție*, iar cea de a doua, este numită *procedură de localizare, izolare, și diagnoză a defectului* [POPE92A]. Aceste sarcini se realizează prin *teste*.
Un test este o procedură de a detecta, sau a localiza defecte. Testele se clasifică în teste pentru detecție, și teste pentru diagnoză. Un *test pentru detecție*, semnalizează numai dacă un circuit este defect, sau este fără defect; nu spune nimic despre identitatea defectului, dacă el există. Un *test de diagnoză*, asigură localizarea și identificarea tipului de defect, precum și alte informații legate de defect. Cantitatea de informație asigurată de testul pentru diagnoză, este numită *rezoluția testului de diagnoză*; un test de detecție, este un test de diagnoză a cărui rezoluție a diagnozei este 0.

Circuitele logice se testează prin aplicarea unei secvențe (cazul circuitelor secvențiale), sau a unui set de *pattern*-uri de intrare (cazul circuitelor combinaționale), ce produc răspunsuri eronate, atunci când circuitul este defect, după care, se face compararea răspunsurilor obținute, cu răspunsurile corecte așteptate. Un asemenea *pattern* de intrare, utilizat în testare, este numit *pattern de test*. În general, testarea unui circuit logic presupune aplicarea mai multor *pattern*-uri de test, fie sub forma unor seturi de teste, fie sub forma unor secvențe de teste. O secvență de teste, reprezintă o serie de *pattern*-uri de test la care contează ordinea în care se aplică aceste *pattern*-uri. *Pattern*-urile de test, împreună cu răspunsurile de ieșire, sunt numite uneori *date de test*.

Dacă într-un circuit există o singură defecțiune, defecțiunea este numită *defecțiune singulară*.

Dacă la același moment, există două sau mai multe defecțiuni, atunci setul de defecțiuni este numit *defect multiplu*.

Modelul defecțiunilor singulare de tip s-a-0 (1) (*Single Stuck-at Faults* SSF) este modelul clasic utilizat în testarea Circuitelor digitale.

Pentru un circuit cu n linii, există cel mult $2 \cdot n$ defecte singulare de tip blocare la 0 sau 1. Pentru defecte multiple, întrucât fiecare linie poate fi fără defect, sau s-a-0, sau s-a-1, numărul defectelor posibile crește dramatic, la valoarea $3^n - 1$. Un circuit cu 100 linii va putea avea aproximativ $5 \cdot 10^{47}$ defecte. Sunt prea multe defecte, deci testarea acestor defecte multiple este impracticabilă. Pentru multe aplicații, s-a dovedit că modelarea defectelor singulare este suficientă. De exemplu, pentru un circuit combinațional, orice set complet de teste pentru detecția defectelor singulare, s-a dovedit că acoperă cel puțin 98% din defectele multiple formate din mai puțin de *6* defecte [AGAR81].

Întrucât multe defecte diferite pot provoca malfuncționarea circuitului în aceeași manieră, este convenabil de grupat aceste defecte echivalente, în *clase de echivalență*.
Două defecte f și g se spune că sunt echivalente funcțional dacă răspunsul circuitului în prezența defectului f este același cu răspunsul circuitului în prezența defectului g, indiferent de combinația de intrare aplicată circuitului respectiv. În general pentru o poartă cu valoare de control c și inversiune i, toate defectele s-a-c de pe intrare sunt funcțional echivalente cu defectul pe ieșire $s-a-(c \oplus i)$. Reducerea setului de defecte analizate bazată pe acest principiu este numită ***equivalence fault collapsing***.

În detecția și localizarea defectelor, se iau în considerare numai defectele reprezentative ale fiecărei clase de echivalență. Este important faptul că setul de teste pentru diagnoză permite diagnoza defectelor doar până la nivelul claselor de echivalență stabilite și el include setul de teste pentru detecție.

Două defecte f și g se spune că sunt *funcțional echivalente sub setul de teste T*, dacă răspunsul circuitului la orice test $t \in T$ în prezența oricăruia din ele este același.

BUPT

Echivalenţa funcţională implică echivalenţa sub orice set de test, pe când echivalenţa sub un anumit set de test nu implică echivalenţa funcţională.

Dacă prin testare este urmărită doar detecţia defectelor, nu şi diagnoza lor, numărul defectelor urmărite la nivelul circuitului poate fi redus pe baza relaţiilor de dominanţă.

Dacă T_g este setul tuturor testelor ce detectează defectul g, se spune că un defect *f domină* defectul g dacă şi numai dacă defectele f şi g sunt funcţional echivalente sub T_g . Deci, orice test t ce detectează defectul g va detecta la aceleaşi ieşiri primare şi defectul f. Prin urmare, din punctul de vedere al detecţie defectelor, nu este necesară considerarea defectului dominant f, întrucât obţinerea unui test ce detectează g va asigura şi detecţia lui f.

În general pentru o poartă cu valoare de control c şi inversiune i, defectul la ieşire $s-a-(\bar{c}+i)$ domină toate defectele de intrare $s-a-\bar{c}$.

Reducerea setului de defecte analizate bazată pe acest principiu este numită ***dominance fault collapsing.***

La alegerea unui anumit model de defecte este important ca defectele modelului ales să fie dominate de defectele celorlalte modele, asigurându-se în felul acesta detecţia defectelor celorlalte modele pe baza setului de teste determinat utilzând modelul ales. Cel mai bun model cu asemenea proprietăţi pare să fie modelul SSF. Cu toate că nu este un model universal, utilitatea sa se bazează pe următoarele considerente:

- poate reprezenta multe defecte fizice diferite
- este independent de tehnologie întrucât conceptul blocării unei linii de semnal la o anumită valoare logică poate fi aplicat oricărui model structural
- Experienţa a arătat că teste ce detectează defecte SSF, detectează şi defecte neclasice; numărul defectelor SSF dintr-un circuit este mic în comparaţie cu cel al altor modele; în plus aceta poate fi redus pe baza tehnicilor *fault-collapsing*.
- SSF poate fi utilizat pentru modelarea altor tipuri de defecte.

În acest sens, pe baza tehnicilor *fault-collapsing* [KARK94] (de dominanţă şi echivalenţă) este demonstrată [ABRA96] următoarea teoremă:

Teoremă 1.1 Într-un circuit combinaţional C iredondant, orice set de teste ce detectează toate defectele singulare de tip s-a-0 (1) de pe intrările primare şi de pe ramurile de fan-out ale circuitului, detectează toate SSF-urile circuitului C.

Intrările primare şi ramurile de fan-out sunt numite *checkpoints*. Numărul lor este: $r = I + \sum_i (f_i - q_i) = I + (G+I)\cdot(f-q)$, unde G este numărul porţilor, I este numărul intrărilor primare, f este numărul de fan-out mediu ($f = \frac{\sum_i f_i}{G+I}$, f_i fiind numărul punctelor de fan-out asociate liniei i), iar q este fracţiunea surselor de semnal cu un singur fan-out ($q = \frac{\sum_i q_i}{G+I}$, q_i fiind 1 doar pentru liniile cu $f_i = 1$).

Astfel, numărul defectelor urmărite la nivelul unui circuit este redus de la $2n$ la $2r$. În general numărul ramurilor de fan-out este mult mai mare decât cel al intrărilor primare, putându-se face aproximarea: $r \approx (G+I)\cdot(f-q)$.

Fracţiunea de defecte urmărite ce au fost eliminate pe baza Teoremei 1.1 este:

BUPT

$1-\frac{2r}{2n}=1-\frac{f-q}{1+f-q}=\frac{1}{1+f-q}$. De notat că acest număr al defectelor *checkpoints* poate fi redus mai departe, prin aplicarea relaţiilor de dominaţă şi de echivalenţă structurii particulare a CUT.

1.5.1.1 Modele de întârziere

In procesul de evaluare a elementelor logice se face o separare între funcţia lor logică şi comportarea în timp a acestora (Fig.1. 22)

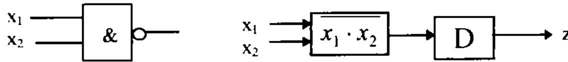


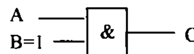
Fig.1. 22

Fig.1. 23 exemplifică pentru o poartă AND modelele de întârziere ce pot fi utilizate pentru calculul întârzierilor şi identificarea defectelor de întârziere. Ele sunt reprezentate în următoarea ordine:

- *(a)* Model cu întârziere nominală independentă de tranziţie *(Nominal transition-independent transport delay)*
- *(b)* Model cu întârzierea dependentă de tranziţie *(Rise and fall delay)*
- *(c)* Model cu întârziere ambiguă *(Ambiguous delay)*
- (d) Model cu întârziere inerţială (*Inertial delay – pulse suppression*)
- *(e)* Model cu întârziere inerţială (*Inertial delay)*

Cele mai uzuale medele fiind:

- Modelul întârzierii de transport (*Transport delay*) - modelul de întârziere de bază el specifică intervalul de timp ce separă schimbarea ieşirii de schimbarea intrării ce o provoacă şi foloseşte valori de obicei întregi ce exprimă multiplii de unităţi de timp.
- Modelul de întârziere unitar (*unit-delay model*) presupune că toate întârzierile din circuit sunt egale, ele fiind egalate cu unitatea.



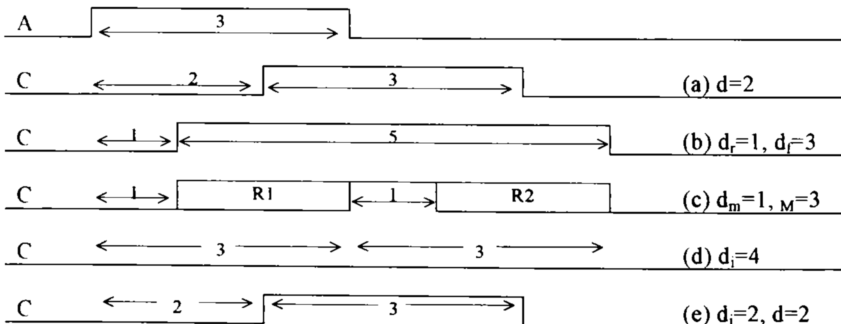


Fig.1. 23

BUPT

În modelarea întârzierilor se presupun:

1. cunoscută dependența întârzierii de direcția tranziției de ieșire rezultate.
2. cunoscute precis valorile întârzierilor

În Fig.1. 24 este ilustrat modelul hardware ce este utilizat frecvent ca și bază în testarea întârzierilor. La baza acestuia se află faptul că testarea defectelor de întârziere se face prin stimularea CUT cu o pereche de două *pattern*-uri de test.

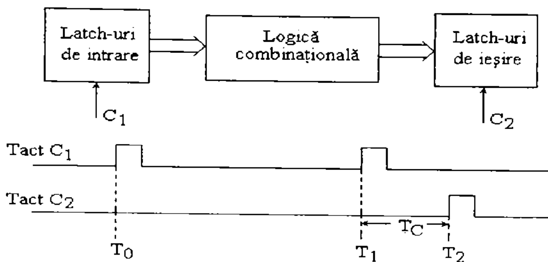


Fig.1. 24

La momentul T_0, se încarcă în latch-urile (presupuse fără *glitch*-uri) vectorul de inițializare V_1. După ce s-a permis stabilizarea tuturor semnalelor din circuit ca urmare a acțiunii lui V_1, se aplică vectorul V_2 prin activarea clock-ului C_1. In final, valorile logice ale ișirilor pimare sunt memorate de clock-ul C_2 în latch-urile de ieșire la momentul $T_2=T_1+T_C$, unde T_C indică perioada clock-ului sistem.

1.5.2 Tipuri de testări

Metodele de testare pot fi clasificate conform mai multor criterii. Tabel 1. 4 prezintă sumar atributele și terminologia celor mai importante metode de testare [ABRA96].

Criteriu	Atributul metodei de testare	Terminologia
Când este realizată testarea	• Concurent cu operarea normală a sistemului • Ca și activitate separată	*On-line testing* *Concurent testing* *Off-line testing*
Unde este sursa stimulilor	• Chiar în interiorul sistemului • Aplicate de un dispozitiv extern (testor)	*Self-testing* *External testing*
Ce se testează	• Erorile de proiectare • Erorile de fabricație • Defectele de fabricație • Căderile din rodaj • Căderile fizice	*Design verification testing* *Acceptance testing* *Burn-in* *Quality assurance testing* *Field testing* *Maintenance testing*
Ce este obiectul fizic ce se testează	• IC • Placă • Sistem	*Component-level testing* *Board-level testing* *System-level testing*
Cum sunt produși stimulii de test și/sau răspunsurile de test	• Obținuți din memorie	*Stored-pattern testing*

BUPT

Criteriu	Atributul metodei de testare	Terminologia
	• Generați pe parcursul procesului de testare	*Algorithmic testing* *Comparison testing*
Cum sunt aplicați stimulii de test	• Într-o ordine predeterminată	
	• În funcție de rezultatele obținute până la acel moment	*Adaptive testing*
Cât de repede sunt aplicați stimulii de test	• Mult mai rar decât operarea normală	*DC (static) testing*
	• La viteza de operare normală	*AC testing* *At speed testing*
Ce sunt rezultatele ce sunt urmărite	• Toate *pattern*-urile de ieşire	
	• Anumită funcție a *pattern*-urilor de ieşire	*Compact testing*
Ce linii sunt accesibile pentru testare	• Numai liniile de I/O	*Edge-pin testing*
	• I/O şi liniile interne	*Guided-probe testing* *Bed-of-nails testing* *Electron-beam testing* *In-circuit testing* *In-circuit emulation*
Cine verifică rezultatele	• Chiar sistemul	*Self-testing* *Self-checking*
	• Un dispozitiv extern (testor)	*External testing*

Tabel 1. 4

Testarea prin programe de diagnoză se realizează *off-line*, la viteza de lucru a sistemului şi la nivelul sistemului. Stimulii sunt generați chiar de sistemul ce lucrează în mod autotest. La sistemele cu logică de control microprogramată programele de diagnoză pot fi şi ele microprograme (*microdiagnostics*).

Este important de menționat faptul că pentru a se putea realiza rularea programelor de diagnoză trebuie ca o anumită parte a sistemului numită *hardcore* să fie funcțională. Stimulii sunt generați de *software* sau de *firmware* şi pot fi aplicați în mod adaptiv. Aceste programe de diagnoză sunt utilizate în cazul testării în exploatare (*field testing*), sau în cazul acțiunilor de mentenanță.

Emularea în circuit este o metodă de testare ce elimină necesitatea existenței *hardcore*-ului funcțional pentru rularea programelor de diagnoză. Această metodă este utilizată la testarea plăcilor cu µP, şi constă în extragerea pe durata testării a µP-lui de pe placă şi în accesarea conexiunilor µP-lui cu restul plăcii cu ajutorul unui testor extern. Testorul poate emula funcționarea µP-lui scos (de obicei prin utilizarea unui µP de acelaşi tip). Această configurație permite rularea programelor de diagnoză utilizând µP-ul şi memoria testorului.

La testarea *on-line*, întrucât stimulii sunt asigurați de *pattern*-urile obținute pe durata funcționării normale, stimulii şi răspunsurile sistemului nu sunt cunoscute în avans. Obiectul de interes al testării *on-line* nu este răspunsul în sine, ci anumite proprietăți ale răspunsului, proprietăți ce trebuie să rămână invariante pe durata

BUPT

funcționării normale.
De exemplu, în cazul unui decodificator, numai una dintre ieșirile sale trebuie să aibă valoarea 1L. Codul de operație al unui cuvânt instrucție (*opcode*) dintr-un set de instrucțiuni ale unui procesor este restricționat și el la un set de coduri de operații legale. Totuși, în cazul general nu există proprietăți atât de ușor definibile sau dacă există ele sunt greu de verificat. Astfel că abordarea generală a testării *on-line* este bazată pe tehnicile de proiectare fiabilă (*reliable design technique*), ce crează proprietăți invariante ce sunt ușor de verificat pe durata operării normale a sistemului. Un exemplu tipic în acest sens este utilizarea unui bit adițional de paritate la nivelul fiecărui octet de memorie. Acest bit este setat astfel încât să se realizeze o proprietate ușor de verificat, și anume el face ca fiecare octet extins (octetul original plus bitul de paritate) să aibă aceeași paritate (număr de unități modulo 2). Bitul de paritate este redondant întrucât el nu conține informație utilă operării normale a sistemului. Acest tip de informație redondantă este caracteristic sistemelor ce utilizează *coduri detectoare și corectoare de erori.*
O altă tehnică de proiectare fiabilă bazată pe redondanță este redondanța modulară ce constă în multiplicarea modulelor. Modulele de rezervă (ce au aceeași funcție, eventual cu implementări diferite) operează asupra aceluiași set de date de intrare și toate trebuie să producă aceleași răspunsuri. Sistemele *self-checking* au subcircuite numite *checker-e*, ce sunt dedicate testării acestor proprietăți invariante.

Testarea *guided-probe* este o tehnică utilizată în testarea plăcilor. Dacă apare o eroare pe parcursul teatării inițiale la conector (*edge-pin*) - această fază este de cele mai multe ori referită prin test GO/NO GO - testorul decide care linie internă va fi monitorizată și indică operatorului să plaseze o sondă pe linia respectivă. Testul este reaplicat. Principiul tehnicii constă în trasarea înapoi a propagării erorilor de-a lungul căilor circuitului. După fiecare aplicare a testului, testorul verifică rezultatele obținute pe linia monitorizată și determină dacă a fost identificat locul de apariție al defectului, sau dacă trebuie continuat procesul de trasare înapoi. Multe testere nu se limitează la monitorizarea unei singure linii, ci ele pot monitoriza un grup de linii, de obicei pinii unui IC.
Testarea *guided-probe* este o procedură de diagnoză secvențială, la care la fiecare pas se monitorizează un subset al liniilor interne ce sunt accesibile testorului. Anumite testere utilizează *patul de cuie* (*bad of nails*) ce permite monitorizarea liniilor accesibile într-un singur pas.

Scopul testării *in-circuit* este verificarea componentelor ce sunt deja montate pe placă. Un *IC clip* este utilizat de testorul extern permițându-i acestuia aplicarea *pattern*-urilor direct la intrările unui *IC* și observarea ieșirilor sale. Testorul trebuie să fie capabil să izoleze electronic *IC*-ul ce se testează față de exterior; de exemplu s-ar putea să trebuiască să asigure *pattern*-urile de intrare furnizate de alte componente.

Testarea algoritmică se referă la generarea *pattern*-urilor de intrare pe durata testării. Exemple tipice de generare hardware a stimulilor de intrare sunt numărătoarele și registrele de deplasare cu reacție. Generarea algoritmică a *pattern*-urilor reflectă capabilitatea anumitor testoare de a produce mai multe *pattern*-uri fixate. Combinația dorită este determinată de un program de control scris într-un limbaj orientat spre testare.
Testarea circuitelor logice presupune parcurgerea a două faze importante:

1. *generarea pattern-urilor de test* pentru circuitul ce se testează (*the test generation stage*), și
2. *aplicarea* acestor *pattern-uri de test* circuitului (*the test application stage*) împreună cu evaluarea răspunsului obținut de la CUT

BUPT

1.5.3 Generarea automată a testelor

Pentru circuitele LSI şi VLSI, este foarte importantă generarea automată a *pattern*-urilor de test, motiv pentru care în ultimii 20 de ani, au fost concentrate eforturile de cercetare, pentru dezvoltarea unor proceduri de test cât mai eficiente şi economice.

Generarea testelor (*Test generation* - TG) este procesul de determinare a stimulilor necesari pentru testarea unui sistem digital. Tehnicile de testare *on-line* nu reclamă TG. TG implică un efort mic când *pattern*-urile de intrare sunt asigurate printr-un registru de deplasare cu reacţie ce lucrează ca şi generator de secvenţe de *pattern*-uri pseudoaleatoare. În contrast, pentru testarea verificării proiectului şi pentru dezvoltarea programelor de diagnoză, este implicat un mare efort în *TG*.
Termenul *Automatic* TG – ATG) se referă la algoritmi TG care pornind de la un model al sistemului poate genera automat teste pentru acesta.

Calitatea testării, depinde mult de gradul de acoperire a defectelor, asigurat de setul sau secvenţa de test, precum şi de lungimea acestuia.

Acoperirea defectelor, realizată de un test, este dată de fracţiunea de defecte ce poate fi detectată, sau localizată pentru circuitul ce se testează la aplicarea testului respectiv. Acoperirea defectelor din circuit, pe care o realizează un test dat, se determină prin procesul *de simulare a defectelor*.

În cadrul procesului de *simulare*, fiecare *pattern* de test este aplicat atât circuitului fără defect, cât şi fiecărui circuit obţinut prin defectare pe baza defectelor posibile; este simulată fiecare comportare de circuit, şi este analizat fiecare răspuns de circuit, pentru a găsi defectele detectate de fiecare *pattern* de test. Simularea defectelor este utilizată de asemenea pentru producerea *dicţionarelor de defecte*, în care se adună informaţiile necesare pentru identificarea unui defect.

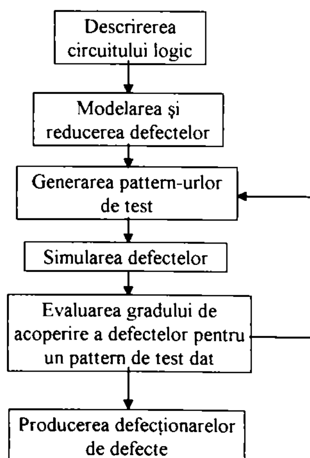


Fig.1. 25

Deci, procesul de generare a testelor, include modelarea şi reducerea defectelor, generarea *pattern*-urilor de test, evaluarea gradului de acoperire a defectelor, şi producerea dicţionarului de defecte. Procedura de generare a testelor este reprezentată în Fig.1. 25.

Primul pas, constă în dezvoltarea unui *dicţionar de defecte* pentru circuit (adică, în modelarea defectelor presupuse), şi în reducerea numărului de defecte în termenii relaţiei de echivalenţă între defecte. În general se adoptă modelul defectelor singulare s-a-1 (0), şi dicţionarul de defecte se generează direct din descrierea logică a circuitului, printr-o aranjare tabelară a defectelor distinctibile.
Pe urmă, se generează *pattern*-urile de test, pentru testarea defectelor din dicţionarul de defecte. *Pattern*-urile de test sunt pe urmă simulate pe circuitele defectate conform

BUPT

defectelor din dicţionar, şi se evaluează pe baza rezultatelor simulării (ce includ liste de defecte testate şi netestate), gradul de acoperire a defectelor [TOKA91]. Dacă acoperirea de defecte obţinută nu convine, se reia procesul de generare a *pattern*-urilor de test [SETH90][MAJU95] pentru defectele netestate, până se obţine o acoperire corespunzătoare a defectelor. În final, se completează dicţionarul de defecte, specificându-se informaţii suficiente pentru detecţia şi localizarea defectelor.

Pentru ca procesul de generare a testelor descris mai sus, să fie cât mai practic şi mai eficient, el este în general automat, sub forma unei colecţii de aplicaţii software. Pentru a asigura un grad ridicat de acoperire a defectelor, la un timp de calcul redus, există o interacţiune strânsă între procedurile de generare a testelor şi cele de simulare a defectelor. Pentru circuitele LSI, şi VLSI, este foarte important ca procedurile de generare a testelor, să fie cât mai eficientizate.

În ultimii ani s-au folosit tot mai mult tehnicile de testare pentru testabilitate (*design for testability* - DFT). Scopul lor este ca prin introducerea unor criterii de testabilitate încă din faza timpurie a proiectării, să se realizeze o reducere a costului testării. Astfel, consideraţiile de testabilitate au devenit atât de importante în ultima perioadă, încât ele pot chiar dicta întreaga structură a proiectului. În [ABRA96] sunt prezentate într-o manieră detailată toate aceste tehnici.

1.5.3.1 Generarea pattern-urilor de test

În Fig.1. 26 este dată o clasificare a modalităţilor practice de generare a testelor.

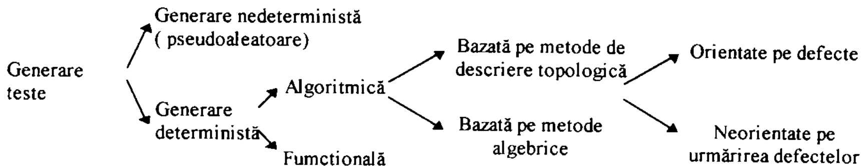


Fig.1. 26

În cazul TG *orientate pe defecte* [BOUB95], se încearcă generarea testelor ce pot detecta (şi localiza) orice defect specificat.

Metodele de generare algoritmică a testelor, ce nu sunt orientate pe defecte se bazează pe identificarea căilor critice şi ele prezintă avantajul că elimină pasul de simulare a defectelor din cadrul procesului de generare automată a stimulilor de test pentru CUT [ABRA96][POPE97A].

În cazul *testării funcţionale* se încearcă generarea unui test, care dacă reuşeşte indică faptul că sistemul realizează funcţia sa specificată. [ABRA96] baleiază toate tehnicile actuale de TG.

Au fost propuşi mulţi algoritmi pentru generarea testelor. Mulţi dintre aceşti algoritmi, cu importanţă mai mult teoretică decât practică, nu au fost utilizaţi în implementări ale aplicaţiilor practice. Numai puţine abordări sunt de interes practic.

Cel mai larg utilizat algoritm, este *algoritmul D* [VLĂD89B] [SHI91]. Acesta este un algoritm complet, în sensul că el generează test pentru orice circuit defect, dacă defectul este testabil. Totuşi, s-a dovedit că algoritmul D este ineficient pentru generarea testelor în cazul circuitelor ce conţin multe porţi XOR.

Pentru surmontarea acestei dificultăţi a algoritmului D, a fost propus în 1981 de către

BUPT

Goel [GOEL81], un nou algoritm numit PODEM (*Path-Oriented DEcision Making*).

În 1983, a fost dezvoltat algoritmul FAN (*a fan-out-oriented test-generation*), de către Fujiwara şi Shimono [FUJI83], ce generează testele mult mai eficient decât primele două. Aceşti trei algoritmi sunt descrişi în detaliu în [FUJI90].

Generarea testelor pentru circuite secvenţiale, este mult mai dificilă decât pentru cele combinaţionale. Aceasta se datorează în special săracei controlabilităţi şi observabilităţi ale acestora [POME94] [SAVA95].
Dar, controlabilitatea şi observabilitatea circuitelor secvenţiale (adică, testabilitatea lor), poate fi crescută la nivelul celei a circuitelor combinaţionale, prin utilizarea unor tehnici de proiectare, cunoscute sub denumirea de *proiectare testabilă* [BHAT89].
Aceste tehnici, permit reducerea metodelor de testare a circuitelor secvenţiale, la cele utilizate pentru testarea circuitelor combinaţionale.

Deci, dacă presupunem că circuitele secvenţiale ce se testează, sunt implementate pe baza acestor tehnici de proiectare testabilă, este suficient să se concentreze eforturile pentru dezvoltarea unor algoritmi de generare a testelor pentru circuitele combinaţionale [HEID91].

Metode algebrice de generare a testelor

Au fost dezvoltate mai multe metode algebrice de generare a testelor pentru un circuit dat. Printre acestea se numără metoda propoziţională a lui Poage (1963), forma normal echivalentă a lui Armstrong (1966), ecuaţiile cauză-efect ale lui Bossen şi Hong (1971), procedura SPOOF a lui Clegg (1973), metoda diferenţelor Booleene a lui Sellers (1968) şi dezvoltată de Ku şi Masson (1975), precum şi funcţia de descriere a structurii a lui Kinoshita [KINO80].

Toate aceste abordări, generează ecuaţii pentru circuitul fără defect, pe care le manipulează pentru generarea testelor. Din păcate, în general, este dificil de manipulat aceste ecuaţii algebrice. În cazul circuitelor mari, se poate întâmpla să fie necesare multe manipulări de astfel de ecuaţii mari pentru obţinerea testelor pentru un defect dat. Mai mult, datorită faptului că toate aceste metode algebrice generează toate testele pentru un defect dat, ele prezintă dezavantajul că reclamă mare cantitate de timp şi memorie, ceea ce le face impracticabile pentru circuite mari.

Metode topologice de generare a testelor

Sunt şi alte metode de generare a testelor, ce utilizează descrierea topologică la nivel de poartă, în loc de manipularea ecuaţiilor Booleene.
Generarea unui test pentru un defect $l\ s-a-v$ presupune parcurgerea a doi paşi fundamentali:

1 *Activarea defectului*. Presupune setarea valorilor la nivelul acelor PI ce determină ca linia l să aibă valoarea $\bar{v}$. Aceasta este o instanţă a problemei *de justificare a liniilor*, care se referă la găsirea unei asignări de valori pentru PI astfel încât să se realizeze setarea unei anumite valori la nivelul unei anumite linii din circuit.
2 *propagarea erorii rezultate* la o PO. Pentru urmărirea propagării erorii în circuit este necesar să se facă o urmărire concomitentă a valorilor de semnal atât în circuitul fără defect cât şi în cel cu defect. În acest sens se definesc valori logice compozite de forma v/v_f, unde v reprezintă valoarea logică din circuitul fără defect, iar v_f reprezintă valoarea logică din circuitul defect. În Fig.1. 27 sunt date aceste valori, simbolurile folosite pentru ele, şi modul de operare cu ele.

BUPT

v/v_f	
0/0	0
1/1	1
1/0	D
0/1	$\overline{D}$

AND	0	1	D	$\overline{D}$	x
0	0	0	0	0	0
1	0	1	D	$\overline{D}$	x
D	0	D	D	0	x
$\overline{D}$	0	$\overline{D}$	0	$\overline{D}$	x
x	0	x	x	x	x

OR	0	1	D	$\overline{D}$	x
0	0	1	D	$\overline{D}$	x
1	1	1	1	1	1
D	D	1	D	1	x
$\overline{D}$	$\overline{D}$	1	1	$\overline{D}$	x
x	x	1	x	x	x

Fig.1. 27

Structura unui algoritm de generare a testelor pentru detecția defectului $l\ s-a-v$ pentru un circuit fără fan-out este în Fig.1. 28

Begin

Setează la x toate valorile

Justify $(l,\overline{v})$

if $v = 0$ **then** Propagate (l, D)

else Propagate $(l,\overline{D})$

end

Fig.1. 28

Justificarea liniilor (Fig.1. 29) este un proces recursiv prin care valoarea de ieşire a unei porţi este justificată de valorile intrărilor sale, şi aşa mai departe până se ajunge la nivelul intrărilor primare:

Justify (l, val)

Begin

setează l la val

If *l este o PI* ***then Return***

c=valoare de control pentru l

i=inversiunea lui l

$inval = val \oplus i$

if $(inval = \overline{c})$

then *for every intrare j a lui l*

Justify $(j, inval)$

else

Begin

selectează o intrare j a lui l

Justify $(j, inval)$

end

end

Fig.1. 29

Selectarea unei intrări se face pe baza alegerii unuia din cuburile primitive ale porții ce duce la obținerea valorii de ieşire dorite.

Pentru propagarea erorii la PO a circuitului, trebuie activată unica cale de la l spre PO. Fiecare poartă a acestei căi are o singură intrare activată de defect, toate celelalte intrări fiind setate la valori necontrolabile. În felul acesta problema propagării defectului spre ieşire este transformată într-un set de probleme de justificare a liniilor (Fig.1. 30).

Propagate (l, err)

Begin

setează l la err

if *l este o PO* ***then Return***

BUPT

```
        k   fan-out-ul lui l
        c   valoarea de control a lui k
        i   inversiunea lui k
        for every intrare j a lui k alta decât l
            Justify (j, c̄)
        Propagate (k, err ⊕ i)
End
```

Fig.1. 30

În cazul circuitelor fără fan-out, problema justificării unei linii se rezolvă independent de justificarea celorlate linii.

Procedura de generare a testelor pentru cazul general al circuitelor cu fan-out se complică din două motive:

1. există mai multe posibilități de propagare a defectului activat pe baza procedurii de justificare a liniei. Şi la aceste circuite, odată ce s-a ales una din aceste căi de propagare, problema proagării erorii se reduce de fapt la un set de probleme de justificare a liniilor.
2. în general la circuitele cu fan-out (reconvergent) problemele de justficare a liniilor nu sunt independente, căutarea soluției implicând un proces de decizie ce se realizează printr-o strategie *Backtrace* ce permite explorarea sistematică şi completă a spațiului tuturor soluțiilor posibile şi revenirea (ce implică restaurarea calculului la o stare calculată înaintea luării deciziei incorecte) de la deciziile incorecte.

Procesul calculului valorilor de semnal impuse în circuit de o anumită asignare de valori este denumit proces de *implicare*. Prima implicare apare după stabilirea valorilor unice de activare şi de propagare a defectului urmărit.

În Fig.1. 31 este schițat în formă abstractă schema recursivă a unui algoritm backtracking de generare a testelor pentru un defect.

```
Solve()
Begin
    if Imply and check() = FAILURE then return FAILURE
    if ((eroare la PO) and toate liniile sunt justificate)
        then return SUCCESS
    if (nici o eroare nu poate fi propagată la o PO)
        then return FAILURE
    selectează o problemă nerezolvată
    Repeat
        Begin
            selectează o maieră neîncercată de rezolvare a ei
            if Solve() = SUCCESS then return SUCCESS
        end
    until s-au încercat toate modalitățile de rezolvare a ei
    return FAILURE
end
```

Fig.1. 31

Problema de la care se pleacă în generarea unui test pentru defectul $l\ s-a-v$ este de a justifica valoarea v a liniei l şi de a propaga eroarea de la l la o PO. La baza algoritmului este tehnica *Divide et Impera* (o problemă ce nu poate fi rezolvată direct este transformată în subprobleme ce se încearcă a se rezolva). Încercarea de rezolvare a unei probleme poate duce la SUCCESS sau FAILURE. La început algoritmul manevrează toate problemele ce au soluții unice, deci care pot fi rezolvate prin procesul de implicare. Acestea sunt procesate de procedura *Imply and check* (care verifică şi

BUPT

inconsistența). Rezultatul procedurii *Solve()* este:

- FAILURE dacă
  1. apare o inconsistență
  2. într-o stare consistentă nu se poate propaga eroarea spre o PO
- SUCCESS dacă se atinge scopul propus, adică dacă dacă s-a propagat eroarea la o PO şi s-au rezolvat toate problemele de justificare a liniilor.

Dacă *Solve()* nu poate determina imediat una din aceste valori, atunci selectează una din problemele curente nerezolvate (o problemă de justificare a liniilor sau de propagare a erorii). Această selecție, precum şi selecția unei modalități de rezolvare a ei, este în principiu arbitrară (din punctul de vedere al obținerii unei soluții pentru un anumit defect detectabil), cu toate că acest proces de selecție afectează eficiența algoritmului, precum şi vectorul de test ce se generează.
Există mai mulți algoritmi TG ai căror structură este similară lui *Solve()* [GOEL81] [FUJI83].

1.5.3.2 Simularea defectelor

Este o parte importantă a procesului de generare a testelor, fiind utilizată în scopul generării dicționarelor de defecte, şi pentru verificarea adecvalității testelor (seturi de stimuli, sau secvențe de stimuli), necesare pentru detecția şi localizarea defectelor logice.
Mai mult, simularea defectelor este adesea necesară pentru determinarea acoperirii defectelor pentru un test dat, adică pentru găsirea tuturor defectelor detectate de un test dat.
Simularea defectelor este utilizată de asemenea pentru analiza operaționalității unui circuit în diferite condiții de defectare, pentru considerarea comportării circuitului, neconsiderată de proiectant. De exemplu, defectele pot crea hazarduri, ce nu există în circuitul fără defect, sau să schimbe un circuit combinațional într-unul secvențial. Pentru analiza acestor defecte este de dorit să se utilizeze analiza de timp a circuitului, care cere simularea defectelor de timp.

Pentru circuite foarte mari, aceste metode de simulare bazate pe software, sunt foarte consumatoare de timp şi costisitoare. Soluționarea acestei situații se face utilizând simulatoare hardware, computere spacializate, cu paralelism înalt şi programabile, având o viteză şi o capacitate de sute, sau mii de ori mai mare decât a simulatoarelor existente.

1.5.3.2.1 Metodologia de simulare

Simulatoarele pot fi de 2 tipuri: *Simulatoare realizate prin compilare* (*Compiler-driven simulators*), abreviate CDS, şi *simulatoare bazate pe tabele şi direcționate de evenimente* (*Table-driven event-directed simulators*), abreviate prin TDS.
In TDS, descrierea circuitului logic ce se simulează este memorată sub formă de tabele în computerul gazdă. Tabelele sunt accesate când este necesar, de programul de simulare. Programul de simulare este independent de circuit, dar pentru fiecare circuit distinct ce se simulează este necesar un nou set de tabele.
In contrast, CDS au circuitul descris implicit în program, sub formă de cod compilat. Primul pas în CDS, este *nivelarea ierarhică a circuitului* (*levelizing*). Pentru aceasta definim nivelul logic al elementului *i* prin nivel(i), definirea făcându-se în mod recursiv după cum urmează:

BUPT

- *nivel(i)* $= 0$, pentru toate intrările primare şi liniile de stare
- *nivel(i)* $= n+1$, dacă *nivel(j)* $< n+1$ pentru toţi predecesorii imediaţi, *j*, ai lui *i*.
- *nivel(k)* $= n$ pentru un anumit predecesor imediat *k* a lui *i*

Dacă elementele de circuit (porţi şi bistabile), sunt simulate în ordine ascendentă a valorii nivelului lor logic, valoarea fiecărui element poate fi determinată complet, întrucât când se evaluează un element logic din nivelul *k*, valorile logice a tuturor intrărilor sale au fost deja evaluate.

După nivelarea ierarhică a circuitului, CDS translatează descrierea circuitului, într-un cod maşină executabil de forma unui program scris în limbaj de asamblare. Simularea circuitului logic este realizată prin executarea codului compilat, în ordine ascendentă a valorii nivelului logic. CDS sunt utilizate de obicei pentru simularea circuitelor logice într-un model cu "0" întârziere, şi deci manevrează numai circuite secvenţiale sincrone [POME95A], unde condiţiile de hazard sunt adesea ignorate.

O simulare mai exactă se obţine cu TDS. Un simulator TDS modelează comportarea dinamică a circuitului simulat, actualizând calculele sale la intervale de timp succesive şi uniforme: 1T, 2T, 3T ..., unde T este ales egal cu timpul mediu de întârziere.
Un eveniment este o schimbare în valoare a unui semnal de linie. Ieşirea unui element va schimba valoarea numai când una sau mai multe intrări s-au schimbat în intervalul precedent. Deci, un element trebuie simulat numai când un eveniment apare la una din intrările sale. Când apare un eveniment, la fiecare element la care această linie are fan-out, există posibilitatea de apariţie a unui nou eveniment. Trebuie simulate numai aceste elemente potenţiale active, ce sunt succesorii unei linii semnal unde a apărut un eveniment. Această tehnică de simulare, bazată pe această observaţie, este numită *simulare condusă de evenimente* (*event- directed*).
Simulatoarele de defecte, deductive şi concurente, sunt de tip TDS. In contrast, simulatoarele de defecte paralele, sunt de tip CDS.

1.5.3.2.2 Simularea paralelă a defectelor

Un computer are mai multe instrucţii orientate pe bit, ce includ instrucţii logice ca şi AND, XOR, OR, NOT. In timpul executării unor astfel de instrucţii, biţii unui cuvânt calculator sunt manipulaţi în mod identic şi independent. Instrucţiile AND, OR, XOR operează pe o pereche de operanzi; bitul *i* al fiecărui operand este combinat pentru obţinerea bitului *i* al rezultatului.

Un avantaj al adoptării unei vederi orientate pe bit al conţinutului cuvintelor unui computer îl constitue faptul că, prin memorarea mai multor termeni pe cuvânt, memoria poate fi utilizată cu multă eficienţă. Un alt avantaj este dat de posibilitatea calculului paralel, conferită de setul de instrucţii ce operează concurent, pe toţi biţii unui cuvânt; motiv pentru care, acest tip de simulare a defectelor, bazată pe procesarea paralelă a operaţiilor orientate pe bit, este numită simulare paralelă a defectelor (*Parallel Fault Simulation* – PFS).

PFS, este o metodă veche, ce a fost larg acceptată în sistemele automate de generare a testelor. Reprezentative pentru această abordare sunt: Analizorul secvenţial, Simulatorul IBM utilizat în proiectarea computerului Saturn şi simulatorul TEGAS.

BUPT

In PFS, dacă se manevrează un cuvânt, sau un şir are n biţi, atunci n probleme diferite pot fi procesate în paralel.
Dacă dorim să simulăm un circuit pentru m defecte diferite, atunci trebuie făcute [m/n] treceri, cu n defecte simulate la fiecare trecere, unde [x] este cel mai mare întreg mai mare sau egal cu x. Pentru a realiza simularea, printr-o *procedură de injecţie* a defectului (*Fault injection*), se injectează un efect logic al defectului în calculul unui element defectat [KANA95].

Injecţia defectelor se face pentru fiecare linie s, şi presupune să se asocieze la linia s două măşti: *mask(s)*, şi *fvalue(s)*.
Fiecărui defect injectat în linia s, îi corespunde o poziţie de bit, unică, în măştile *mask(s)* şi *fvalue(s)*. Această poziţie de bit trebuie să fie diferită de poziţiile de bit din măştile corespondente a oricăruia din cele $n-1$ defecte rămase, ce se simulează paralel cu defectul de pe linia s. Aceste măşti sunt definite după cum urmează:

- $mask(s)_i = 1$, dacă există un defect pe linia de semnal s, când se simulează bitul i al computerului.
- $mask(s)_i = 0$, în celelalte cazuri
- $fvalue(s)_i = 1$, dacă defectul pe s este s-a-1
- $fvalue(s)_i = 0$, dacă defectul pe s este s-a-0; unde indicele i indică poziţia de bit

Aceste 2 măşti sunt utilizate pentru injectarea defectelor în fiecare bit al cuvântului s, formându-se astfel, *cuvântul mască* s', procesul fiind modelat în Fig.1. 32.
Relaţia de formare a lui s' este:

$$s' = s \cdot \overline{mask(s)} + mask(s) \cdot fvalue(s);$$

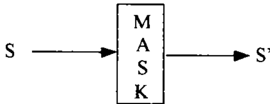


Fig.1. 32

Măştile utilizate pentru injectarea defectelor, sunt generate în timpul procesării circuitului pentru simularea defectelor.

Exceptând cheltuielile cerute de generarea măştilor şi realizarea injecţiei defectelor, rezultă că, prin utilizarea acestei metode, n defecte pot fi procesate aproape la fel de rapid ca şi unul singur.

Pentru exemplificare, presupunem o poartă AND cu 2 intrări şi cele 4 defecte asociate porţii sunt date în (Fig.1. 33).
Cele 4 defecte sunt spacificate în Fig.1. 33.a, şi valorile măştilor asociate cu liniile de semnal A, B, şi C sunt definite în Fig.1. 33.b.
De exemplu, mask(B) = [00100], întrucât linia B are asociată poziţia de bit 3, şi fvalue(B) = [00100], pentru că defectul este C s-a-1, şi este procesat în bitul 3. Similar, întrucât liniei C îi sunt asociate 2 defecte (C s-a-0 şi C s-a-1), ele se injectează pe biţii 4 şi 5 ai cuvântului C, având deci mask(C) = [00011] cu $fvalue(C)_4 = 0$ şi $fvalue(C)_5 = 1$.
După ce s-au generat măştile pentru injectarea defectului, se realizează PFS.
Presupunem că vrem să simulăm un pattern de intrare cu A = 1 şi B = 0. Pentru aceasta, se iniţializează prima dată cuvintele A şi B, la valorile A = [11111] şi B = [0000]. Simularea este trecută de la intrările primare, spre ieşirile primare. Pentru a evalua cum se cuvine valoarea logică a fiecărui element, acestea vor fi ordonate, adică nivelate ierarhic. Detaliile de ordonare se vor descrie mai târziu.

BUPT

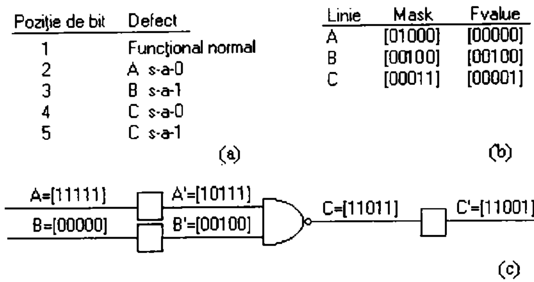


Fig.1. 33

In Fig.1. 33.c, A şi B sunt mascate pentru a forma A' şi B'.

$$A' = A \cdot \left(\overline{mask(A)}\right) + mask(A) \cdot fvalue(A) =$$
$$= [11111] \cdot \overline{[01000]} + [01000] \cdot [00000] =$$
$$= [11111] \cdot [10111] + [00000] = [10111]$$
$$B' = B \cdot \left(\overline{mask(B)}\right) + mask(B) \cdot fvalue(B) =$$
$$= [00000] \cdot \overline{[00100]} + [00100] \cdot [00100] =$$
$$= [00000] + [00100] = [00100]$$

Pe urmă, se simulează poarta AND:

$$C' = \overline{A \cdot B} = \left(\overline{[10111] \cdot [00100]}\right) = \overline{[00100]} = [11011]$$

Pe urmă, se face mascarea lui C:

$$C'' = C' \cdot \left(\overline{mask(C')}\right) + mask(C') \cdot fvalue(C') =$$
$$= [11011] \cdot \overline{[00011]} + [00011] \cdot [00001] =$$
$$= [11000] + [00001] = [11001]$$

Biţii 3 şi 4 ai ieşirii C', fiind diferiţi de bitul 1 al funcţionării normale, acest *pattern* de test va detecta defectele ce corespund acestor biţi; adică B s-a-1 şi C s-a-0.

1.5.3.2.3 Simularea deductivă a defectelor

Simularea deductivă a defectelor (*Deductive Fault Simulation* – DFS) a fost dezvoltată de Armstrong (1972) şi a fost utilizată în sistemul LAMP. Această metodă constă în simularea explicită doar a comportării circuitului fără defect, şi deducerea simultană din starea curentă bună a circuitului, a tuturor defectelor detectabile pe orice linie internă sau de ieşire în timpul stării curente.

Utilizând conceptul simulării deductive, putem calcula toate defectele detectabile la acelaşi moment, şi astfel, pentru fiecare *pattern* de test aplicat, este necesar un singur pas de simulare. Timpul de simulare/trecere pentru DFS, pare să fie mult mai mare decât timpul de simulare /trecere la PFS.

In 1974, Chang a arătat experimental că DFS este mai rapidă decât PFS în situaţia când se simulează un circuit mare, cu număr mare de defecte. Simularea paralelă a defectelor este mai rapidă decât cea deductivă numai pentru circuite secvenţiale mici, cu mai puţin de 500 de porţi. Totuşi, comparaţiile indică că un simulator paralel poate fi proiectat cu mai puţină memorie decât unul deductiv.

BUPT

Un DFS simulează circuitul fără defect, şi calculează *liste de defecte*.

In acest sens, o listă de defecte asociată cu linia A, notată L_A, conţine numele sau indexul fiecărui defect ce produce o eroare pe linia A, când circuitul este în starea lui logică curentă. Adică, fiecare defect din listă, înserat singur în circuit , ar cauza complementarea stării bune a semnalului de linie asociat.
Aceste liste de defecte sunt propagate nivel cu nivel prin circuit, de la nivelul intrărilor primare, spre cel al ieşirilor primare. Astfel, se generează o listă de defecte pentru fiecare semnal de linie, şi aceasta, este actualizată după necesar, cu fiecare schimbare în starea logică a circuitului. Un DFS utilizează o procedură selectivă, directată de evenimente (*the event-directed or selective procedure*), prin care ieşirea de stare a unui element este calculată numai când apare un eveniment la una din intrările sale.

In DFS, sunt două tipuri de evenimente:

- *evenimente logice*, ce apar când un semnal de linie ia o nouă valoare logică
- *evenimente listă*, ce apar când se schimbă o listă de defecte

Când apare un *eveniment listă* la una din intrările sale, chiar dacă nu a apărut nici un eveniment logic la intrări, simulatorul trebuie să recalculeze o listă de defecte asociată ieşirii unui element.

Ilustrăm procedura pentru calculul listelor de defecte în cazul porţilor logice. Presupunem o poartă NOR având intrările A = 0, B = 0, C = 1, şi D = 1. Presupunem că intrările A, B, C, şi D au listele de defecte din Fig.1. 34.

Acestea sunt listele curente, asociate cu elemente logice ce alimentează aceste intrări. Datorită efectului căilor reconvergente din circuit, unele defecte apar în mai multe liste de intrare. Listele de defecte din Fig.1. 34, sunt:

$L_A = \{a, e\}$
$L_B = \{b, c\}$
$L_C = \{a, b, c, d\}$
$L_D = \{a, d, f\}$

Considerăm defectul a din Fig.1. 34.. Intrucât defectul a apărut în L_A, L_C, L_D, el cauzează complementarea stărilor lui A, C, şi D (adică, A = 1, şi C = D = 0) şi face ieşirea E să fie 0. Aceasta implică că defectul A, nu cauzează eroare pe linia E, şi deci nu aparţine lui L_E.

Pe urmă, considerăm defectul d. Intrucât defectul d apare atât în L_C, cât şi în L_D, obţinem C = D = 0, şi deci E = 1. Prin urmare, defectul d, cauzează eroare pe linia E, şi deci este conţinut în L_E. Defectele ce se propagă spre E, sunt cele ce cauzează complementarea ieşirii bune a lui E.
Deci, pentru a cauza 1 pe E, un defect trebuie să realizeze toate valorile de intrare 0 la intrarea porţii cu ieşirea E, şi în acelaşi timp nu trebuie să cauzeze schimbarea în 1 a nici unei intrări pentru poarta cu ieşirea E.
In acest exemplu, intrările fără defect sunt A = B = 0 şi C = D = 1.
Deci, avem:

$$L_E = \overline{(L_A \bigcup L_B)} \bigcap L_C \bigcap L_D$$

BUPT

Prin calculul de mai sus, utilizând operaţii cu mulţimi, se poate calcula pornind de la defectele ce se propagă spre intrările unui element logic, lista defectelor ce se propagă spre ieşirea lui. In plus, trebuie luate în consideraţie defectele interne asociate elementului logic, ce produc ieşire incorectă pentru intrări logice corecte ale elementului în analiză.

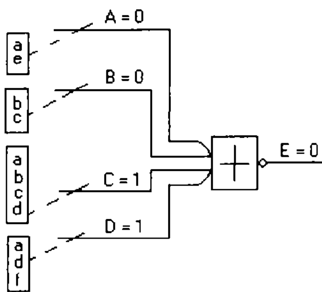


Fig.1. 34

In Fig.1. 34, sub presupunerea defectelor singulare s-a-0 (1), avem defectul E s-a-1. Bazat pe acest exmplu, se completează lista defectelor asociate cu E, după cum urmează:

$$L_E = \left(\left(L_A \bigcup L_B\right) \bigcap L_C \bigcap L_D\right) \bigcup \{E/1\} = \{d, E/1\}$$

Pentru aceeaşi poartă NOR cu 4 intrări, considerăm stimulul A = B = C = D = 0, având deci ieşirea corectă E =1. Tinând cont că, orice defect ce cauzează 1 la orice linie de intrare, sau 0 pe linia de ieşire, va produce ieşire logică incorectă, obţinem:

$$L_E = L_A \bigcup L_B \bigcup L_C \bigcup L_D \{E/0\} = \{A/1, B/1, C/1, D/1, E/0\}$$

Procedurile de calcul a defectelor pentru alte tipuri de porţi logice sunt şi ele directe, şi implică consideraţii de genul celor de mai sus.

Pâna acum am considerat simularea defectelor la nivelul porţii; când dimensiunea circuitului este foarte mare (cazul LSI, sau VLSI), este mai costisitoare simularea la nivel de poartă decât simularea la nivel funcţional a unităţilor de memorie, a registrelor, a decodoarelor, etc..

In *simularea funcţională*, se specifică doar comportarea intrării/ieşirii pentru fiecare bloc funcţional, a cărui realizare la nivel de poartă este necunoscută. Această metodă, face realizabilă simularea circuitelor mari, întrucât blocurile mari funcţionale, au cerinţe mai mici pentru resursele calculatorului gazdă, comparativ cu cele reclamate de simularea la nivel de poartă.

La implementarea simulatorului funcţional al sistemului LAMP, a fost dezvoltată o asemenea metodă de propagare a efectelor defectelor prin blocurile funcţionale printr-o simulare deductivă; metoda a fost dezvoltată de Menon şi Chapell.

1.5.3.2.4 Simularea concurentă a defectelor

Această metodă, dezvoltată de Ulrich & Baker în 1973, cunoscută sub denumirea de *Concurent Fault Simulation* CFS, realizează o singură trecere de simulare pentru calculul tuturor defectelor detectabile la fiecare test aplicat.

Uzual, comportarea unui circuit defect este numai puţin diferită decât cea pentru circuitul fără defect. Activităţile în circuitul defect, şi în cel funcţional corect, sunt uneori identice, majoritatea timpului aproape identice, şi numai rareori substanţial diferite.

Bazată pe această similitudine între circuitul defect şi cel fără defect, CFS constă în simularea circuitului fără defect, şi simularea concurentă a circuitului cu defect, numai când activitatea celui cu defect diferă de cea a circuitului fără defect.

Există anumite asemănări între simularea deductivă şi cea concurentă. Diferenţa dintre

BUPT

ele va fi clarificată după ilustrarea conceptului şi a operaţiilor fundamentale ale CFS.

Fiecare intrare în lista de defecte, este dată de numele sau indexul fiecărui defect ce produce o eroare (pe o intrare, sau o ieşire a unei porţi), şi de valorile sale de intrare şi de ieşire.

Pentru a facilita procesarea rapidă, intrările sunt sortate prin indexul defectului, şi sunt memorate în ordinea indicilor lor, într-o listă de structură.

In DFS, o listă de defecte asociată cu linia D, L_D, conţine numai indicele, sau numele defectelor ce produc o eroare pe linia D. Deci, *lista de defecte deductivă* este un subset al *listei de defecte concurentă*, prin urmare, CFS reclamă mult mai mult spaţiu de memorie comparativ cu DFS.

In CFS, se simulează explicit atât circuitul fără defect, cât şi circuitele defecte, pe când în DFS, numai circuitul normal este simulat explicit, cele defecte fiind simulate deductiv. O altă diferenţă semnificativă [LEVE80] între DFS şi CFS, este legată de faptul că un simulator concurent procesează numai circuitele active.

In simularea deductivă, datorită procesului complex al operaţiilor cu mulţimi, este necesar să se reevalueze toate defectele din listă. In contrast, în simularea concurentă, întrucât fiecare defect dintr-o listă este procesat separat, este posibilă procesarea rapidă prin tehnica *table lookup*.

Dezavantaj pentru CFS este faptul că necesită mai mult spaţiu de memorie comparativ cu DFS, şi ca şi-n cazul DFS, nu se poate prevede înainte de rulare cantitatea de memorie necesară. Acest aspect poate deveni o problemă serioasă în cazul circuitelor mari de talia VLSI, fiind astfel necesară alocarea dinamică a memoriei.

1.5.3.2.5 Simularea defectelor de întârziere

Datorită cerinţelor de calitate mereu crescânde ale chip-urilor VLSI şi datorită faptului că timpii statistici se consideră ca şi bază de proiectare a circuitelor de mare viteză, testarea întârzierilor a căpătat o mare importanţă teoretică şi practică [POME95B].
Scopul global al testării întârzierilor este să garanteze că întârzierile de propagare pe toate căile unui circuit dat sunt mai mici decât perioada clock-ului sistem.

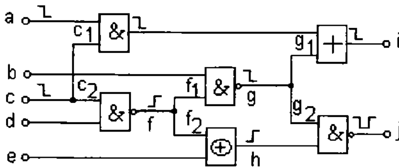


Fig.1. 35

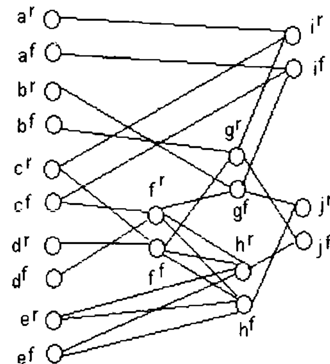


Fig.1. 36

In continuare se prezintă o metodă de simulare accelerată a defectelor pentru defectele de întârziere de pe cale. Ea se bazează pe utilizarea unei logici cu 6 valori. Pentru a putea simula circuitele VLSI complexe cu secvenţe de *pattern*-uri lungi şi în timp CPU rezonabil procedura aplică procesarea paralelă a *pattern*-urilor la toate

BUPT

stadiile procedurii de calcul.

De multe ori, multe căi ce nu pot fi testate sub condiția restrictivă de robustețe, pot fi testate dacă restrângem această condiție.
Acesta este motivul modificării logicii de 6 variabile și a derivării unei logici bazată pe 4 valori logice, ce să permită să se țină cont de detecția nerobustă a defectelor de întârziere pe căile de propagare.
Fig.1. 35 și Fig.1. 36 arată cum poate fi descrisă structura unui circuit combinațional C printr-un graf orientat $G_p = (V_p, E_p)$, numit graful căilor.

1.5.3.2.5.1 Detecția robustă și nerobustă a căilor ce prezintă defecte de întârziere

Def.1.1. Fie $P=(k_1^{t1}, ..., k_n^{tn})$ o cale în calea ilustrată în Fig.1. 35, unde k_1 and k_n corepund unei PI și unei PO a circuitului combinațional, respectiv $t_1, ..., t_n \in \{r, f\}$. Fie APD(P) întârzierea de propagare a tranziției t_1 de-a lungul lui P.
Atunci, se spune că P are o cale defectă (*path fault*), dacă APD(P) depășește T_C ($APD(P) > T_C$).

Def.1.2. Fie $<V_1, V_2>$ două *pattern*-uri de test aplicate logicii combinaționale din mediul hardware arătat în Fig.1. 24. La momentul T_1, când circuitul s-a stabilizat ca urmare a vectorului de inițializare V_1, toate semnalele se spune că au valoarea lor inițială.
Valoarea finală a unui anumit semnal este definită ca fiind valoarea logică binară (0 sau 1) pe care ar lua-o acel semnal sub vectorul de propagare V_2 la momentul T_2, dacă circuitul ar fi fără defecte de întârziere.

Def.1.3. Un test din două *pattern*-uri $<V_1, V_2>$ este numit test robust pentru un defect pe calea $P= (k_1^{t1}, ...,k_n^{tn})$ dacă el detectează sub presupunerea defectelor multiple toate defectele de întârziere din P:

- el provoacă tranziția t_1 la PI k_1 și
- garantează independent de alte defecte de întârziere că toate semnalele k_i (i=2, ...n) nu pot să-și atingă valoarea lor finală sub V_2 fără ca t_{i-1} dorită să fi ajuns la semnalul k_{i-1}.

Def.1.4. Două *pattern*-uri de test $<V_1, V_2>$ sunt numite test nerobust pentru un defect de întârziere pe calea $P=(k_1^{t1}, ...,k_n^{tn})$, dacă el detectează un defect de întârziere pe P sub presupunerea defectului singular:

- el provoacă tranziția t_1 la PI K_1 și
- V_2 cauzează ca toate intrările porților activate ce sunt în afara căii structurale ce corespunde lui P să ia valori finale ce nu sunt de control.

De exemplu să considerăm circuitul C arătat în Fig.1. 35. Fie vectorul de inițializare V_1=(1, 1, 1, 1, 0) și vectorul de propagare V_2=(0, 1, 0, 1, 0). Conform definiției 3, acesta este un test robust pentru defectul de pe calea $P_1=(c^f, f^r, g^f, i^f)$. Pe de altă parte, $<V_1,V_2>$, nu reprezintă un test robust pentru defectul de pe calea $P_2 = (c^f, f^r, g^f, j^r)$, dar îndeplinește condițiile de test nerobust pentru P_2 (Calea P_2 nu este testabilă sub condiția restrictivă de robustețe).

De notat că testul nerobust $<V_1,V_2>$ al căii P_2 poate fi validat ca fiind robust, de un test robust pentru calea $P_3=(c^r, f^f, h^r, j^r)$ (e.g. $<V_1',V_2'>$ = ((0,0,1,1,0), (0,0,0,1,0)).

Dacă aplicarea lui $<V_1',V_2'>$ duce la o valoare finală 0L neașteptată la PO j, calea P_3 este fără defect ($APD(P)<T_C$). În acest caz, tranziția de cădere la PO j ajunge

BUPT

înaintea lui T_C, astfel răspunsul de test nu poate fi invalidat de un defect de întârziere pe calea P_3.
Totuşi, chiar şi în cazul când un test nerobust nu este validabil de un altul robust, detecţia nerobustă prezintă avantajul că oferă anumite informaţii despre defect.

1.5.3.2.5.2 Simularea robustă şi nerobustă a defectelor de întârziere de pe căile de propagare a semnalelor

Pentru realizarea simulării robuste a căilor ce prezintă defecte de întârziere se face uz de 6 valori (propuse de [POPE96]) , şi anume: os, op, o-, 1s, 1p, 1-. Fiecare din aceste valori poate fi văzută ca şi o pereche ordonată (fv, pds). Dacă se face referire la un anumit semnal k al circuitului, prima fv(k) $\in \{0,1\}$ reprezintă valoarea finală a semnalului k sub cele două *pattern*-uri de test <V_1,V_2>. A doua componentă pds(k) $\in \{s,p,-\}$, numită starea de întârziere a căii sub semnalul k, este definită după cum urmează:

Def.1.5. Un anumit semnal k al unui circuit combinaţional C are pds(k)=s, dacă şi numai dacă este garantat că el rămâne stabil la fv(k) între T_1 şi T_2 (el nu prezintă tranziţii sau hazarduri T_1 şi T_2).
Altfel, pds(k)=p, dacă şi numai dacă, există cel puţin o cale de la o PI la un semnal k a cărei defect de întârziere pe cale este detectabil robust la nivelul semnalului k (în conformitate cu Def.1.3).
În final, dacă if pds(k)≠s şi pds(k)≠p, atunci pds(k)=-.
Fig.1. 37 .arată tabele de propagare a celor 6 valori logice pentru porţile AND şi XOR.

^	0s	0p	0-	1s	1p	1-
0s	0s	0s	0s	0s	0s	0s
0p	0s	0-	0-	0p	0-	0-
0-	0s	0-	0-	0-	0-	0-
1s	0s	0p	0-	1s	1p	1-
1p	0s	0-	0-	1p	1p	1p
1-	0s	0-	0-	1-	1p	1-

⊕	0s	0p	0-	1s	1p	1-
0s	0s	0p	0-	1s	1p	1-
0p	0p	0-	0-	1p	1-	1-
0-	0-	0-	0-	1-	1-	1-
1s	1s	1p	1-	0s	0p	0-
1p	1p	1-	1-	0p	0-	0-
1-	1-	1-	1-	0-	0-	0-

Fig.1. 37

Pentru ilustrare vom considera o poartă AND cu cele două semnale de intrare k_1 şi k_2 şi semnalul de ieşire q. De observat că (k_1,k_2)=(0p,0p), (k_1,k_2)=(0p,1-) şi (k_1,k_2)=(0p,1p) duce la q=0-, ceea ce indică că defectele pe căi ce sunt detectabile robust la k_1, nu îndeplinesc condiţia de detecţie robustă la q. Aceasta se datorază faptului că tranziţia căzătoare, hazardul posibil sau o întârziere pe tranziţia crescătoare a lui k_2 poate cauza valoarea finală 0L a ieşirii q a porţii, cu toate că tranziţia lui k_1 nu a ajuns încă. Aceasta contrazice Def.1.3. şi, ca şi consecinţă, <V_1,V_2> nu poate fi test robust pentru nici o cale spre semnalul q. Pe de altă parte, (k_1,k_2)=(1p,1p), (k_1,k_2)=(1p,1-) şi (k_1,k_2)=(1p,1s) duce la q=1p, indicând că defectele de pe căi detectabile robust la k_1 sunt detectabile şi la q. Ultima afirmaţie este datorată faptului că ieşirea q nu poate lua valoarea ei finală 1L fără ca tranziţia crescătoare să fi ajuns la k_1 (Def.1.3.).

In continuare urmă să considerăm cazul simulării nerobuste a defectelor de întârziere de pe căile de propagare. În mod cert toate combinaţiile de valori de intrare ale porţii AND menţionate, ce duc la pds(q)=p în simularea robustă a defectelor de întârziere de pe căi, vor duce şi în simularea nerobustă a defectelor de întârziere de pe căi la acelaşi pds(q)=p.

BUPT

Întrcât în detecția nerobustă a defectelor de întârziere de pe căi sunt decisive doar valorile finale ale intrărilor din afara căii ale porților activate, cele două combinații de intrare: $(k_1,k_2)=(0p,1-)$ și $(k_1,k_2)=(0p,1p)$ permit și detecția nerobustă a defectelor pe căile de la k_1 la ieșirea q a porții, ceea ce va fi luat în considerare prin q=0p. Astfel, pentru realizarea simulării nerobuste a defectelor de întârziere de pe căi, tabelele de propagare pentru poarta AND trebuiesc modificate așa cum se arată în Fig.1. 38:

^	0s	0p	0-	1s	1p	1-
0s	0s	0s	0s	0s	0s	0s
0p	0s	0-	0-	0p	0p	0p
0-	0s	0-	0-	0-	0-	0-
1s	0s	0p	0-	1s	1p	1-
1p	0s	0p	0-	1p	1p	1p
1-	0s	0p	0-	1-	1p	1-

(a)

^	0p	$0\overline{p}$	1p	$1\overline{p}$
0p	$0\overline{p}$	$0\overline{p}$	0p	0p
$0\overline{p}$	$0\overline{p}$	$0\overline{p}$	$0\overline{p}$	$0\overline{p}$
1p	0p	$0\overline{p}$	1p	1p
$1\overline{p}$	0p	$0\overline{p}$	1p	$1\overline{p}$

(b)

Fig.1. 38

Acum, dacă reamintim că nici pds(q)=s, nici pds(q)=- nu pot fi utilizate pentru detecția unui defect de întârziere, considerarea tablourilor de propagare cu 6 valori logice din Fig.1. 38.a., relevă faptul că cele două rânduri ce corespund lui k_1=0s și k_1=0- precum și rândurile ce corespund lui k_1=1s și k_1=1- sunt complet identice sub aspectul detectabilității defectelor de pe căi la nivelul semnalului de ieșire q rezultat.

Întrucât aceasta se menține valabil atât pentru porțile OR cât și XOR, combinăm cele două valori logice 0s și 0- (1s și 1-) la una nouă, notată prin $0\overline{p}$ ($1\overline{p}$). Ca urmare, pentru simularea nerobustă a defectelor de întârziere de pe căile de propagare a semnalelor obținem o logică cu patru valori binare.

1.5.3.2.5.3 Codificarea valorilor logice

Întrcucât valoarea finală a semnalului k, fv(k), este fie 0L fie 1L în ambele tipuri de simulări logice (cu 6 sau cu 4 variabile logice), codificarea sa este trivială și ea poate fi realizată cu un singur bit. Pe de altă parte, starea de detectabilitate a căii pds(k) are trei valori logice în simularea bazată pe logica cu 6 variabile și în consecință, trebuie să fie codificată printr-o pereche ordonată de biți:

pds(k)=(s(k), p(k)),

unde s(k) și p(k) sunt numite s-bitul și p-bitul valorii logice a semnalului k.

Valori logice ale semnalului k	fv(k)	s(k)	p(k)
0s/1s	0/1	1	0
0p/1p	0/1	0	1
0-/1-	0/1	0	0

(a)

Valori logice ale semnalului k	fv(k)	p(k)
0p/1p	0/1	1
$0\overline{p}$ /$1\overline{p}$	0/1	0

(b)

Fig.1. 39

În cazul logicii de simulare cu 4 variabile utilizată în simularea nerobustă a defectelor de întârziere de pe căile de propagare, există doar două valori posibile pentru pds(k). Astfel, este suficient p-bitul pentru codificarea lui pds(k).

Fig.1. 39.a. și Fig.1. 39.b. ilustrează codul ce a fost ales pentru reprezentarea valorilor logice ale simulării cu 6 valori logice, respectiv cu 4 valori logice.

BUPT

Pentru procesarea paralelă a perechilor de *pattern*-uri pe parcursul procesului de simulare, pentru reprezentarea setului de L valori logice ale semnalului k, se folosesc 3 cuvinte maşină de lungime L în cazul simulării robuste şi 2 astfel de cuvinte în cazul celei nerobuste (care corespund de obicei unui set de L perechi de *pattern*-uri $<V_1,V_2>_1$, ..., $<V_1,V_2>_L$):

1. Toţi biţii ce reprezintă valoarea finală a semnalului k sunt memoraţi în primul cuvânt, numit *fv-cuvânt* [fv(k)].
   vec[fv(k)] = $(fv(k)_1, \ldots, fv(k)_L)$
2. Similar,
   vec[p(k)] = $(p(k)_1, \ldots, p(k)_L)$ reprezintă *p-cuvântul*.
3. În cazul simulării robuste mai avem nevoie de:
   vec[s(k)] = $(s(k)_1, \ldots, s(k)_L)$
   ce este referit ca fiind *s-cuvânt*.

De exemplu, pentru L=5 şi făcând uz de logica cu 6 valori, setul {1p,0s,0-,1s,0p} de valori logice ale semnalului k este reprezentat de:
vec[fv(k)] = (1,0,0,1,0), vec[p(k)] = (1,0,0,0,1) şi vec[s(k)] = (0,1,0,1,0).

1.5.3.2.5.4 Simularea paralelă a căilor de întârziere

În primul rând PI ale logicii combinaţionale trebuiesc iniţializate la valorile lor corecte conform perechilor de *pattern*-uri $<V_1,V_2>_1$, ..., $<V_1,V_2>_L$.
Atât în simularea robustă cât şi în cea nerobustă a defectelor de întârzieri de pe căi, pentru perechea i, se asignează p pentru PI α dacă valoarea ei iniţială este 1(0) şi valoarea finală 0(1). Dacă nu există nici o diferenţă între valoarea iniţială şi cea finală a PI α (dacă PI α rămâne stabilă la 0L sau la 1L între T_1 şi T_2, PI α este iniţializată la 0s sau la 1s în cazul robust şi la $0\bar{p}$ sau $1\bar{p}$ în cazul nerobust).
Aceste valori logice trebuiesc propagate spre PO a logicii combinaţionale. Aceasta se poate realiza prin realizarea evaluărilor de porţi în ordinea nivelului lor, conform tabelelor de propagare corespondente, aşa cum este ilustrat în mod reprezentativ pentru porţile AND şi XOR în Fig.1. 37 şi Fig.1. 38.

Considerând o poartă AND cu semnalele de intrare k_1 şi k_2 şi semnalul de ieşire q, avem:

$$fv(q) = fv(k_1) \cdot fv(k_2),$$
$$s(q) = fv(k_1)/ \cdot s(k_1) + fv(k_2)/ \cdot s(k_2) + fv(k_1) \cdot s(k_1) \cdot fv(k_2) \cdot s(k_2), \text{ şi}$$
$$p(q) = fv(k_1) \cdot fv(k_2) \cdot [p(k_1) + p(k_2)] + fv(k_1)/ \cdot p(k_1) \cdot fv(k_2) \cdot s(k_2) + $$
$$+ fv(k_2)/ \cdot p(k_2) \cdot fv(k_1) \cdot s(k_1)$$

în cazul simulării robuste şi:

$$fv(q) = fv(k_1) \cdot fv(k_2),$$
$$p(q) = fv(k_1) \cdot p(k_2) + fv(k_2) \cdot p(k_1)$$

în cazul simulării nerobuste.

Formule similare pot fi obţinute pentru toate tipurile de porţi.

Fig.1. 40 şi Fig.1. 41 rezumă operatorii binari ce sunt necesari pentru determinarea valorii finale şi a stării de detectabilitate a căii prin procesare paralelă a perechilor de *pattern*-uri în simulare robustă şi nerobustă a defectelor de întârziere de pe căile de propagare a defectelor pentru toate tipurile de porţi. Prin aplicarea legii de asociativitate aceste operaţii pot fi extinse cu uşurinţă la porţi cu mai mult de două intrări.

BUPT

Tipul porții	Operații binare
	Simulare robustă și nerobustă a întârzierilor
AND	$vec[fv(q) = vec[fv(k_1)] \cdot vec[fv(k_2)]$
NAND	$vec[fv(q) = \{vec[fv(k_1)] \cdot vec[fv(k_2)]\}/$
OR	$vec[fv(q) = vec[fv(k_1)] + vec[fv(k_2)]$
NOR	$vec[fv(q) = \{vec[fv(k_1)] + vec[fv(k_2)]\}/$
BUF	$vec[fv(q) = vec[fv(k)]$
INV	$vec[fv(q) = \{vec[fv(k)]\}/$
XOR	$vec[fv(q) = vec[fv(k_1)] \oplus vec[fv(k_2)]$
XNOR	$vec[fv(q) = \{vec[fv(k_1)] \oplus vec[fv(k_2)]\}/$

Fig.I. 40

Tipul porții	Operații binare	
	Simulare robustă a defectelor de întârziere	Simulare nerobustă a defectelor de întârziere
AND NAND	$vec[s(q)] = vec\overline{[fv(k_1)]} \cdot vec[s(k_1)] +$ $+ vec\overline{[fv(k_2)]} \cdot vec[s(k_2)] +$ $+ vec[fv(k_1)] \cdot vec[s(k_1)] \cdot vec[fv(k_2)] \cdot vec[s(k_2)]$ $vec[p(q)] = vec[fv(k_1)] \cdot vec[fv(k_2)] \cdot$ $\cdot [vec[p(k_1)] + vec[p(k_2)]] +$ $+ vec\overline{[fv(k_1)]} \cdot vec[p(k_1)] \cdot vec[fv(k_2)] \cdot vec[s(k_2)] +$ $+ vec\overline{[fv(k_2)]} \cdot vec[p(k_2)] \cdot vec[fv(k_1)] \cdot vec[s(k_1)]$	$vec[p(q)] =$ $= vec[fv(k_1)] \cdot vec[p(k2)] +$ $+ vec[fv(k_2)] \cdot vec[p(k_1)]$
OR NOR	$vec[s(q)] = vec[fv(k_1)] \cdot vec[s(k_1)] +$ $+vec\ [fv(k_2)] \cdot vec[s(k_2)] +$ $+vec\overline{[fv(k_1)]} \cdot vec[s(k_1)] \cdot vec\overline{[fv(k_2)]} \cdot vec[s(k_2)]$, and $vec[p(q)] = vec\overline{[fv(k_1)]} \cdot vec\overline{[fv(k_2)]} \cdot$ $\cdot [vec[p(k_1)] + vec[p(k_2)]] +$ $+ vec[fv(k_1)] \cdot vec[p(k_1)] \cdot vec\overline{[fv(k_2)]} \cdot vec[s(k_2)] +$ $+ vec[fv(k_2)] \cdot vec[p(k_2)] \cdot vec\overline{[fv(k_1)]} \cdot vec[s(k_1)]$	$vec[p(q)] =$ $= vec\overline{[fv(k_1)]} \cdot vec[p(k_2)] +$ $+ vec\overline{[fv(k_2)]} \cdot vec[p(k_1)]$
BUF INV	$vec[s(q)] = vec[s(k)]$ $vec[p(q)] = vec[p(k)]$	$vec[p(q)] = vec[p(k)]$
XOR XNOR	$vec[s(q)] = vec[s(k_1)] \cdot vec[s(k_2)]$ $vec[p(q)] = vec[p(k_1)] \cdot vec[s(k_2)] +$ $+ vec[p(k_2)] \cdot vec[s(k_1)]$	$vec[p(q)] = vec[p(k_1)] +$ $+ vec[p(k_2)]$

Fig.I. 41

Aplicând procedura de simulare descrisă mai sus circuitului nostru C cu cele două *pattern*-uri de test indicate (Fig.I. 35), obținem:

(a, b, c, d, e, f, g, h, i, j) = (0p, 1s, 0p, 1s, 0s, 0-, 1p, 0p, 1p, 1p, 1-) și

(a, b, c, d, e, f, g, h, i, j) = (0p, 1p/, 0p, 1p/, 0p/, 0p/, 1p, 0p, 1p, 0p, 1p)

BUPT

1.5.3.2.5.5 Detecția căilor de întârziere

După ce s-au evaluat toate porțile circuitului combinațional pe baza operărilor binare detaliatate în Fig.1. 40 și Fig.1. 41, trebuiesc identificate toate căile cu defecte de întârziere ce sunt detectabile robust sau nerobust de cele L perechi de *pattern*-uri simulate.
De exemplu, considerăm o singură pereche de *pattern*-uri: <V_1,V_2>.
Detectarea căii cu defecte de întârziere detectate de <V_1,V_2> este trivială în simularea robustă a căilor cu defecte de întârziere. De fapt, ea poate fi realizată simplu prin trasarea într-o manieră directă, de la nivelul PO la cel al PI, a tuturor căilor structurale pe care toate semnalele au fie valoare 0p, fie 1p. (Fig.1. 42).

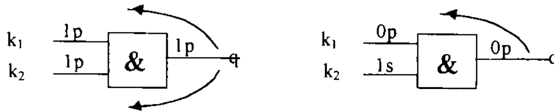


Fig.1. 42

Considerăm circuitul C din Fig.1. 35; trasarea va începe de-alungul h, g_1, g, f_1, f, c_2, și c, care reprezintă singura cale ce prezintă defect pe cale detectabil robust. Ca urmare, întrcât c=0p, vom marca calea P_1 = (c^f, f^r, g^f, h^f) ca fiind testată robust.
Identificarea căilor defecte este ceva mai complicată în cazul simulării nerobuste.
Presupunem că trasarea a ajuns la semnalul de ieșire q al unei porți g cu intrările k_1, ...,k_n. În acest caz trebuiesc determinate acele intrări ale porții k_i ce au pds(k_i)=p și ale căror valoare finală fv(k_i) este observabilă la ieșirea q a porții, adică pentru care activarea locală a căii:

$$\text{lps}(q_{ki}) = p(k_i) \cdot q_{k_i}^{fv} = 1, \text{ unde } q_{k_i}^{fv} = fv(q(k_i)) \oplus fv(q(\overline{k_i})),$$

reprezintă diferența Booleană q_{ki} cu privire la valorile finale ale semnalulu q și k_i.

Fig.1. 43 ilustrează evaluarea activării locale a căii pentru o poartă AND cu cele două valori logice 0p și 1p la semnalele sale de intrare k_i și k_j. Pe scurt, întrucât:

$$q_{k_i}^{fv} = \left(fv(k_i) \wedge fv(k_j)\right) \oplus \left(\overline{fv(k_i)} \wedge fv(k_j)\right) = (0 \wedge 1) \oplus (1 \wedge 1) = 1$$

și p(k_i)=1, activarea locală a căii pentru semnalul k_i este egală cu 1L, adică lps(q_{ki})=1.

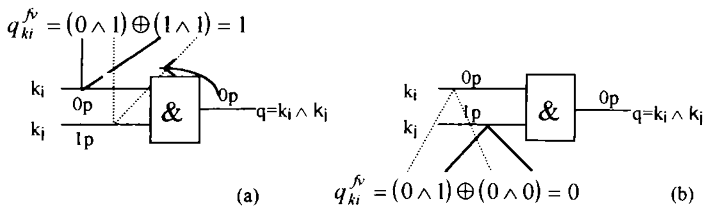


Fig.1. 43

Astfel, trasarea căilor va începe de la ieșirea q a porții peste intrarea k_i, spre PI a circuitului (Fig.1. 43.a.).

Pe de altă parte, considerând cealaltă intrare a porții, k_j:

$$q_{k_j}^{fv} = (0 \wedge 1) \oplus (0 \wedge 0) = 0$$

și lps(q_{kj})=0. Aceasta indică că nici un defect a nici unei căi de la PI la semnalul k_j nu

BUPT

poate fi observat la ieşire de semnal q; trasarea căilor nu trebuie continuată peste intrarea de poartă k_j (Fig.1. 43.b).

De exemplu, la circuitul din Fig.1. 35., pentru poarta NAND cu semnalul de ieşire j, obţinem $p(g_2)=1$, $j_{g2}^{fv}=1$, $p(h)=1$, şi $j_h^{fv}=0$.Astfel, în simularea nerobustă, pe lângă calea structurală ce corespunde lui P_1 avem de trasat de-a lungul lui j, g_2, g, f_1, f, c_2 şi c; şi în consecinţă, trebuie marcat defectul pe calea $P_2=(c^f, f^r, g^f, j^r)$ ca fiind nerobust detectabil.
Ca şi consecinţă a faptului că $j_h^{fv}=0$, trasarea nu trebuie să înceapă de-a lungul lui j, h, f_2, f, c_2 şi c, întrcât defectul de cale în $P_3=(c^f, f^r, h^r, j^f)$ nu este detectabil nici robust, nici nerobust de către perechea de *pattern*-uri indicată în Fig.1. 35.

Pentru a se putea realiza şi procesarea paralelă a *pattern*-urilor pe parcursul trasării căilor, se introduce vec[po(k)] ca şi mască de observabilitate a căii pentru semnalul k.
Iniţializând trasarea căilor la o anumită PO q prin:

$$vec[po(q)] = vec[p(q)],$$

se procedează recursiv în maniera *depth first* menţionată de la evaluarea unei intrări de poartă k

$$vec[po(k)] = vec[po(q)] \cdot vec[lps(qk)],$$

unde vec[lps(qk)] este determinat prin procesare paralelă a *pattern*-urilor conform:

$$vec[lps(qk)]=vec[p(k)] \cdot vec[q_k^{fv}].$$

Astfel, când trasarea a ajuns la o PI să zicem α, vec[po(α)] indică toate perechile din cele L perechi simulate ce pot fi utilizate pentru detectarea căilor defecte pe căile structurale trasate de la PO q la PI α.

1.5.4 Modalităţi de evaluare a răspunsurilor

Generarea pe parcursul testării a răspunsurilor de test aşteptate se poate face:

- fie prin utilizarea unei copii bune a unităţii ce se testează (*gold unit*),
- fie prin utilizarea emulării în timp real a unităţii testate. Acest tip de testare este numită *comparison testing* (denumire improprie întrucât compararea datelor de răspuns este inerentă în multe alte metode de testare).

Metodele bazate pe verificarea unei anumite funcţii *f(R)*, derivate din răspunsul *R* al unităţii ce se testează, se spune că realizează testare compactă; iar *f(R)* se spune că este o reprezentare comprimată, sau *semnătură* a lui *R*.
De exemplu, se pot număra numărul de valori 1L (sau tranziţii $0\rightarrow1$, respectiv $1\rightarrow0$) obţinute la ieşirea unui circuit şi să se compare numărul obţinut cu valoarea ce s-ar obţine în cazul circuitului fără defecte. O asemenea procedură de test simplifică procesul de testare, întrucât compararea bit cu bit a răspunsului unităţii testate cu a răspunsului aşteptat este înlocuită cu compararea doar a semnăturilor. De asemenea, întrucât nu mai este necesară memorarea întregului răspuns, se reduce semnificativ necesarul de memorie al testorului.
Tehnicile de compresie sunt utilizate în special în cazul circuitelor *self-testing*, unde calculul lui *f(R)* este realizat de un hardware special adăugat circuitului. Aceste circuite *self-testing* au şi un hardware adiţional ce realizează generarea stimulilor. În [FUJI90] şi [ABRA96] sunt prezentate tehnici de proiectare pentru circuitele cu capabilităţi încorporate de autotestare (*Built-In Self-Test* - BIST).
Dacă unitatea ce se testează se comportă incorect, ea fiind recuperabilă şi trebuind

BUPT

reparată, atunci trebuie diagnozată cauza erorii detectate. Termenii de diagnoză şi reparare pot fi utilizaţi atât în cazul căderilor fizice, cât şi în cel al erorilor de proiectare când repararea semnifică de fapt reproiectarea unităţii.
Pentru diagnoza defectelor sunt disponibile două tipuri de metode:

- *analiza cauză-efect*, ce enumeră toate defectele posibile să existe într-un model de defectare şi determină - înaintea experimentului de testare - toate răspunsurile ce le corespund pentru un anumit test aplicat. Acest proces ce se bazează pe simularea defectelor construieşte o bază de date numită *dicţionar al defectelor*. Astfel, diagnoza este un proces de căutare în dicţionarul de defecte a unui răspuns pre-calculat ce corespunde celui obţinut de la unitatea ce se testează. Dacă căutarea se încheie cu succes, dicţionarul de defecte indică defectele posibile, sau componentele posibile a fi defecte la unitatea ce se testează.
- *analiza efect-cauză* se regăseşte în cazul tehnicii *guided-probe*. O astfel de analiză procesează răspunsul (efectul) actual al unităţii ce se testează şi încearcă să determine direct doar defectele (cauza) ce ar determina un asemenea răspuns [HOHA82] [CĂTU85].

Pe de altă parte, tehnicile de diagnoză pot fi diferenţiate funcţie de nivelul la care se desfăşoară aceasta, în:

- diagnoză la nivel logic
- diagnoză la nivel de sistem.

1.6 Concluzii

Analiza funcţională a structurilor de însumare făcută în cadrul capitolului reflectă particularitatea funcţională a acestora de propagare pe două direcţii a semnalelor în cadrul sumatoarelor. Această particularitate funcţională ridică probleme în testarea, respectiv generarea stimulilor de test necesari procesului de testare, întrucât un defect apărut la nivelul lanţului de propagare şi/sau generare a semnalului CY va afecta totdeauna bitul sumă ce include semnalul respectiv, dar poate afecta şi valorile biţilor sumă mai semnificative în raport cu valoarea bitului respectiv şi eventual valoarea transportului generat prin însumarea celor doi operanzi.
Pornind de la aceste considerente, ce plasează structurile de însumare într-o subclasă specială a circuitelor combinaţionale, şi ţinând cont de importanţa la nivelul sistemelor de calcul a funcţiei implementate de aceste structuri de însumare în contextul dinamicii tehnologice la care asistăm, se justifică din plin eforturile concentrate în direcţia creşterii performabilităţii acestora în sensul creşterii performanţei operaţionale a structurilor de însumare în conjucţie cu performanţele lor de testabilitate.
Astfel, în cadrul dizertaţiei problema creşterii performanţelor de testabilitate a structurilor de însumare va fi abordată în două direcţii:

- se va identifica un algoritm de generare a stimulilor pentru testarea structurilor de însumare, care ţinând cont de analiza lor funcţională va permite obţinerea unui set minim de teste pentru detecţie;
- se vor da soluţii de reconfigurare a structurilor de însumare ce să permită autotestarea lor. Soluţionările de reconfigurare vor viza pe de o parte testarea *off-line* a structurilor de însumare, pe de altă parte testarea lor *on-line*.

BUPT

2. Eficientizarea testării structurilor de însumare

2.1 Analiza de performanţă la propagarea serială a transportului

2.1.1 Analiza de performanţă la sumatorul RC A

Cu toate că logica implicată în sinteza acestor sumatoare este de n ori mai mare decât cea implicată în implementarea sumatoarelor seriale, performanţa RCA nu este mult mai bună decât cea a sumatorului serial. Aceasta întrucât cele n FA nu pot opera întotdeauna în paralel. În general FA de rangul i aşteaptă *CY-out* al FA de rang $i-1$, s.a.m.d.

Condiderând τ ca fiind întârzierea pe o poartă, atunci întârzierea ce apare la nivelul FA este 3τ şi calea de propagare a lui CY implică o întârziere de 2τ pentru fiecare FA, deci timpul de însumare a 2 numere de n cifre binare este dat de următoarele:

- 1τ pentru formarea CY-urilor parţiale $A_i \cdot B_i$ şi a sumelor parţiale $A_i \oplus B_i$
- $2 \cdot (n-2)\tau$ pentru propagarea transportului prin rangurile 1 ... n-2
- 2τ pentru asimilarea la nivelului ultimului rang a sumei parţiale şi a lui CY parţial cu *CY-in* pentru a se forma C_{n-1} şi S_{n-1}.

Se obţine un timp total de operare egal cu:

$$T_{op\,RCA} = (2 \cdot n - 1) \cdot \tau . \qquad (2.1)$$

Cu toate acestea, odată cu dezvoltarea circuitelor VLSI, popularitatea acestor sumatoare a crescut. Aceasta se datorează faptului că:

- Costul de realizare tehnologică a sumatorului este mic, el având o structură regulată la care sunt dominante porţile şi nu conexiunile.
- Pe baza utilizării unei tehnici tehnologice de înlocuire a porţilor logice cu comutatoare se poate îmbunătăţi performanţa de viteză a acestor sumatoare prin creşterea vitezei de propagare pe rând. Calea rezultată pentru CY este cunoscută sub denumirea de *Manchester carry-chain*, iar sumatorul obţinut în acest fel ca fiind *Manchester Adder.*

În evaluările anterioare a fost folosit timpul de operare calculat pentru cazul cel mai defavorabil. În realitate, anumite analize şi experimente au arătat că media distanţei maxime de propagare a lui CY este $\log_2 n$.

Ne propunem să realizăm în continuare o analiză a propagării lui CY. În acest sens, presupunem că adunăm două numere A şi B de n biţi.

În general, se vor genera mai multe CY ce vor avea diferite distanţe de propagare de-a lungul căii de propagare a lui CY.

Fie: l cea mai mare din aceste distanţe,

$P_n(k)$, probabilitatea ca $l \geq k$,

$p_n(k)$, probabilitatea ca $l = k$.

Atunci lungimea medie, $\bar{l}$, a distanţelor de propagare cele mai lungi este dată de:

$$\bar{l} = \sum_{k=0}^{n} k \cdot p_n(k) = \sum_{k=0}^{n} k \cdot [P_n(k) - P_n(k+1)] = \sum_{k=0}^{n} P_n(k)$$

Un CY de lungime cel puţin k apare fie undeva în primele $n-1$ ranguri, cu probabilitatea $P_{n-1}(k)$, fie într-o secvenţă de ranguri ce se termină cu rangul n.

Pentru ultimul caz rangurile implicate pot fi împărţite în 3 grupe distincte:

BUPT

- $n-k$ ranguri în care nu este CY de lungime $> k-1$. Acest caz apare cu probabilitate $1-P_{n-k}(k)$.
- Rangul ce generează CY, şi care pentru generarea lui CY trebuie să aibă $A_i = B_i = 1$. Acest caz apare cu probabilitatea $\frac{1}{4}$.
- $k-1$ ranguri ce propagă CY generat în rangul anterior. Fiecare din aceste ranguri are $A_i = 1 si B_i = 0$, sau $A_i = 0 si B_i = 1$. Aceasta se întâmplă cu probabilitatea $\frac{1}{2}$ pentru fiecare rang, şi deci cu probabilitatea $\left(\frac{1}{2}\right)^{k-1}$ pentru toate cele $k-1$ ranguri.

$$\Rightarrow P_n(k) = P_{n-1}(k) + \frac{1}{4}\cdot\left(\frac{1}{2}\right)^{k-1}\cdot\left[1-P_{n-k}(k)\right] = P_{n-1}(k) + \frac{1-P_{n-k}(k)}{2^{k+1}}$$

$$\Rightarrow P_n(k) - P_{n-1}(k) = \frac{1-P_{n-k}(k)}{2^{k+1}} \quad dar\, 0 \le P_{n-k}(k) \le 1$$

$$\Rightarrow P_n(k) - P_{n-k}(k) \le \frac{1}{2^{k+1}} \quad \forall n \ge k$$

$$\Rightarrow P_n(k) = \sum_{i=k}^{n}\left[P_i(k) - P_{i-1}(k)\right] \le \frac{n-k-1}{2^{k+1}}$$

Întrucât $0 \le P_n(k) \le 1$, avem:

$$Pn(k) \le \min\left\{1, \frac{n-k+1}{2^{k+1}}\right\}.$$

Pentru a determina o limită superioară pentru $\bar{l}$, fie N o valoare astfel încât $2^N \le n \le 2^{N+1}$ $si \frac{(n-k+1)}{2^{k+1}} < 1 pt. \forall k < N.$

Atunci:

$$\bar{l} = \sum_{k=1}^{n} Pn(k) = \sum_{k=1}^{N-1} Pn(k) + \sum_{k=N}^{n} Pn(k) \le \sum_{k=1}^{N-1} 1 + \sum_{k=N}^{n} \frac{n}{2^{k+1}} = N - 1 + \frac{n}{2^N}.$$

Deci, $\overline{l^\bullet} = N$ la $n = 2^N$ şi $\overline{l^\bullet} = N+1$ la $n = 2^{N+1}$.

Adică $\overline{l^\bullet} = \log_2 n$ la ambele capete ale intervalului $\left[2^N, 2^{N+1}\right]$.

Rezultă că $\overline{l^\bullet} \le \log_2 n$ oriunde în intervalul dat şi prin urmare $\bar{l} = \log_2 n$.

Rezultatele experimentale ce validează aproximarea $\log_2 n$ se găsesc în [HEND61] şi ele sugerează valoarea $\log_2(1{,}25 \cdot n)$ ca fiind o mai bună estimare.

Nu se face o discuţie detailată a performanţelor sumatorului de tip Manchester, întrucât, spre deosebire de situaţia celorlalte tipuri de sumatoare, mijloacele de obţinere a unor performanţe superioare celor ale sumatorului RCA sunt pur tehnologice [WAST93] (şi ele nu interesează în contextul dizertaţiei). Pentru o analiză mai detaliată se poate consulta [MEAD80] şi [FENW87].

2.1.2 Analiza de performanţă la sumatorul CCA

Întrucât există 5 niveluri de logică şi calea de propagare a lui CY la nivelul fiecărui CFA implică 2 τ, timpul de operare al CFA este dat după cum urmează (în toate cele 3 cazuri s-au inclus două niveluri de întârziere pentru iniţializare) [KORE89]:

BUPT

- Cazul cel mai favorabil: $7 \cdot \tau$

Apare când nu există CY-uri şi timpul de operare este determinat de întârzierea de la punctul de intrare până la ieşirea porţii ce formează *CY complete*.

- Media cazurilor: $(2 \cdot \log_2 n + 4) \cdot \tau$

Calea de propagare a lui CY este prin $\log_2 n$ ranguri, cu o întârziere de 2τ pentru fiecare rang. Logica de formare a lui *CY complete* mai adaugă 2τ.

- Cazul cel mai defavorabil: $(2 \cdot n + 4) \cdot \tau$

Calea de propagare a lui CY este peste n ranguri cu o întârziere de 2τ / rang la care se adaugă 2τ pentru logica lui *CY complete*.

Pentru comparare cu alte sumatoare se va considera timpul mediei cazurilor ca şi timp operaţional la acest sumator.

$$T_{OPCCA} = (2 \cdot \log_2 n + 4) \cdot \tau \qquad (2.2)$$

Problema esenţială în proiectarea unui sumator CCA este determinarea lungimii căii critice (deci a timpului de adunare) în acord cu cazul cel mai defevorabil de propagare a lui CY pentru fiecare pereche de doi operanzi. În acest sens există următoarele limitări ale proiectării sumatoarelor CCA:

- Fan-in-ul mare al porţii ce generează *CY complete*
- Pentru sumatoare mari este puţin probabil să fie suficientă o singură poartă, recurgându-se astfel la o structură de sumator multinivel. Ca şi alternativă pentru aceasta, tehnica *CY completion* poate fi combinată cu o formă limitată de *CY lookahead*.
- Operarea asincronă a sumatorului poate complica proiectul unităţii ALU, sau a CPU-ului în ansamblu; întrcât fiecare din acestea sunt proiectate în general pentru operări sincrone. În felul acesta, timpul pierdut pentru resincronizare poate depăşi orice câştig al utilizării acestei metode. Dar, odată cu WSI a apărut o creştere a interesului pentru calculatoare asincrone, şi este posibil ca aceste sumatoare să devină mai practice în viitor.

2.2 Analiza de performanţă la propagarea anticipată a transportului

2.2.1 Analiza de performanţă la sumatoarele CLA, RCLA, BCLA

Analiza de performanţă la sumatorul CLA

Timpul de operare a unui asemenea sumator este 4τ, el nu depinde de dimesiunea operanzilor, fiind compus din următoarele elemente:

- τ pentru obţinerea semnalor P, G şi cele ale sumelor parţiale
- 2τ pentru formarea semnalelor C
- τ pentru asimilarea transporturilor şi a sumelor parţiale

Problema principală este dată de limitările ce apar datorită cerinţelor de fan-in şi fan-out ridicat.

Pe baza ecuaţiilor, (1.5) şi (1.6), rezultă că pentru un sumator de n cifre binare, fiecare semnal G_i este dus la $(n-i)$ intrări, şi fiecare semnal P_i este dus la $(i+1) \cdot (n-i)$ intrări - pentru semnalele G avem un maxim de valoare n, iar pentru semnalele P, un maxim de valoare $\frac{(n+1)^2}{4}$. Deci, pentru valori mari pentru n, apar probleme serioase de fan-

BUPT

out la aceste sumatoare. Legat de problema fan-in-ului, specificăm faptul că fiecare semnal C_i implică o poartă OR cu $(i+2)$ intrări şi porţi AND cu fan-in în intervalul: $[2,i+2]$- adică un maxim de valoare $n+1$ pentru ambele situaţii.
Cerinţele de fan-out şi fan-in ridicat, pot duce la scăderea performanţei, creşterea costului, sau chiar la obţinerea unor proiecte de sumatoare nerealizabile practic.

În concluzie, sumatoarele CLA în formă pură, nu sunt potrivite pentru adunarea numerelor de dimensiuni mari.

Analiza de performanţă la sumatorul RCLA

Sumatorul RCLA poate fi obţinut dintr-un sumator RCA prin înlocuirea fiecărui FA cu un sumator CLA de m cifre binare. Deci, propagarea transportului prin CLA va juca şi la acest sumator un rol important în stabilirea timpului de operare.
Cu toate că calea lui CY prin CLA, precum şi prin FA constă din două porţi de întârziere, asimilarea lui *CY-in* cu bitul parţial al sumei implică un τ la nivelul FA, pe când la nivelul CLA sunt implicate 3τ. Ca urmare, calea de propagare a lui CY este crucială doar pentru primele $N-1$ blocuri ale CLA. Deci, timpul de operare al sumatorului RCLA este dat de suma următorilor termeni:

- $1\ \tau$ pentru formarea semnalelor P şi G
- 2τ pentru formarea lui C_{m-1}
- $2\cdot(N-2)\tau$ pentru propagare prin blocurile $1,2,...,N-2$
- 3τ pentru formarea biţilor $S_{n-1},S_{n-2},...,S_{n-m}$ (şi a lui C_{m-1}), ceilalţi biţi sumă fiind deja formaţi

Se obţine un timp total de adunare de valoare:

$$T_{OP\,RCLA} = 2\cdot(N+1)\tau. \qquad (2.3)$$

Analiza de performanţă la sumatorul BCLA

Sumatorul BCLA poate fi văzut ca şi un sumator CLA analog sumatorului CSkA, având logica *CY-lookahead* ce serveşte acelaşi rol la nivelul CLA-ului ca şi logica *skip* la nivelul CSkA-ului (spargerea căilor de propagare mari).
Cazul cel mai defavorabil al propagării lui CY este totuşi diferit de cel de la CSkA, el fiind arătat în Fig.2. 1.

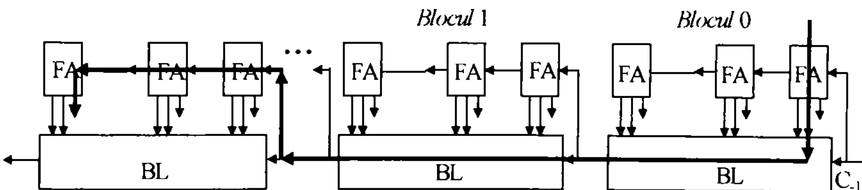


Fig.2. 1

Timpul total de adunare este:

$$T_{OP\,BCLA} = 2\cdot(m+N-1)\tau \qquad (2.4)$$

fiind compus din următoarele elemente:

- 1τ pentru a forma semnalele P şi G
- $2\cdot(N-1)\tau$ pentru propagare prin unităţile *CY-lookahead* $0,1,...,N-2$
- $2\cdot(m-1)\tau$ pentru propagare prin primele $m-1$ FA ale ultimului bloc

BUPT

- 1τ pentru formarea lui S_{n-1}

Pentru o valoare dată pentru n, dimensiunea de bloc optimă din punctul de vedere al timpului de operare este cea pentru care: $\frac{d}{dm}\left(\frac{n}{m}+m-1\right)=1-\frac{n}{m^2}=0$, adică pentru $m=\sqrt{n}$ (soluțiile practice fiind date de $\lfloor n \rfloor$, sau de $\lfloor n \rfloor+1$, sau în orice caz de o valoare întregă convenabilă apropiată de acestea). Se obține astfel un timp operațional optim de valoare: $2\cdot\left(2\sqrt{n}-1\right)\tau$.

Pentru valori ale lui m mai mici decât valoarea lui optimă, există un număr mai mare de blocuri și propagrea prin unitățile *lookahead* va afecta performanța sumatorului, iar pentru valori ale lui m mai mari decât valoarea lui optimă, numărul FA din fiecare bloc este mare, și propagarea prin acestea va afecta performanțele sumatorului.

2.2.2 Analiza de performanță la sumatoare organizate pe superblocuri

2.2.2.1 Analiza de performanță la sumatorul SBCLA

Timpul operațional al SBCLA este determinat de cazul cel mai defavorabil din Fig.2. 2.

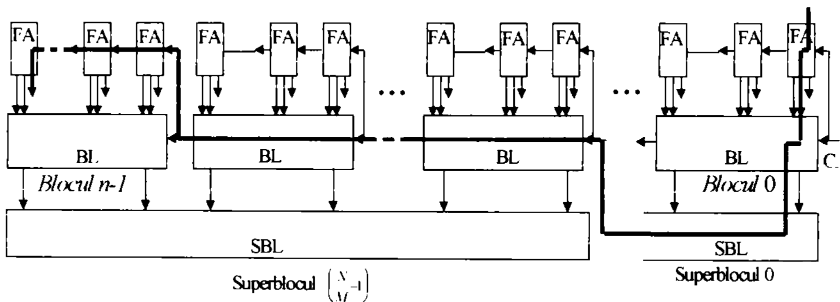


Fig.2. 2

Astfel, timpul de adunare se obține prin însumarea următorilor termeni:

- τ pentru formarea semnalelor P și G
- 2τ pentru formarea semnalelor P_i^j și G_i^j
- $2\cdot\left(\frac{N}{M}-1\right)\tau$ pentru propagare prin unitățile *lookahead* ale superblocurilor
- $2\cdot(M-1)\tau$ pentru propagare prin unitățile block-*lookahead* ale ultimului superbloc
- $2\cdot(m-1)\tau$ pentru propagare prin primele $(m-1)$ FA ale ultimului bloc
- τ pentru formarea lui S_{n-1}

Obținându-se un timp total al însumării de valoare:

$$T_{OPSBCLA}=2\cdot\left[\frac{N}{M}+M+m-1\right]\tau. \tag{2.5}$$

Pentru o dimensiune de bloc dată și pentru o dimensiune a operanzilor dată, dimensiunea optimă a superblocurilor este cea pentru care:

BUPT

$\frac{d}{dm}\left(\frac{N}{M}+M+m-1\right)=1-\frac{N}{M^2}=0$, adică $M=\sqrt{N}=\sqrt{\frac{n}{m}}$ (în practică valoarea întreagă cea mai apropiată de aceasta). Considerând această valoare optimă, se observă că pentru o dimensiune de bloc dată şi pentru o dimensiune a operanzilor ce se însumează şi ea dată, pentru $\sqrt{\frac{n}{m}}>2$, sumatorul SBCLA este mai bun decât sumatorul BCLA.

Ţinând cont şi de faptul că dimensiunea optimă de bloc la BCLA este $\sqrt{n}$, se poate concluziona că sumatorul SBCLA este mai bun, când $n>16$, m şi M fiind alese în mod potrivit. Diferenţiind relaţia de timp de mai sus în raport cu m, şi rezolvând ecuaţiile relevante, se obţine valoarea optimă teoretic pentru dimensiunea blocului şi a superblocului de valoare $\sqrt[3]{n}$. Bine-înţeles, în practică, selectarea unei valori pentru m poate schimba ecuaţia de performanţă în aşa măsură încât valoarea optimă pentru M să nu mai fie egală cu m.

2.2.2.2 Analiza de performanţă la sumatorul ISBCLA

Timpul operaţional al ISBCLA este determinat de cazul cel mai defavorabil din Fig.2. 3.

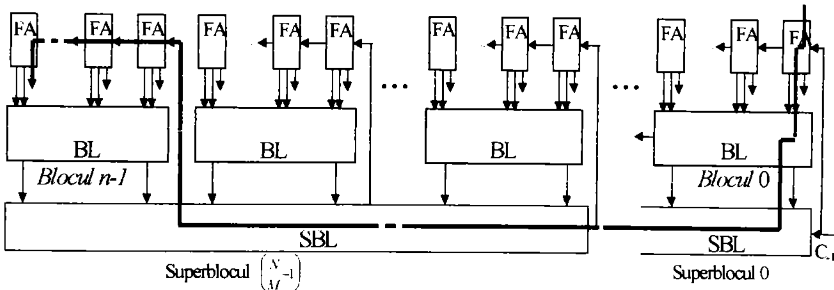


Fig.2. 3

Pe baza cazului cel mai defavorabil se determină timpul de operare ca fiind dat de suma următorilor termeni:

- τ pentru formarea semnalelor P şi G
- 2τ pentru formarea semnalelor P_i^j şi G_i^j
- $\left(2\cdot\frac{N}{M}\right)\tau$ pentru propagare prin unităţile *lookahead* ale superblocurilor
- $2\cdot(m-1)\tau$ pentru propagare prin primele $(m-1)$ FA ale ultimului bloc
- τ pentru formarea lui S_{n-1}

Rezultă un timp de operare de:

$$T_{OP\,ISBCLA}=2\cdot\left(\frac{N}{M}+m+1\right)\tau. \tag{2.6}$$

Pentru determinarea valorilor optime pentru m şi M în raport cu performanţa sistemului se ţine cont de faptul că o unitate *superblock-lookahead* reduce cu o cantitate mai mare calea de propagare a lui CY decât o face o unitate *block-lookahead*. Ceea ce înseamnă că valoarea M trebuie fixată înaintea valorii m.

BUPT

Pe baza expresiei timpului de operare ce a fost determinată, rezultă că valoarea M se ia cât mai mare posibil.
Având valoarea M fixată, pentru determinarea valorii lui m se ţine cont că aceasta trebuie să verifice relaţia: $\frac{d}{dm}\left(\frac{n}{mM}+m+1\right)=1-\frac{n}{m^2M}=0$. Se obţine o valoare egală cu $\sqrt{n/M}$. (De observat că această valoare este considerabil mai mică decât cea obţinută pentru sumatorul BCLA).
În concluzie, se alege M cât mai mare posibil şi m ca fiind valoarea întreagă cea mai apropiată de $\sqrt{n/M}$.

De exemplu, dacă, în cazul unui sumator de 64 biţi, valoarea practică maximă pentru M este 4, rezultă pentru m valoarea optimă de: $\sqrt{64/4}=4$.

2.2.2.3 Analiza de performanţă la sumatorul SRCLA

Timpul operaţional al SRCLA este determinat de cazul cel mai defavorabil din Fig.2. 4.

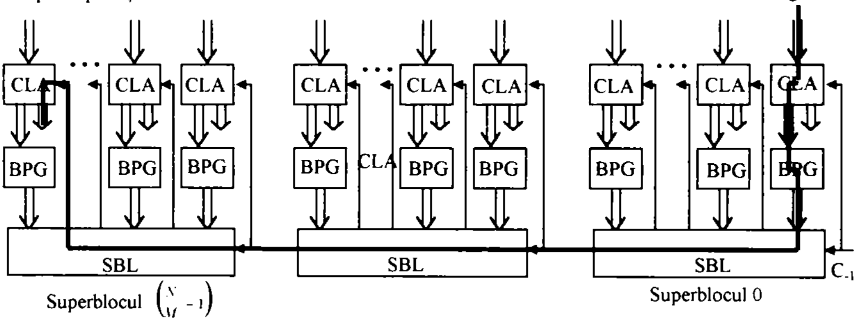


Fig.2. 4

Pe baza Fig.2. 4 se obţine pentru timpul de operare valoarea

$$T_{OP\,SRCLA}=2\left(\frac{N}{M}+3\right)\tau \qquad (2.7)$$

fiind format din următoarele elemente:

- τ pentru formarea semnalelor P şi G
- 2τ pentru formarea semnalelor P_i^j şi G_i^j
- $\left(2\cdot\frac{N}{M}\right)\tau$ pentru propagare prin superblocuri (2τ pentru fiecare superbloc)
- 2τ pentru producerea lui CY în rangul $n-1$
- τ pentru formarea lui S_{n-1}

Expresia timpului de operare arată că cele mai bune performanţe vor fi obţinute dacă se aleg atât m cât şi M cât mai mari posibile. Pe de altă parte, se observă că pentru valori date ale lui m şi M performanţele SRCLA vor fi mai bune decât cele ale ISBCLA. Întrcât valorile m şi M sunt determinate pentru ambele sumatoare pe baza aceloraşi considerente de fan-out şi de fan-in, se poate conclude că sumatoarele SRCLA sunt mai performante sub aspectul timpului de operare decât sumatoarele ISBCLA.

BUPT

2.2.3 Analiza de performanţă la sumatorul PyCLA

Biţii numerelor ce se însumează parcurg arborele de sus în jos, la bază se combină semnalele generate cu C_{-1}, şi rezultatele parcurg arborele de jos în sus generând semnalele CY şi biţii sumă calculaţi.

Un sumator PyCLA are $\log_2 n$ niveluri, şi întrucât întârzierea prin fiecare nivel este 2τ, timpul de operare a unui asemenea sumator este:

$$T_{OP\,PyCLA} = (2 \cdot \log_2 n)\tau \qquad (2.8)$$

rezultând o substanţială îmbunătăţire, în special pentru valori mari ale lui n, comparativ cu situaţia sumatorului RCA la care se face o traversare a $(2 \cdot n)$ niveluri logice. Pe de altă parte, cu toate că sumatorul CLA piramidal implică utilizarea a $(2 \cdot n)$ celule, pe când sumatorul RCA implică doar n celule, implicaţiile la nivelul spaţiului de integrare sunt de valoare $(n \cdot \log n)$ [NGAL85], [SWAR90].
În concluzie, cu un mic compromis în domeniul spaţiului de integrare [CHIE94], se obţine o substanţială creştere în viteza de operare.

Sumatorul Ling este o altă variantă de sumator cu anticipare transportului, performanţa lui înaltă fiind obţinută pe baza utilizării tehnologiei de cablare [LING81], [DORA88].

La baza proiectării structurilor de însumare CLA stă faptul că prin examinarea biţilor termenilor ce se însumează este posibilă determinarea tuturor transporturilor, eliminându-se astfel întârzierea ce ar apărea altfel în propagarea lor.

Sumatorul CLA în formă pură realizează exact acest lucru, dar el nu este o soluţie practică decât pentru realizarea sumatoarelor de dimensiune mică, întrucât pe măsură ce creşte dimensiunea sumatorului, cresc şi cerinţele de fan-in şi de fan-out ale acestuia.
O soluţie pentru problema fan-in-ului în cazul sumatoarelor mari poate fi utilizarea porţilor multinivel (sumatorul CLA piramidal), iar pentru obţinerea fan-out-ului cerut, pot fi utilizate buffer-e. Dar acestea, implica o întârziere suplimenateră (în general timpul de operare va fi mai degrabă proporţional cu $\log_2 n$ decât constant), şi astfel, în practică, performanţa obţinută nu este neapărat mai bună decât cea obţinută prin utilizarea altor tehnici.
Prin urmare, în general, sumatoarele mari utilizează limitat tehnica *lookahead* în formă pură; ele utilizează această tehnică combinată cu alte tehnici ce surmomtează dificultăţile de fan-out şi de fan-in.

La sumatoarele RCLA rangurile sumatorului sunt grupate în blocuri ce sunt organizate astfel încât tehnica CY-*lookahead* este folosită doar în interiorul blocurilor, între blocuri făcându-se propagarea serie a transporturilor.
Prin urmare, sumatorul RCLA se situează între sumatorul RCA şi sumatorul CLA în formă pură, în sensul că dacă dimensiunea blocurilor este 1, el degenerează în sumator RCA şi dacă dimensiunea blocurilor este egală cu cea a sumatorului, se ajunge la un structura sumatorului CLA în formă pură.
De unde rezultă că pentru o dimensiune de sumator dată, pe măsură ce creşte dimensiunea blocurilor sale, va creşte atât performanţa lui, cât şi dificultatea implementării sale.

Sumatorul BCLA inversează proiectul sumatorului RCLA, astfel încât CY-urile la nivelul blocurilor sunt propagate, iar cele dintre blocuri sunt determinate prin tehnica CY-*lookahead*

BUPT

Pentru ca caştigul de performanţă obţinut pe baza tehnicii *lookahead* să nu fie estompat de pierderea determinată de propagarea lui CY, se impune ca la proiectarea sumatorului BCLA dimensiunea sumatorului să fie suficient de mare, iar dimensiunea blocurilor să fie suficient de mică.

În cazul structurilor de însumare de dimensiune mare, durata propagării prin RCLA-uri sau BCLA-uri este probabil să fie destul de mare astfel încât orice performanţă câştigată să fie estompată de cantitatea mare a hardware-ului investit.
În asemenea situaţii se obişnuieşte să se utilizeze tehnica *lookahead* pe mai mult de un nivel.
Teoretic se pot utiliza orice număr de niveluri *lookahead*, dar în practică fiecare astfel de nivel implică o anumită întârziere, şi astfel, pentru o dimensiune de sumator dată, există un număr de niveluri care dacă este depăşit nu se aduce nici o îmbunătăţire în performanţă, ba mai mult, se poate ajunge la o pierdere în acest sens. Cele mai comune implementări sunt cu două niveluri de *lookahead*; bineînţeles numărul optim de niveluri pentru o implementare dată depinde de dimensiunea sumatorului, precum şi de tehnologia folosită.

SBCLA grupează rangurile sumatorului în blocuri care la rândul lor sunt grupate în superblocuri.Semnalele CY sunt produse atât la nivelul blocurilor cât şi la cel al superblocurilor, dar la toate cele 3 niveluri sunt implicate totuşi anumite propagări. Versiunea îmbunătăţită a SBCLA, numită ISBCLA elimină o parte din problemele legate de propagările serie ale transporturilor la sumatoarele de tip SBCLA.

2.3 Analiza de performanţă la propagarea cu omitere a transportului

2.3.1 Analiza de performanţă la sumatorul CSkA cu un nivel de omitere

Cazul cel mai defavorabil de propagare prin FA-re, este cazul când CY este generat în rangul cel mai puţin semnificativ al unui bloc şi se propagă spre rangul cel mai semnificativ al altui bloc, unde se opreşte. În această situaţie nu este folositoare logica CY-*skip*, şi rezultă o întârziere lungă.

Combinând această informaţie cu calea cea mai lung posibilă din logica *CY-skip*, rezultă cazul cel mai defavorabil din punct de vedere al căii de propagare, ce este ilustrat în Fig.2. 5.
Acest caz apare la adunarea perechii de operanzi: *010101 ...101* şi *001010 ...011* având $C_{-1} = 0$. Prin urmare, timpul de adunare este

$$T_{OPCSkA} = \left(2 \cdot \frac{n}{m} + 4m - 5\right) \cdot \tau \qquad (2.9)$$

el fiind compus din următoarele:

- O întârziere de o poartă *(τ)* pentru a forma T_i, suma parţială $A_i \oplus B_i$ (care pentru $i = 0$ este CY generat de rangul 0).
- În primul bloc C_0 se propagă prin $m-1$ FA cu o întârziere de $2 \cdot \tau$ / FA; se obţine o întârziere de $2 \cdot (m-1)\tau$.

BUPT

- $\left(2\cdot\left(\frac{n}{m}-2\right)+1\right)$ τ pentru ca C_{m-1} să se propage prin logica *skip* a blocurilor $1,2,\ldots,\frac{n}{m}-2$.
- În ultimul bloc C_{n-m-1} se propagă prin $m-1$ FA cu o întârziere de $2\cdot(m-1)\tau$.
- o întârziere de o poartă pentru asimilarea lui C_{n-2} şi a bitului sumă parţială pentru formarea lui S_{n-1} (ceilalţi biţi ai sumei fiind deja formaţi).

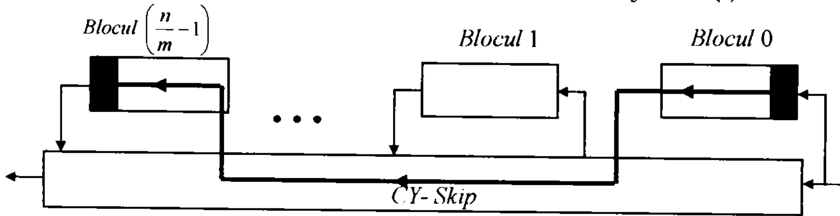


Fig.2. 5

Pe baza datelor prezentate, rezultă că este foarte importantă dimensiunea blocurilor, în sensul că pentru operanzi de o dimensiune dată, se pot proiecta mai multe astfel de sumatoare (cu diferite dimensiuni de bloc), ce prezintă aceleaşi performanţe. Aceasta sugerează o examinare mai atentă a partiţionării sumatorului.

Dacă presupunem că n este perfect divizibil cu m, şi că toate blocurile au dimensiune egală, atunci pentru un n dat dimensiunea de bloc optimă (în termeni de timpi de operare) este cea pentru care $\frac{d\left(2\cdot\frac{n}{m}+4\cdot m-5\right)}{dm}=-2\cdot\left(\frac{n}{m^2}-2\right)=0$, adică când $m=\sqrt{\frac{n}{2}}$, sau este egal cu valoarea întregă cea mai apropiată de aceasta. În practică, factori ca şi fan-in şi fan-out, ambii dependenţi de realizarea tehnologică folosită, vor juca un rol crucial în alegerea lui m pentru o implementare dată.

Dar, aşa cum se va vedea mai departe, cerinţa ca toate blocurile de însumare să aibă aceeaşi dimensiune nu va duce totdeauna la obţinerea celui mai rapid sumator CSkA.

Presupunem că se alege m astfel încât $n-m\cdot\left]\frac{n}{m}\right[\neq 0$ (adică n nu este perfect divizibil cu m ,împărţirea lor duce la obţinerea restului p). Atunci sumatorul poate fi partiţionat în $\left]\frac{n}{m}\right[+1$ blocuri, astfel încât primele $\left]\frac{n}{m}\right[$ blocuri au fiecare câte m biţi, iar ultimul are p biţi.

Calea cea mai defavorabilă este similară cu cea din cazul precedent în care se înlocuieşte blocul $\frac{n}{m}-1$ cu blocul $\left]\frac{n}{m}\right[-2$.

Deci timpul de operare al sumatorului este $\left(2\cdot\left]\frac{n}{m}\right[+4\cdot m-5\right)\cdot\tau$, fiind format din:

BUPT

- O întârziere de o poartă (τ) pentru a forma T_i, biţii sumelor parţiale $A_i \oplus B_i$ şi a biţilor CY parţiali.
- $\left(2 \cdot \left]\frac{n}{m}\right[- 3\right)$ τ pentru ca C_0 să se propage prin logica *skip*.
- $4 \cdot (m-1)$ τ pentru ca CY s să se propage prin primele $m-1$ ranguri ale blocului 0 şi prin primele $m-1$ ranguri ale blocului $\left]\frac{n}{m}\right[-2$.
- $1\,\tau$ pentru asimilarea ultimului CY şi a bitului parţial de sumă.

Pe de altă parte, dimensiunile blocurilor sunt restricţionate de cerinţele de fan-in ale CSkA; maximul atins de un fan-in fiind: $\left]\frac{n}{m}+1\right[$.

Situaţia tocmai descrisă este o particularizare a cazului mai general, şi anume: Pentru o anumită dimensiune de sumator, optarea pentru dimensiuni neuniforme de blocuri duce în general la obţinerea unor performanţe mai bune decât în situaţiile utilizării blocurilor cu dimensiune uniformă.

Se poate obţine o îmbunătăţire a performanţelor CSkA prin reducerea dimensiunilor blocurilor marginale (crescându-se dimensiunea celor interioare), reducându-se astfel timpii de propagare la nivelul acestor blocuri.

Ca şi exemplu, se consideră un sumator cu 64 cifre binare ce are 12 blocuri cu următoarele dimensiuni: $[4,4,4,4,8,8,8,8,4,4,4,4]$. Timpul de adunare al acestui sumator este $35 \cdot \tau$, care este mai bun decât cel obţinut pe baza relaţiei (2.9). Acest timp a fost obţinut pe baza formulei date în cazul blocurilor de dimensiune uniformă în care s-a înlocuit $\frac{n}{m}$ cu 12 (nr. blocuri) şi $(m-1)$ cu 3 (adică 4-1).

Media dimensiunii blocurilor acestui sumator este 5,3 care este destul de apropiată de valoarea optimă teoretică $\sqrt{32} = 5{,}7$.

Pentru a determina modul în care să se facă împărţirea în blocuri de dimensiune variabilă astfel încât timpul de operare să fie minim, vom presupune că împărţim un sumator de n biţi în K blocuri, $K \geq 2$, de dimensiuni: $m_{K-1}, m_{K-2}, \ldots, m_0$.

Cazul cel mai defavorabil al căii de propagare va fi cel din Fig.2. 5. în care se înlocuiesc primul şi ultimul bloc cu alte două blocuri p şi q, pe care le numim *bloc dominant superior*, respectiv *bloc dominant inferior* (pot exista mai multe perechi de asemenea blocuri – ce dau acelaşi timp de operare).

Timpul de însumare al acestui sumator este dat de cazul cel mai defevorabil al sumelor T_K^* date de următoarele:

- O întârziere de o poartă pentru a forma suma parţială, etc
- $2 \cdot (m_p - 1)$ τ pentru propagare prin blocul p; CY traversează blocul de la bitul 2 la bitul $(m_p - 1)$ al blocului.
- $(2 \cdot (q-p-1)+1)$ τ pentru logica *skip*; logica *skip* conţine două porţi (AND şi OR) pentru fiecare bloc omis, şi o poartă OR la nivelul ultimului bloc.
- $2 \cdot (m_p - 1)\tau$ pentru propagare prin blocul q; CY traversează blocul de la bitul 2 la bitul $(m_p - 1)$ al blocului.
- τ pentru formarea lui S_{n-1}, ceilalţi biţi sumă fiind deja formaţi.

BUPT

Astfel $T_K^\bullet = \left[2\cdot\left(m_q + m_p + q - p\right) - 3\right]\cdot\tau$ şi $T_k = \max\{T_K^\bullet\}$.Problema care se pune este să se aleagă $K, m_0, m_1, .., m_{K-1}$, astfel încât $m_0 + m_1 + ... + m_{K-1} = n$ şi T_k să fie minim.

Vom da un algoritm pentru determinarea blocurilor dominante în orice configuraţie. Justificarea algoritmului este următoarea: Presupunem că p este blocul dominant inferior. Un bloc j poate fi un candidat mai bun decât blocul q pentru blocul dominant superior dacă: $T_K^\bullet = \left[2\cdot\left(m_q + m_p + q - p\right) - 3\right]\cdot\tau$, adică dacă $m_j + j > m_q + q$. În mod similar, dacă presupunem că q este blocul dominant superior, atunci un bloc i poate fi un candidat mai bun decât blocul p pentru blocul dominant inferior dacă: $m_j + i > m_q + p$. Deci, blocurile dominante p şi q pentru orice configuraţie pot fi găsite prin aplicarea următorului algoritm:

```
i := K - 1 ; i := 0 ;
q := K - 1 ; p := 0 ;
repeat
        if m_j + j > m_q + q then q := j ;
        if m_i + i > m_p + p then p := i ;
        j := j - 1 ; i := i + 1
until j = i + 1 or j = i
```

Astfel, pe baza presupunerilor legate de timpul de operare făcute mai sus, este optimă o configuraţie cu numărul cel mai mic de blocuri a căror dimensiune creşte de la unul la altul, iar dimensiunea blocurilor adiacente diferă cu cel puţin o unitate (*condiţia de diferenţă prin unitate*).

Pentru un sumator dat pot fi mai multe configuraţii ce satisfac aceste condiţii şi pe de altă parte, pot exista şi altele ce nu o satisfac, dar care dau şi ele un timp de operare optim.

Această cerinţă de optimalitate o vom verifica mai departe în doi paşi.

1. În primul pas vom arăta că fiind dată o configuraţie ce nu satisface condiţia de diferenţă prin unitate, prin ajustarea dimensiunii blocurilor astfel încât această condiţie să fie îndeplinită, se va îmbunătăţi timpul de operare al sumatorului, sau în cel mai rău caz el va rămâne neschimbat.
2. În al doilea pas, vom arăta că pentru clasa de configuraţii ce satisface condiţia de diferenţă prin unitate, se poate îmbunătăţi timpul de operare dacă ne asigurăm ca blocurile marginale să aibă dimensiunea cea mai mică.

Pentru primul pas al verificării, se consideră o repetare a următorilor paşi până când nu mai sunt posibile alte schimbări:

1) se ia fiecare pereche de blocuri adiacente j şi $j-1$ astfel încât $|m_j - m_{j-1}| \geq 2$

2) Dacă $m_j > m_{j-1}$, atunci înlocuieşte perechea de blocuri $\lfloor m_j, m_{j-1} \rfloor$ cu $\lfloor m_j - 1, m_{j-1} + 1 \rfloor$, altfel înlocuieşte aceeaşi pereche cu $\lfloor m_j + 1, m_{j-1} - 1 \rfloor$. Acest proces va fi oprit când configuraţia dată satisface condiţia de diferenţă prin unitate.

Acum, pentru un pas de înlocuire arbitrar, fie k blocul a cărui dimensiune este micşorată şi fie l blocul a cărui dimensiune este crescută. Dacă înaintea reducerii:

BUPT

1. Blocul k a fost un bloc dominant

Atunci după reducere,

- dacă blocul k mai este bloc dominant, atunci nu au mai fost alte perechi de blocuri dominante; reducerea a îmbunătăţit totuşi timpul de operare.
- Dacă blocul k nu mai este un bloc dominant, atunci a existat şi o altă pereche de blocuri dominante; timpul de operare este totuşi nemodificat. De notat că, aşa cum arată următorul raţionament, creşterea dimensiunii blocului l nu a crescut timpul de operare.

  Presupunem că blocul l a devenit acum un bloc dominant şi că timpul de operare a crescut. În cazul cel mai defevorabil contribuţia căii *CY-skip* la timpul operaţional a crescut în noua configuraţie de 2τ ori. Contribuţia blocului k la timpul operaţional al configuraţiei originale a fost $2\cdot(m_k-1)\cdot\tau$, şi contribuţia blocului l la timpul operaţional al noii configuraţii este $2\cdot(m_l-1+1)\cdot\tau=2\cdot m_l\cdot\tau$. Deci creştera în timp este presupusă: $[2\cdot(m_l+2-2\cdot(m_k-1))]\cdot\tau=2\cdot(m_l-m_k+2)\cdot\tau$.

  Dar, întrucât $m_k-m_l\geq 2,\Rightarrow 2\cdot(m_l-m_k+2)\leq 0$, şi deci nu este nici o schimbare în fapt.

2. Blocul k nu a fost un bloc dominant.

Deci, dacă k nu a fost un bloc dominant şi $m_k-m_l\geq 2$; rezultă că creşterea în dimensiune a blocului l nu-l poate face bloc dominant; deci nu este nici o schimbare în timpul de operare.

Pentru al doilea pas de verificare, se începe prin a observa că dându-se o configuraţie $[m_0,m_1,...,m_{k-1}]$ ce satisface condiţia de diferenţă prin unitate, algoritmul anterior va duce la blocurile 0 şi $K-1$ ca şi blocuri dominante. Prin urmare problema iniţială se reduce la minimizarea lui $2\cdot(m_{K-1}+m_0+K-1)-3$, adică la minimizarea lui $(m_0+m_{K-1}+K)$. Aceasta se face după cum urmează.

Se ia o configuraţie ce satisface condiţia de diferenţă prin unitate şi se aplică o permutare π pentru a obţine o nouă configuraţie $[m_{\pi(0)},m_{\pi(1)},...,m_{\pi(K-1)}]$ astfel încât $m_{\pi(0)}\leq m_{\pi(1)}\leq m_{\pi(2)}\leq...\leq$ şi $m_{\pi(K-1)}\leq m_{\pi(K-2)}\leq m_{\pi(K-3)}\leq...$

Atunci timpul operaţional al noii configuraţii va fi mai bun, sau acelaşi, cu cel al vechii configuraţii, aceasta întrucât blocurile 0 şi $K-1$ sunt încă blocuri dominante, dar $m_{\pi(0)}\leq m_0$ şi $m_{\pi(K-1)}\leq m_{K-1}$. Odată ce m_0 sau m_K, sau ambele, sunt fixate, K poate fi minimizat făcând cât mai mare posibilă diferenţa între dimensiunile blocurilor adiacente. Întrucât diferenţa poate fi 0 sau 1, ea va fi aleasă să fie 1, dacă este posibil, altfel ea va fi 0.

În concluzie, configuraţia optimă pentru orice dimensiune de sumator va avea forma: $[i,i+1,i+2,...,j+2,j+1,j]$. Astfel, de exemplu pentru $n=12$, configuraţia $[1,2,3,3,2,1]$ va fi mai bună decât $[1,1,2,2,2,2,1,1]$.

Putem arăta acum cum trebuie construită configuraţia optimă pentru un n dat. Valoarea cea mai mică posibilă atât pentru i cât şi pentru j este 1, şi sunt două cazuri când ea poate fi aleasă să fie exact atât:

BUPT

I. (a) Dacă $n = k^2$, pentru un anumit k, atunci este optimă configurația cu $(2k-1)$ blocuri: $[1,2,...,k-1,k,k-1,...,1]$. Un sumator cu această configurație are un timp de operare de $(4k-3)\cdot\tau = (4\sqrt{n}-3)\cdot\tau \approx 4\sqrt{n}\tau$. De exemplu, pentru un sumator de 64 cifre binare este optimă configurația: $[1,2,3,4,5,6,7,8,7,6,5,4,3,2,1]$ și duce la un timp de adunare de $29\cdot\tau$.

(b) Dacă pentru un anumit k avem $n = k\cdot(k+1)$, atunci este optimă configurația de $2\cdot k$ blocuri: $[1,2,3,...,k,k,...,2,1]$. Un sumator cu această configurație are un timp de operare de: $(4k-1)\cdot\tau = (4\sqrt{\lfloor n \rfloor}-1)\approx 4\sqrt{n}\cdot\tau$. De exemplu, pentru un sumator de 30 de biți, este optimă configurația: $[1,2,3,4,5,5,4,3,2,1]$ și duce la un timp de operare de $19\cdot\tau$.

Dacă n nu satisface nici una din condițiile de mai sus, alegem cele mai mici k și l astfel încât $n < k\cdot(k+1) si n < l^2$ și:

II. (a) Dacă $k\cdot(k+1) < l^2$ atunci se pornește de la configurația: $[1,2,...,k,k,...,2,1]$ și se elimină în mod repetat blocurile marginale până când suma dimensiunilor sumatorului rămas este limitată de n.

De exemplu, dacă $n = 40$, atunci $k = 6$, $l = 7$, $k\cdot(k+1) = 42$ și $l^2 = 49$ și pornind de la configurația: $[1,2,3,4,5,6,6,5,4,3,2,1]$ se obține ca și configurație optimă configurația: $[2,3,4,5,6,6,5,4,3,2]$.

(b) Dacă $k\cdot(k+1) > l^2$ atunci se pornește de la configurația: $[1,2,...,l-1,l,l-1,...,2,1]$ și se elimină în mod repetat blocurile marginale până când suma dimensiunilor sumatorului rămas este limitată de n.

De exemplu, dacă $n = 32$, atunci $k = l = 6$, $k\cdot(k+1) = 42$ și $l^2 = 36$ și pornind de la configurația: $[1,2,3,4,5,6,5,4,3,2,1]$ se obține ca și configurație optimă configurația: $[2,3,4,5,6,5,4,3,2]$.

Timpul de operare a acestor ultime sumatoare se determină ușor pe baza observației că atunci când o configurație se obține dintr-o alta prin eliminarea blocurilor marginale, atunci nu apare nici o schimbare în timp: cu toate că blocul dominant al noii configurații este cu o unitate mai mare decât cel din vechea configurație, lungimea căii *CY-skip* pentru noua configurație este cu o unitate mai mică în cazul cel mai defavorabil.

Prin urmare, în cazurile 1(b) și 2(a) sumatorul rezultat va avea un timp de operare $(4k-1)\cdot\tau \approx 4\sqrt{n}\cdot\tau$, pe când în cazurile 1(a) și 2(b) sumatorul rezultat va avea un timp de operare de: $(4l-3)\cdot\tau \approx 4\sqrt{n}\cdot\tau$.

2.3.2 Analiza de performanță la sumatorul CSkA multinivel

În Fig.2. 6, având logica de omitere a lui CY la nivel de bloc simbolizată prin BSk și logica de omitere a lui CY la nivel de superbloc simbolizată prin SBSk, este dat cazul cel mai defavorabil din punct de vedere al performanței, el fiind compus din următoarele:

- τ pentru a forma T_i, suma parțială $PS_i = A_i \oplus B_i$ și CY parțial $A_i \cdot B_i$.
- $2\cdot(m-1)\cdot\tau$ pentru propagare prin primul bloc

BUPT

- $[2\cdot(M-2)+1]\cdot\tau$ pentru propagare prin blocurile *skip* $1,2,..,M-1$ a primului superbloc

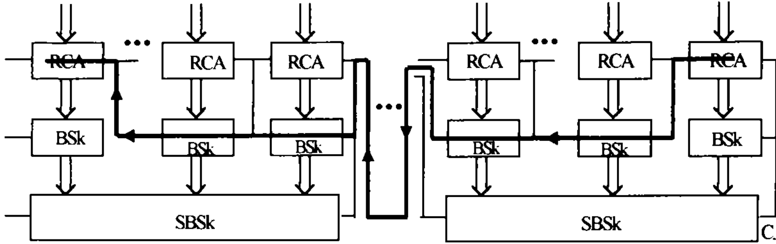


Fig.2. 6

- $2\cdot\left(\frac{n}{m\cdot M}-2\right)\cdot\tau$ pentru propagare prin logica *skip* a superblocurilor $1,2,..,\frac{n}{m\cdot M}-2$
- $[2\cdot(M-2)+1]\cdot\tau$ pentru propagare prin logica *skip* a blocurilor $1,2,..,M-1$ ale ultimului superbloc.
- $2\cdot(m-1)\cdot\tau$ pentru propagare prin ultimul bloc.
- τ pentru asimilarea lui C_{n-2} şi alui PS_{n-1} în formarea lui S_{n-1}, restul biţilor sumei fiind deja calculaţi.

Aceasta duce la un timp de operare de:

$$T_{OPCSk\text{-}lmm}\left(4\cdot m+4\cdot M+\frac{2n}{m\cdot M}-12\right)\cdot\tau. \quad (2.10)$$

Determinarea dimensiunii optime pentru blocuri şi pentru superblocuri este mai dificilă în cazul acestor sumatoare CSkA multinivel.

Cu toate că performanţele sumatoarelor CSkA nu sunt la fel de bune ca cele ale sumatoarelor deja discutate, s-a dovedit totuşi destul de populară utilizarea lor în conjuncţie cu alte tehnici de însumare (de exemplu cu tehnica *Manchester CY-chain*). Acest fapt se datorează regularităţii mari a structurii lor care favorizează implementarea lor în tehnologie de integrare pe scară largă.

Un sumator CSkA cu dimensiune de blocuri diferită nu este la fel de regular ca şi structură, şi deci s-ar putea să nu fie la fel de potrivit pentru anumite tehnologii de implementare.

Analize mai recente şi mai acurate, ce ţin cont de implementarea tehnologică a acestor sumatoare se găsesc în [GUYO87], [TURR89], [CHAN90], [HERO90] şi [CHAN92]; acestea includ discuţii legate de organizarea pe blocuri de dimensiune m, $m\geq 2$ a sumatoarelor CSkA.

[LYNC92] descrie proiectarea unui sumator VLSI ce combină tehnicile: *carry-lookahead*, *carry-select* şi *Manchester carry-chain*; un astfel de sumator a fost implementat în microprocesorul Am29050 de la Advanced Micro Devices.

BUPT

2.4 *Analiza de performanță la structurile de însumare cu propagare specială a transportului*

Analiza de performanță la sumatorul CdSumA

Un sumator CdSumA are $(1+\log_2 n)$ niveluri, și întrucât întârzierea prin fiecare nivel este 2τ, timpul de operare a unui asemenea sumator este:

$$T_{OPCdSumA} = 2\cdot(1+\log_2 n)\cdot\tau \qquad (2.11)$$

Proiectarea cu aceste sumatoare prezintă avantajul că acest sumator poate fi realizat complet *pipeline* [LEIG92], ceea ce va duce la îmbunătățirea performanțelor sale efective cu un factor egal cu $\log_2 n$. Un alt avantaj al unei asemenea proiectări este absența oricărei probleme legate de fan-in, întrucât se deduce pe baza ecuațiilor sale de funcționare că acesta nu va depăși niciodată valoarea 2.

Pe de altă parte există la acest sumator o problemă legată de fan-out. Astfel, la ultimul nivel al acestor sumatoare, semnalul $C_{\frac{n}{2}-1}$ este utilizat pentru selectarea a $n/2$ biți sumă. Astfel, dacă se folosește această tehnică pentru implementarea sumatoarelor mari, va fi necesar un alt aranjament.

In [OMON94] se dă un exemplu de utilizare a tehnicii *conditional-sum* în conjuncție cu o altă tehnică de proiectare a sumatoarelor. Un alt exemplu de implementare a unui sumator bazat pe această tehnică în conjuncție cu tehnica de detectare a sfârșitului propagării lui CY este descrisă în [MART80].

[GOSL80] descrie proiectarea unui sumator ce combină anticiparea la nivel de bloc cu însumarea condiționată.

Aceași tehnică a fost utilizată și în implementarea sistemului IBM RS/6000 [MONT90], [MARK90]; această implementare utilizează anumite avantaje ale tehnologiei VLSI și utilizează buffer-are suplimentară pentru satisfacerea cerințelor de fan-out la nivelul unui sumator de dimensiune medie.

Analiza de performanță la sumatorul CSlA

Timpul de operare a unui CSlA proiectat ca mai sus este $6\cdot\tau$, compus din următoarele:

- Pentru a forma P_i, G_i și T_i
- 2τ pentru a forma P_i^j, G_i^j, C_i^0 și C_i^1
- τ pentru a forma S_i^0 și S_i^1
- 2τ pentru a selecta suma

Cu toate că datele de performanță operațională ale acestui sumator sunt destul de bune, și cu toate că dimesiunea blocurilor pare să poată fi aleasă în mod arbitrar (în general se optează pentru minimizarea costului) este important de observat că, la nivelul sumatorului, apar aceleași limitări de fan-in și fan-out ca în cazul tuturor sumatoarelor ce au la bază CLA. Astfel, cerințele de fan-out și de fan-in depind de dimensiunea blocurilor. De exemplu, fan-in-ul și fan-out-ul cerut de sumatorul *lookahead-ripple carry select* (care este cel mai bun) de 64 biți, este același cu cel cerut de un sumator CLA în formă pură de 16 biți.

[TYAG93] a arătat că pentru realizări VLSI, cu doar un mic sacrificiu al spațiului de integrare, un sumator CSlA cu propagare serială între blocuri poate fi mai rapid decât un sumator CSkA cu un nivel de omitere. Astfel, acest tip de sumator devine atractiv pentru aceste tehnologii.

BUPT

Analiza de performanță la sumatorul PyA

Timpul operațional al sumatorului piramidal este dat de:

- un τ pentru nivelul inițial al logicii sale
- 2τ pentru fiecare din nivelurile rămase

Astfel, se obține un timp total de însumare de valoare:

$$T_{OPPyA} = (1 + 2 \cdot \log_2 n) \cdot \tau . \qquad (2.12)$$

Se observă că pentru valori mari ale lui n PyA probleme serioase legate de fan-out (atât $C_{n/2-1}$ cât și $S_{n/2}$ trebuie să alimenteze $\left(\frac{n}{2}+1\right)$ intrări) și de fan-in (valoarea maximă de fan-in implicat este $\left(\frac{n}{2}+1\right)$, ce apare la formarea lui *CY-out* final). În practică, aceste dificultăți legate de fan-out și fan-in, se rezolvă prin apelare la tehnici hibride de proiecare a sumatoarelor, în sensul că se utilizează tehnica de proiectare piramidală [LEIG92] la nivel de bloc, iar blocurile se înlănțuie pe baza utilizării unor alte tehnici de însumare.

BUPT

2.5 Eficientizarea testării prin C–testabilitate la structurile de însumare cu propagare serială a transportului

2.5.1 Testarea sumatorului RCA

Tehnicile de testare pseudoaleatoare bazate pe partiționare sunt indicate pentru testarea circuitelor structurate ca și arii logice iterative (*ILA*) ce sunt compuse din celule identice interconectate într-un *pattern* regulat, așa cum se arată în Fig.2. 7.

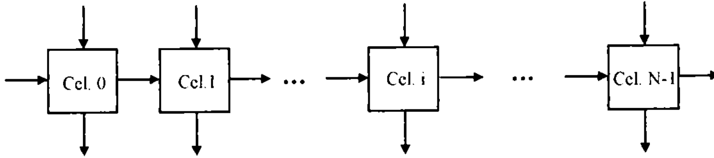


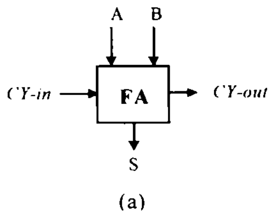
Fig.2. 7

Problema partiționării este rezolvată în mod natural prin utilizarea fiecărei celule ca și segment ce este testat exhaustiv.

Vom considera o arie logică iterativă unidimensională și unilaterală caracterizată prin faptul că interconectările între celule sunt într-o singură direcție. La început vom considera că celule sunt combinaționale. Un exemplu tipic pentru o asemenea structură este sumatorul RCA.

Anumite structuri de arii logice iterative au proprietatea că pot fi testate pseudoexhaustiv, cu un număr de teste ce nu depinde de numărul celulelor din structură. Aceste arii se spune că sunt *C-testabile*.

Următorul exemplu arată că sumatorul RCA este *C-testabil*.



(a)

	X, Y			
CY-in	0 0	0 1	1 0	1 1
0	0, 0	0, 1	0, 1	1, 0
1	0, 1	1, 0	1, 0	1, 1

(b)

Fig.2. 8

Considerăm tabelul de adevăr din Fig.2. 8.b. al unei celule de însumare a sumatorului RCA.

În acest tabel au fost separate valorile de intrare primară ale celulelor, ce sunt *A* și *B*, de valorile de intrare asigurate de celula anterioară (*CY-in*).

O intrare în tabel dă valoarea lui *CY-out* și a lui *S*. Se poate observa că $CY-out = CY-in$ pentru 6 combinații de intrare (celulele hașurate din tabel). Astfel, dacă aplicăm fiecărei celule a sumatorului RCA valorile *A* și *B* corespunzătoare unei asemenea intrări, atunci fiecare celulă va avea la intrare aceeași valoare *CY-in*. Deci, aceste 6 teste pot fi aplicate concurent la toate celulele.

Pentru celelalte 2 intrări avem $CY-out = \overline{CY-in}$. Testele corespunzătoare acestor 2 intrări pot fi aplicate celulelor alternative, așa cum se arată în Fig.2. 9.

BUPT

În concluzie, un sumator RCA de dimensiune arbitrară poate fi testat pseudoexhaustiv cu numai 8 teste, fiind astfel o structură logică iterativă *C-testabilă* [TUNG87].

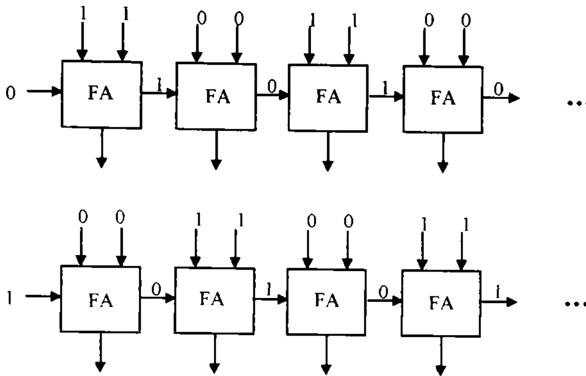


Fig.2. 9

	C_{-1}	A_0B_0	A_1B_1	$A_2\,B_2$	•••	$A_i\,B_i$	$A_{i+1}\,B_{i+1}$...	$S_0\,S_1\,S_2\,S_3$ •••	C_n
T_1	0	00	00	00	•••	00	00 •••	0 0 0 0 •••	0
T_2	0	01	01	01	•••	01	01 •••	1 1 1 1 •••	0
T_3	0	10	10	10	•••	10	10 •••	1 1 1 1 •••	0
T_4	1	01	01	01	•••	01	01 •••	0 0 0 0 •••	1
T_5	1	10	10	10	•••	10	10 •••	0 0 0 0 •••	1
T_6	1	11	11	11	•••	11	11 •••	1 1 1 1 •••	1
T_7	0	11	00	11	•••	00/11	11/00 •••	0 1 0 1 •••	0/1
T_8	1	00	11	00	•••	11/00	00/11 •••	1 0 1 0 •••	1/0

Tabel 2. 1

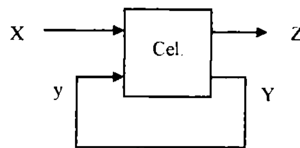


Fig.2. 10

Notăm cu $(y_0, x_0, x_1 ... x_{N-1})$ un vector de intrare pentru o structură logică iterativă cu N celule; unde y_0 este intrarea y aplicată primei celule, şi x_i $(i = 0,1...N-1)$ este intrarea x aplicată celulei i. Vom nota vectorul corespondent de ieşire cu $(Z_0, Z_1,... Z_{N-1})$ unde Z_i este răspunsul la ieşirea primară a celulei i. Considerăm un circuit secvenţial realizat dintr-o celulă a ariei prin conectarea ieşirilor Y la intrările y (Fig.2. 10). Dacă circuitul porneşte din starea y_0, răspunsul său la secvenţa de intrare $(y_0, x_0, x_1 ... x_{N-1})$ este secvenţa $(Z_0, Z_1,... Z_{N-1})$. Prin urmare, se poate utiliza circuitul secvenţial pentru reprezentarea structurilor logice iterative.

BUPT

Cu aceste precizări, se poate concluziona că verificarea tabelului de adevăr a unei celule din *ILA* corespunde verificării fiecărei intrări din tabelul de stare a circuitului secvențial echivalent.

Tabelul de adevăr a celulei sumatorului RCA (arătat în Tabel 2. 1.b) poate fi interpretat ca și tabel de stare a circuitului secvențial ce modelează sumatorul RCA [CHIH94].

Vom considera verificarea unei intrări (y, x) a tabelului de adevăr a unei celule *ILA*. Pentru a avea numărul total de teste independent de numărul celulelor din *ILA*, aceeași combinație de intrare (y, x) trebuie să fie aplicată simultan celulelor spațiate la intervale regulate (să zicem k) de-alungul ariei. Pentru exemplul sumatorului RCA avem $(k = 1)$ și $(k = 2)$. Aceasta înseamnă pentru circuitul secvențial ce modelează *ILA*, că secvența de k pași ce începe cu x, și se aplică în starea y, va trebui să aducă circuitul înapoi în starea y.
Atunci, un test aplică combinația de intrare (y, x) la fiecare a k-a celulă și k teste sunt suficiente pentru aplicarea ei la fiecare celulă din *ILA*. Următoarele rezultate caracterizează C-testabilitatea la *ILA*.

Teorema 2.1 Un *ILA* este *C-testabil* dacă și numai dacă circuitul său secvențial echivalent are un tabel de stări redus și dacă pentru oricare stare y există o secvență ce aduce circuitul înapoi în starea y [FRIE73].

Diagrama de stări a unui circuit secvențial ce modelează un *ILA* C-testabil este fie un graf puternic conectat (complet), fie este compusă din grafuri puternic conectate disjuncte.

Pentru generarea testelor de verificare a unei intrări (y, x) în tabelul de stare ce reprezintă un *ILA* C-testabil se utilizează un set de secvențe de distingere a perechilor de stări, care să permită verificarea stării următoare $N(y, x) = p$.

O secvență de distingere a unei perechi de stări p și q - $DS(p, q)$ - este o secvență de intrare pentru care secvențele de ieșire produse de circuit pornind din stările p și q sunt diferite. (Întrucât tabelul de stare se presupune a fi redus, $DS(p, q)$ există pentru oricare pereche de stări p și q).

De exemplu, pentru tabelul de stare din Fig.2. 11, $DS(y_0, y_1) = 1$ și $DS(y_1, y_2) = 01$.

	0	*1*
y_0	y_1, *0*	y_3, *1*
y_1	y_1, *0*	y_2, *0*
y_2	y_0, *0*	y_2, *0*
y_3	y_0, *0*	y_3, *1*

Fig.2. 11

Un set de secvențe de distingere a prechilor de stări pentru o stare p, $SDS(p)$, este un set de secvențe astfel încât pentru fiecare stare q diferită de p există o $DS(p, q)$ ce aparține lui $SDS(p)$.
Cu alte cuvinte, $SDS(p)$ conține secvențe ce disting starea p de oricare altă stare.

Pentru tabelul din Fig.2. 11, $SDS(y_1) = \{1, 01\}$, unde intrarea 1 servește atât pentru $DS(y_1, y_0)$ cât și pentru $DS(y_1, y_3)$.
Verificarea intrării (y, x) a tabelului redus de stare a unui circuit secvențial, sub presupunerea că circuitul se află în starea y se face pe baza următoarei proceduri [LOMB90]:

1. Se aplică la intrarea circuitului x, care va trece circuitul în starea p.

BUPT

2. Se aplică circuitului secvenţa D_i, ce aparţine lui $SDS(p)$. Presupunem că D_i duce circuitul în starea r_i.
3. Se aplică $T(r_i, y)$, secvenţă de transfer ce duce circuitul din starea r_i în starea y. (O asemenea secvenţă există întotdeauna pentru un ILA C-testabil).
4. Se reaplică spaţial secvenţa $I_i = (x, D_i, T(x_i, y))$ de câte ori este necesar pentru a avea o intrare aplicată fiecărei celule din ILA. Se obţine astfel un vector de test t_i. Fie k_i lungimea lui I_i. Testul t_i aplică (y, x) la fiecare a k-a celulă a lui ILA.
5. Aplică ariei k_i - 1 versiuni deplasate ale lui t_i.
6. Repetă paşii 1-5 pot orice altă secvenţă D_i din $SDS(p)$.

Aplicarea procedurii prezentate pentru verificarea stării $(y_0, 0)$ a tabelului de stare din Fig.2. 11 (ce modelează ILA C-testabil) duce la următoarele:
Fie $p = N(y_0, 0) = y_1$ şi $SDS(y_1) = \{1, 01\}$.

Primul grup de teste utilizează $D_1 = 1$, $r_1 = y_2$ şi $T(y_2, y_0) = 0$. Deci, $I_1 = (010)$ şi primul test este $t_1 = y_0(010)^*$ unde notaţia $(S)^*$ indică $SSS...$ (y_0 este aplicat primei celule). Întrucât $k_1 = 3$, aplicăm lui ILA 2 versiuni deplasate ale lui t_1; de asemenea trebuie să ne asigurăm că y_0 este aplicată primei celule testate din fiecare test. Cele două noi teste sunt $y_2 0(010)^*$ şi $y_1 10(010)^*$.

Al doilea grup de teste utilizează $D_2 = 01$. În mod similar se obţine $t_2 = y_0(0010)^*$. Întrucât $k_2 = 4$, sunt necesare trei versiuni deplasate ale lui t_2 pentru completarea acestui pas. Cele 7 teste obţinute verifică intrarea $(y_0,0)$ în toate celulele ILA, independent de numărul lor. Setul de teste pentru întregul ILA combină testele tuturor intrărilor din tabelul de adevăr al unei celule.

ILA considerat în cazul prezentării anterioare este unilateral, unidimensional şi compus din celule combinaţionale cu ieşiri direct observabile. Modelul de defect utilizat presupune că celula defectă rămâne circuit combinaţional. În continuare se amintesc pe scurt rezultate ale unor considerări mai generale.

Problema testării ILA -urilor ale căror celule nu au ieşiri direct observabile (adică numai ieşirile ultimei celule sunt PO) este studiată de Friedman [FRIE73], Parthasarathy şi Reddy [PART81].

Testarea ILA-urilor bidimensionale cu interconectări unidirecţionale este discutată în [ELHU86]. Shen şi Ferguson [SHEN84] au analizat proiectarea dispozitivelor de înmulţire VLSI, construite ca şi matrici bidimensionale şi au arătat cum pot fi modificate aceste proiecte pentru a deveni C-testabile.

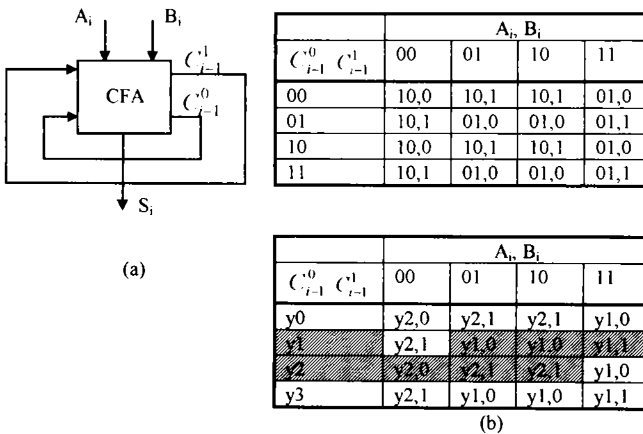
Cheng şi Patel [CHEN85] au dezvoltat un test pseudoexhaustiv pentru un sumator RCA sub presupunerea unui model de defect extins ce include şi defecte ce transformă o celulă defectă într-un circuit secvenţial cu 2 stări.

Sridhar şi Hayes [SRID81A] au examinat testarea ILA-urilor bilaterale în care interconectările între celule merg în ambele direcţii. Tot ai eu arătat cum metodele de testare a ILA-urilor combinaţionale pot fi extinse la ILA-uri compuse din celule secvenţiale.
Aceieaşi doi autori [SRID81A] [SRID81B] au studiat testarea sistemelor *bit-sliced*.

BUPT

2.5.2 Testarea sumatorului CCA

Schema acestui sumator (Fig.1.5.) este obținută prin interconectarea celulelor CFA într-o structură de tip ILA la care 1/3 din ieșiri sunt legate la o logică externă de obținere a semnalului carry-complete [POPE98D]. Modelul de defect utilizat presupune că celula defectă rămâne circuit combinațional.
ILA considerat este unilateral, unidimensional fiind compus din celule combinaționale (Fig.2. 12) al căror model secvențial prezintă două intrări primare (A_i, B_i), două intrări de stare (C_{i-1}^{0} și C_{i-1}^{1}) și tabelul de stare din Fig.2. 12.b.
Relațiile (1.2) și (1.3) dau ecuațiile ce stau la baza sintezei sumatorului CCA.



C_{i-1}^{0} C_{i-1}^{1}	A_i, B_i 00	01	10	11
00	10,0	10,1	10,1	01,0
01	10,1	01,0	01,0	01,1
10	10,0	10,1	10,1	01,0
11	10,1	01,0	01,0	01,1

C_{i-1}^{0} C_{i-1}^{1}	A_i, B_i 00	01	10	11
y0	y2,0	y2,1	y2,1	y1,0
y1	y2,1	y1,0	y1,0	y1,1
y2	y2,0	y2,1	y2,1	y1,0
y3	y2,1	y1,0	y1,0	y1,1

Fig.2. 12

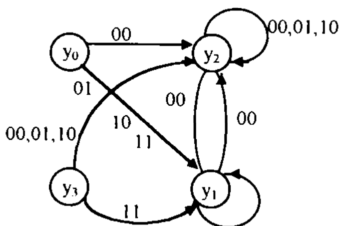


Fig.2. 13

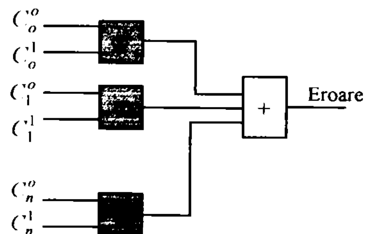


Fig.2. 14

În continuare se enunță și se demonstrează următoarea teoremă originală [POPE98D]:

Teorema 2. 2 Prin adăugarea la structura sumatorului CCA a unei logici suplimentare specifice, testarea lui completă pentru detecție în raport cu defectele de blocare singulară la 1 sau 0 se poate realiza cu numai 8 teste, indiferent de numărul rangurilor sumatorului.

BUPT

Demonstrație: Graful asociat celulei CFA (Fig.2. 13), nu este un graf complet conectat, datorită neconectării complete a stărilor y_3 şi y_0.

Întrucât semnalul *CY-complete* nu este 1 în starea y_0, această stare nu va fi niciodată activă la momentul verificării răspunsurilor de ieşire ale circuitului în regim de test, sau în regim de funcționare normală. Prin urmare, starea y_0 este o stare de tranziție în care sumatorul RCA intră la momentul inițierii adunării şi din care, pe măsură ce fiecare celulă realizează însumarea la nivelul ei, se trece fie în starea y_1, fie în starea y_2 (în funcție de propagările transporturilor). În cazul în care datorită defectării logicii de formare a semnalului *CY-complete* se ajunge să se facă citirea datelor de răspuns la momente în care există celule aflate în starea y_0, datele citite vor fi eronate.

Pe de altă parte, se poate face sesizată prezența stării eronate y_3 prin adăugarea unei logici suplimentare care să sesizeze la momentul citirii răspunsurilor dacă există celulă cu semnale $C_i^0 = C_i^1 = 1$. În Fig.2. 14 este prezentată această logică suplimentară impusă de realizarea C-testabilă a sumatorului CCA, cu observația că circuitele coincidență din figură pot fi substituite de porțile XOR ale celulelor, iar poarta OR ce se impune ca şi dispozitiv suplimentar va fi înlocuită de o poartă NOT-OR.

În aceste condiții, determinarea *pattern*-urilor de test se face pe baza verificărilor intrărilor liniilor y_1 şi y_2 ale tabelului din Fig.2. 12.b, pentru care graful asociat este complet conectat (Fig.2. 13).

Testarea completă a acestui ILA se realizează cu aceleaşi 8 teste din Tabel 2. 2. De notat că *pattern*-urile de la intrările A şi B sunt aceleaşi cu cele de la testarea sumatorului RCA.

	$C_{-1}^0\ C_{-1}^1$	A_0B_0	A_1B_1	$A_2\ B_2$	•••	$A_i\ B_i$	$A_{i+1}\ B_{i+1}$...	$S_0\ S_1\ S_2\ S_3$ •••
T_1	0 1	0 1	0 1	0 1	•••	0 1	0 1 •••	0 0 0 0 •••
T_2	0 1	1 1	1 1	1 1	•••	1 1	1 1 •••	1 1 1 1 •••
T_3	0 1	1 0	1 0	1 0	•••	1 0	1 0 •••	0 0 0 0 •••
T_4	1 0	0 0	0 0	0 0	•••	0 0	0 0 •••	0 0 0 0 •••
T_5	1 0	0 1	0 1	0 1	•••	0 1	0 1 ••	1 1 1 1 •••
T_6	1 0	1 0	1 0	1 0	•••	1 0	1 0 •••	1 1 1 1 •••
T_7	0 1	0 0	1 1	0 0	•••	11/00	00/11 •••	1 0 1 0 •••
T_8	1 0	1 1	0 0	1 1	•••	00/11	11/00 •••	0 1 0 1 •••

Tabel 2. 2

2.6 Eficientizarea testării prin C–testabilitate la structurile de însumare cu propagare anticipată a transportului

2.6.1 Testarea sumatorului CLA

Pattern-urile de test pentru testarea completă a sumatoarelor ce au la bază sumatoare de tip CLA în formă pură [POPE98A] se pot obține pe baza:

1. *pattern*-urilor de testare pseudoexhaustivă a blocurilor CLA ce formează sumatorul, sau a
2. setului complet de teste, obținut prin metode deterministe (funcționale), ce permite detecția tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale blocurilor sumatorului CLA.

BUPT

1. În prima situație, setul complet de teste pentru detecția defectelor sumatorului CLA la care dimensiunea blocurilor ce se însumează este *m*, este format din:
   - $2^m \cdot (2^m + 1)$ pentru care *CY-in=CY-out*
   - $2^m \cdot (2^m - 1)$ teste pentru care $CY - in = \overline{CY - out}$ pentru toate blocurile CLA, deci ele vor aplica valori de intrare alternative la nivelul blocurilor CLA dacă acestea se înlănțuie.

2. Pentru situația când setul complet de teste pentru detecția defectelor de blocare singulară la 1 sau 0 a sumatorului CLA este obținut în manieră deterministă (funcțională), am demonstrat următoarea teoremă originală [POPE98A]:

Teorema 2.3 Fiind dată structura de sumator CLA având logica CY implementată pe două niveluri logice (AND, OR) numărul total de teste pentru detecție a defectelor la un sumator cu n ranguri în raport cu defectele de blocare singulară la 1 sau 0 este:

$$N_{TCLA} = \frac{(n+2)\cdot(n+1)}{2} + 1. \qquad (2.13)$$

Demonstrație: La baza implementării acestor sumatoare stau ecuațiile:

$$S_i = (A_i \oplus B_i) \oplus C_{i-1}$$

$$C_i = G_i + P_i \cdot G_{i-1} + P_i \cdot P_{i-1} \cdot G_{i-2} + \ldots + P_i \cdot P_{i-1} \cdot P_{i-2} \cdot \ldots \cdot P_0 \cdot C_{-1}$$

unde $i = \overline{1,n}$.

Orice eroare singulară ce apare la nivelul termenilor ce intervin în expresia ieşirii observabile S_i va fi sesizată la ieşirea circuitului. Pentru identificarea erorilor la nivelul biților A_i şi B_i este suficientă aplicarea celor 4 combinații posibile ale acestor biți. Problema se pune în a identifica numărul testelor necesare pentru detecția defectelor expresiei logice ce corespunde lui C_{i-1}. Pe baza observației că defectele la nivelul semnalelor C_{i-1} sunt structural echivalente cu cele la nivelul semnalelor C_{n-1}, ne interesează mai departe să identificăm numărul testelor necesare pentru detectarea ultimelor. În Fig.2. 15. este dată implementarea de principiu pe bază de porți AND şi OR a ecuației ce corespunde semnalului C_{n-1}, precum şi defectele ce trebuiesc detectate la nivelul acesteia şi care au fost obținute prin aplicarea tehnicilor de *fault collapsing*. Rezultă că numărul total al defectelor ce trebuiesc detectate la nivelul acestui circuit este:

- $\sum_{2}^{k=n+1} k = \frac{(n+1)\cdot(n+2)}{2} - 1$ defecte de blocare la 1 a intrărilor circuitului
- 1 defect de blocare la 1 a ieşirii circuitului
- $(n+1)$ defecte de blocare la 0 a intrărilor porții OR ce generează semnalul de ieşire C_{n-1}

Numărul total al defectelor pentru care este necesară generarea testelor este: $\frac{(n+1)\cdot(n+2)}{2} + n + 1$. Ținând cont de faptul că testul:

T_1: $A_i = 0$ $B_i = 1$, $(i = \overline{0,n-1})$ având $C_{-1} = 0$ (şi generând $C_{n-1} = 0$),

BUPT

detectează toate defectele de punere pe 1 ale intrărilor G_i, $(i=0,n-1)$, precum şi defectele de punere pe 1 a ieşirii C_{n-1}, respectiv de blocare la 1 a lui C_{-1}; numărul testelor obţinute va fi cu n mai mic decât cel al defectelor urmărite pentru detecţie, adică $\frac{(n+1)\cdot(n+2)}{2}+1$.

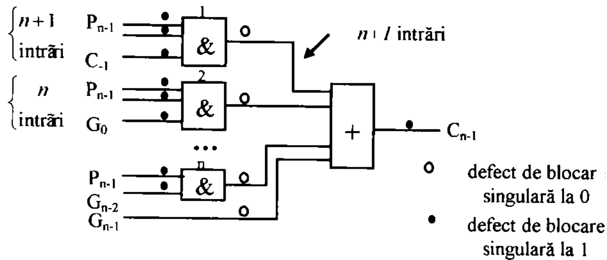


Fig.2. 15

Pentru detectarea oricăruia din celelalte defecte figurate în Fig.2. 15, este necesară generarea unor teste individuale conform următoarei proceduri:

1. Pentru defectele de blocare la 0 a intrărilor porţii OR este necesară aplicarea a n teste, ce se identifică după cum urmează:
   - Detectarea defectului de blocare la 0 a ieşirii primei porţi AND se face cu testul:

   T_2 : $A_i=1$ $B_i=0$, $(i=0,n-1)$ având $C_{-1}=1$ (şi generând $C_{n-1}=1$).
   - Pentru detectarea individuală a defectelor de blocare la 0 ale ieşirilor celorlalte $n-1$ porţi AND se aplică $n-1$ teste ce prezintă următoarele valori pentru testarea ieşirii porţii j, $(j=2,n)$:

   T_j : $A_{j-2}=1$, $B_{j-2}=1$ $j=2,n$ şi $A_i=0$, $B_i=1$ ($i=0,n-1$, $i\neq j-2$) având $C_{-1}=0$ (şi generând $C_{n-1}=1$).
   - Blocarea la 0 a lui G_{n-1} se detectează prin aplicarea testului:

   T_{n+2} $A_i=0$, $B_i=1$ ($i=0,n-2$) şi $A_{n-1}=1$, $B_{n-1}=1$ având $C_{-1}=0$ (şi generând $C_{n-1}=1$).

2. Pentru detectarea defectelor de blocare la 1 a intrărilor porţilor AND este necesară aplicarea a încă $\frac{n\cdot(n+1)}{2}$ teste; acestea se identifică după cum urmează:
   - Pentru detectarea defectelor de blocare la 1 a intrărilor P ale primei porţi AND, trebuiesc create condiţii de activare a defectului urmărit şi de propagare a acestuia, atât la nivelul porţii AND, cât şi la nivelul porţii de ieşire OR, fiind necesare în acest sens un număr de n teste.

     Astfel, testarea blocării la 1 a intrării P_k $(k=1,n-1)$ a porţii 1 se face prin aplicarea testului:

   T_k : $A_k=0$, $B_k=0$ împreună cu $A_i=0$, $B_i=1$, când $(i=0,n-1)$ şi $i\neq k$, sub condiţia: $C_{-1}=1$ (şi generând $C_{n-1}=0$).
   - Pentru detectarea defectelor de blocare la 1 a intrărilor P ale celorlalte porţi AND, trebuiesc create, pentru fiecare poartă AND în mod individual condiţii

BUPT

de activare a defectului şi de propagare a acestuia, atât la nivelul porţii AND, cât şi la nivelul porţii de ieşire OR. Astfel, pentru poarta j, $(j=\overline{2,n})$, testarea blocării la 1 a intrărilor P_k $(k=\overline{j-1,n-1})$ se face prin aplicarea a $(n+1-j)$ teste. În acest sens, testarea intrării P_k se face cu:

T_α $A_k=0$, $B_k=0$ împreună cu $A_{j-2}=1$, $B_{j-2}=1$ şi $A_i=0$, $B_i=1$ când $(i=\overline{0,n-1})$, $i \neq j-2$ şi $i \neq k$, sub condiţia $C_{-1}=0$ (şi generând $C_{n-1}=0$).

Numărul total de teste implicat de testarea blocării la 1 a intrărilor P a tuturor acestor porţi AND este $\frac{n\cdot(n-1)}{2}$.

.De notat că testele ce au fost determinate baleiază toate cele 4 combinaţii de intrare ale semnalelor A_i şi B_i, necesare pentru detecţia defectelor de blocare la 0 sau 1 ale acestora la nivelul intrărilor porţilor XOR ce implementează semnalele de ieşire S_i, $(i=\overline{1,n})$ ale sumatorului

În concluzie, numărul total al testelor necesare pentru detecţia defectelor de blocare singulară la 1 sau 0 a sumatoarelor CLA având logica CY realizată pe două niveluri logice este: $N_{TCLA}=\frac{(n+2)\cdot(n+1)}{2}+1$.

Tinând cont de această teoremă, în Anexa A Fig.A.1 se prezintă în pseudocod procedura de generare automată a testelor pentru un sumator CLA cu n ranguri de însumare.

2.6.2 Testarea sumatorului RCLA

Schema acestui sumator (Fig.1.8.) este obţinută prin interconectarea a *N* *(1 N n)* blocuri CLA (ce nu sunt C-testabile) de *m* biţi printr-o logică de propagare serială a transportului. Sumatorul în ansamblu are o structură de tip ILA ale cărui celule sunt constituite de blocurile CLA.

Modelul de defect utilizat presupune că celula defectă rămâne circuit combinaţional.

ILA ce corespunde structurii sumatorului RCLA este unilateral, unidimensional fiind compus din celule combinaţionale (sumatorul CLA) al căror model secvenţial prezintă m intrări primare (A_i, B_i) şi o intrare de stare.

Pattern-urile de test pentru testarea completă a sumatoarelor RCLA [POPE98C], ce au la bază sumatoare de tip CLA se pot obţine pe baza:

1. *pattern*-urilor de testare pseudoexhaustivă a blocurilor CLA ce formează sumatorul, sau a
2. setului complet de teste, obţinut prin metode deterministe (funcţionale), ce permite detecţia tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale blocurilor CLA.

1. În prima situaţie, setul complet de teste pentru detecţia defectelor sumatorului RCLA la care dimensiunea blocurilor ce se însumează este *m*, este format din:
   - $2^m\cdot(2^m+1)$ teste ce aplică aceleaşi valori de intrare la toate blocurile CLA, şi pentru care *CY-in=CY-out* pentru toate blocurile CLA
   - $2^m\cdot(2^m-1)$ teste pentru care $CY-in=\overline{CY-out}$ pentru toate blocurile CLA, deci ele vor aplica valori de intrare alternative la nivelul blocurilor CLA care se înlănţuie.

BUPT

De notat faptul că numărul acestor testelor este independent de numărul rangurilor sumatorului.
In Anexa A Fig.A2. se dă în pseudocod algoritmul de obținere a acestor teste pentru un sumator RCLA ce are $n = 2^k$ $(k \in N)$ ranguri și dimeniunea blocurilor este m.

2. Pentru situația când setul complet de teste pentru detecția defectelor de blocare singulară la 1 sau 0 a sumatorului CLA este obținut în manieră deterministă (funcțională), am demonstrat următoarea teoremă originală [POPE98C]:

Teorema 2.4 Fiind dată structura de sumator RCLA având blocuri de dimensiune m, numărul total de teste pentru detecția defectelor de blocare singulară la 0 sau 1 a unui sumator cu n ranguri este constant în raport cu n și are expresia:

$$N_{T\,RCLA} = 1 + \frac{(m+2)\cdot(m+1)}{2} \qquad (2.14)$$

dintre care $\left(2 + \sum_{k=1}^{m-1} k\right)$ sunt teste unice (se aplică simultan blocurilor pare și impare) și $(2 \cdot m)$ sunt teste ce aplică valori alterntive blocurilor sumatorului.

Demonstrație: Pe baza rezultatelor date de Teorema 2.3, s-a obținut că testarea completă a blocurilor CLA în formă pură se realizează cu un număr de $\left(\frac{(m+2)\cdot(m+1)}{2} + 1\right)$ teste, m fiind dimensiunea blocurilor.
Sumatorul RCLA este un ILA C-testabil, ale cărui celule prezintă o singură variabilă de stare (CY), și deci va trebui să identificăm, la nivelul testelor blocurilor CLA, care sunt testele pentru care $CY\text{-}in = CY\text{-}out$ (aceste teste se vor constitui ca și teste ce se aplică în mod unic atât blocurilor pare cât și celor impare) și care sunt testele pentru care $CY - in = \overline{CY - out}$ (aceste teste se vor aplica în manieră combinată blocurilor pare și celor impare ale sumatorului RCLA).

- Ținând cont de faptul că testul $A_i = 0$, $B_i = 1$, $(i = \overline{0, m-1})$ având $C_{-1} = 0$ generează $C_{m-1} = 0$, acest test se va constitui ca și test ce aplică valori unice la nivelul blocurilor sumatorului RCLA.
- Testul $A_i = 1$ $B_i = 0$, $i = \overline{0, m-1}$ având $C_{-1} = 1$ generează și el $C_{m-1} = C_{-1} = 1$, deci și acest test se va constitui ca și test ce aplică valori unice la nivelul blocurilor sumatorului RCLA.
- Cele $m-1$ teste ce prezintă următoarele valori pentru testarea blocării la 0 a ieșirii celei de a j-a porți AND:
  $A_{j-2} = 1$, $B_{j-2} = 1$ și $A_i = 0$, $B_i = 1$ ($i = \overline{0, m-1}$, $i \neq j-2$), $j = \overline{2, m}$ având $C_{-1} = 0$, generează $\overline{C_{m-1}} = C_{-1} = 0$. În aceeași situație este și testul ce detectează blocarea la 0 a lui G_{m-1}, având valorile de intrare: $A_i = 0$, $B_i = 1$ $(i = \overline{0, m-2})$ și $A_{m-1} = 1$, $B_{m-1} = 1$ cu $C_{-1} = 0$. Aceste teste vor fi alternate cu cele m teste ce detectează blocarea la 1 a intrărilor P_k $(k = \overline{1, m})$ ale primei porți AND; pentru testarea blocării la 1 a intrării P_k a porții 1 trebuiesc aplicate la intrările blocului sumator următoarele valori: $A_k = 0$, $B_k = 0$ și $A_i = 0$, $B_i = 1$ ($i = \overline{0, m-1}$, $i \neq k$),

BUPT

având $C_{-1} = 1$. Rezultă un total de $(2 \cdot m)$ teste ce aplică valori de intrare alternative blocurilor pare şi celor impare, şi anume:

1. un rând de teste ce au $C_{-1} = 0$, şi aplică
   - la nivelul blocurilor pare valorile: $A_{j-2} = 1$, $B_{j-2} = 1$ $j = \overline{2, m+1}$ şi $A_i = 0$, $B_i = 1$ ($i = \overline{0, m-1}$, $i \neq j-2$), iar
   - la nivelul celor impare valorile: $A_k = 0$, $B_k = 0$ $(k = \overline{0, m-1})$ şi $A_i = 0$, $B_i = 1$ ($i = \overline{0, m-1}$, $i \neq k$),
2. un alt rând de teste având $C_{-1} = 1$ iar ca şi valori ale intrărilor primare A şi B:
   - la nivelul blocurilor pare se aplică testele mai sus stabilite pentru blocurile impare, iar
   - la nivelul blocurilor impare se aplică testele mai sus stabilite pentru blocurile pare

- Pentru testarea blocării la 1 a intrărilor P_k $(k = \overline{j-1, n-1})$ la nivelul celei de a j-a porţi AND $(j = \overline{2, m})$ a unui bloc CLA se aplică $(m+1-j)$ teste ce au următoarele valori pentru testarea intrării P_k a porţii j $(j = \overline{2, m})$:

  $A_k = 0$, $B_k = 0$, împreună cu $A_{j-2} = 1$, $B_{j-2} = 1$ şi $A_i = 0$, $B_i = 1$ ($i = \overline{0, m-1}$, $i \neq j-2$ şi $i \neq k$), sub condiţia $C_{-1} = 0$.

  Nici unul din aceste teste nu generează transport, deci ele vor fi aplicate în mod unic la nivelul blocurilor pare şi al celor impare ale sumatorului.

În concluzie au fost obţinute un număr de $2 \cdot m$ teste ce aplică valori alterntive blocurilor CLA şi un număr de $2 + \sum_{j=2}^{m}(m+1-j) = 2 + \frac{(m-1) \cdot m}{2} = 2 + \sum_{k=1}^{m-1} k$ teste ce aplică valori unice tuturor blocurilor sumatorului RCLA.

Însumând testele determinate, obţinem că testarea completă în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip RCLA se face cu un număr de $N_{T\,RCLA} = 1 + \frac{(m+2) \cdot (m+1)}{2}$.

Bazat pe această teoremă, în Anexa A Fig.A.3 se prezintă în pseudocod procedura de determinare automată a *pattern*-urilor necesare testării complete pentru detecţia defectelor de blocare singulară la 1 sau la 0 a unui sumator RCLA cu n ranguri ce are dimensiunea m pentru blocurile CLA.

2.6.3 Testarea sumatorului BCLA

Schema acestui sumator (Fig.1.10.) este obţinută prin interconectarea a N $(1 \cdot N \; n)$ blocuri formate din sumatoare RCA (ce sunt C-testabile) de m biţi conectate la o logică de anticipare a transportului şi interconectate prin cascadarea transportului obţinut prin anticipare. Sumatorul în ansamblu are o structură de tip ILA ale cărui celule sunt constituite de blocurile sumatorului [POPE98B].

În Fig.1.10. este dată organizarea de principiu a unei părţi a unui asemenea sumator. Unitatea *CY-lookahead* implementează ecuaţiile de anticipare de la nivelul unui singur bloc, şi este detaliată în Fig.1.11.

BUPT

Modelul de defect utilizat presupune că celula defectă rămâne circuit combinațional.

ILA ce corespunde structurii sumatorului BCLA este unilateral, unidimensional fiind compus din celule combinaționale (blocurile sumatorului) al căror model secvențial prezintă m intrări primare (A_i, B_i) și o intrare de stare (*CY-in*). Pentru identificarea numărului total de teste necesar testării complete în raport cu defectele de blocare singulară la 1 sau 0 a sumatoarelor de tip BCLA am demonstrat următoarea teoremă originală [POPE98B].:

Teorema 2.5 Fiind dată structura de sumator BCLA având blocuri de dimensiune m, numărul total de teste pentru detecția defectelor de blocare singulară la 0 sau 1 a unui sumator cu n ranguri este constant în raport cu n și are expresia:

$$N_{TBCLA} = 7 + \frac{(m+1)\cdot(m+2)}{2}. \qquad (2.15)$$

Demonstrație: *Pattern*-urile de test pentru testarea completă a acestui tip de sumator se pot obține pe baza:

1. *pattern*-urilor de testare pseudoexhaustivă a sumatoarelor RCA ce compun blocurile sumatorului, și a
2. setului complet de teste, obținut prin metode deterministe (funcționale), ce permite detecția tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale logicii de anticipare a transporturilor la nivelul blocurilor.

În concluzie, setul complet de teste pentru detecția defectelor sumatorului BCLA este format din:

- cele 8 teste necesare testării complete a sumatoarelor RCA (date în Tabel 2.1) ce aparțin blocurilor sumatorului, la care se adaugă
- testele suplimentare necesare unei testări complete a logicii de anticipare a transporturilor, al căror număr este dependent de dimensiunea m a blocurilor, și anume: $(2\cdot m)$ teste ce aplică valori de test alternative blocurilor pare și impare ale sumatorului și $\left(2+\sum_{k=1}^{m-1} k\right)$ teste ce aplică valori unice atât blocurilor pare cât și celor impare. Ultimele teste includ testele:
  1. $A_i = 0$, $B_i = 1$, $(i = \overline{0, n-1})$ având $C_{-1} = 0$ și
  2. $A_i = 1$, $B_i = 0$, $(i = \overline{0, n-1})$ având $C_{-1} = 1$,

  teste ce au fost luate în considerare pentru testarea pseudoexhaustivă a blocurilor RCA. Prin urmare, ele trebuiesc excluse dintre testele ce realizează testarea logicii de anticipare a transportului între blocurile sumatorului BCLA. Deci, se mai iau în considerare doar $\sum_{k=1}^{m-1} k$ teste ce aplică valori unice atât blocurilor pare cât și celor impare.

Însumând testele determinate pe baza analizei sumatorului și ținând cont de proprietatea sa de C-testabilitate, putem concluziona că sunt necesare pentru testarea lui completă un număr de $8+\sum_{k=1}^{m-1} k = 8 + \frac{(m-1)\cdot m}{2}$ teste ce aplică valori unice blocurilor pare și impare ale sumatorului, și un număr de $(2\cdot m)$ teste ce aplică valori alternative la nivelul blocurilor sumatorului, rezultând astfel un total de $N_{TBCLA} = 7 + \frac{(m+1)\cdot(m+2)}{2}$.

BUPT

Utilizând Teorema 2.5, în Anexa A Fig.A.4. am prezentat în pseudocod procedura de determinare automată a *pattern*-urilor necesare testării complete pentru detecția defectelor de blocare singulară la 1 sau la 0 a unui sumator BCLA cu n ranguri ce are dimensiunea m pentru blocurile RCA.

2.6.4 Eficientizarea testării prin C–testabilitate la structurile de însumare organizate pe supeblocuri

2.6.4.1 Testarea sumatorului SBCLA

Schema acestui sumator (Fig.1.12.) este obținută prin interconectarea a N ($N = \frac{n}{M}$) supeblocuri, formate din sumatoare BCLA (ce sunt C-testabile) cu M blocuri de m biți, ce sunt conectate la o logică de anticipare a transportului și interconectate prin cascadarea transportului obținut prin anticipare. Sumatorul în ansamblu are o structură de tip ILA ale cărui celule sunt constituite de supeblocurile sumatorului, ce includ și ele la rândul lor ILA-uri constituite din blocurile sumatoarelor BCLA. În Fig.1.12. este prezentată organizarea de principiu a unei părți a unui asemenea sumator.
Unitatea *CY-lookahead* implementează ecuațiile de anticipare de la nivelul unui singur supebloc, și Fig.1.13. detaliază primul nivel de *lookahead.*

Modelul de defect utilizat presupune că celula defectă rămâne circuit combinațional.

ILA ce corespunde structurii sumatorului SBCLA este unilateral, unidimensional fiind compus din celule combinaționale (supeblocurile sumatorului) al căror model secvențial prezintă $M \cdot m$ intrări primare (A_i, B_i) și o intrare de stare (*CY-in*). Pentru testarea structurilor sumatoare de tip SBCLA în raport cu defectele de blocare singulară la 1 sau 0 demonstrez următoarea teoremă originală:

Teorema 2.6 Fiind dată structura de sumator SBCLA având superblocuri de dimensiune M formate din blocuri de dimensiune m, numărul total de teste pentru detecția defectelor de blocare singulară la 0 sau 1 a unui sumator cu n ranguri este constant în raport cu n și are expresia:

$$N_{TSBCLA} = \left(8 + \frac{m \cdot (m+3)}{2} + \frac{M \cdot (M+3)}{2}\right) \quad (2.16)$$

Demonstrație: *Pattern*-urile de test pentru testarea completă a acestui tip de sumator ce este o structură C-testabilă se pot obține pe baza:

1. *pattern*-urilor de testare pseudoexhaustivă a sumatoarelor RCA ce compun blocurile sumatorului,
2. setului complet de teste, obținut prin metode deterministe (funcționale), ce permite detecția tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale logicii de anticipare a transporturilor la nivelul blocurilor superblocurilor.

În concluzie, setul complet de teste pentru detecția defectelor sumatorului SBCLA este format din:

- cele 8 teste necesare testării complete a sumatoarelor RCA ce aparțin blocurilor sumatorului la care se adaugă

BUPT

- testele suplimentare necesare unei testări complete a logicii de anticipare a transporturilor la nivelul blocurilor supeblocurilor, al căror număr este dependent de dimensiunea blocurilor (m). Conform teorema 2.3 numărul acestor teste este $\frac{(m+2)\cdot(m+1)}{2}+1$, dar întrucât ele includ testele:
  1. $A_i = 0$, $B_i = 1$, $(i = \overline{0,n-1})$ având $C_{-1} = 0$ şi
  2. $A_i = 1$, $B_i = 0$, $(i = \overline{0,n-1})$ având $C_{-1} = 1$,

  ce au fost luate în considerare pentru testarea pseudoexhaustivă a blocurilor RCA, aceste două teste trebuiesc excluse dintre testele ce realizează testarea logicii de anticipare a transportului între blocurile sumatorului SBCLA. Deci, se iau în considerare ca şi teste suplimentare doar cele $\sum_{k=1}^{m-1} k = \frac{(m-1)\cdot m}{2}$ teste ce aplică valori unice atât blocurilor pare cât şi celor impare, precum şi cele $(2\cdot m)$ teste ce aplică valori alternative blocurilor respective, rezultând un număr de $\frac{m\cdot(m+3)}{2}$
- testele suplimentare necesare unei testări complete a logicii de anticipare a transporturilor la nivelul supeblocurilor, al căror număr este dependent de dimensiunea supeblocurilor (M); acestea la rândul lor fiind obţinute pe baza testelor necesare testării complete a structurilor de însumare de tip CLA.
  Conform Teoremă 2.3 numărul testelor este $\frac{(M+2)\cdot(M+1)}{2}+1$, din care din nou trebuiesc excluse cele două teste comune cu cele ale setului ce testează sumatoarele RCA.
  Deci, se iau în considerare ca şi teste suplimentare impuse de testarea logicii de anticipare a transportului la nivelul supeblocurilor doar cele $\sum_{k=1}^{m-1} k = \frac{(M-1)\cdot M}{2}$ teste ce aplică valori unice atât supeblocurilor pare cât şi celor impare, precum şi cele $(2\cdot M)$ teste ce aplică valori alternative supeblocurilor respective, rezultând un număr de $\frac{M\cdot(M+3)}{2}$

Însumând testele determinate pe baza analizei sumatorului şi ţinând cont de proprietatea sa de C-testabilitate, putem concluziona că sunt necesare pentru testarea lui completă un număr de $N_{TSBCLA} = \left(8 + \frac{m\cdot(m+3)}{2} + \frac{M\cdot(M+3)}{2}\right)$ teste.

Bazat pe această teoremă, în Anexa A Fig.A.5 am prezentat în pseudocod procedura de determinare a *pattern*-urilor necesare testării complete pentru detecţia defectelor de blocare singulară la 1 sau la 0 a unui sumator SBCLA cu n ranguri ce are supeblocuri de dimensiune M formate din blocuri de dimensiune m:

2.6.4.2 Testarea sumatorului ISBCLA

Sumatorul SBCLA îmbunătăţit (ISBCLA) este şi el o structură de tip ILA, la care celula este dată de configuraţia obţinută prin modificarea schemei supeblocului sumatorului SBCLA (în sensul că toate CY-urile de bloc ale nivelului supebloc sunt produse simultan), care la rândul ei este constituită printr-o repetare spaţială a blocurilor sumatorului BCLA.

BUPT

Sumatorul ISBCLA este prezentat în Fig.1.14. În Fig.1.15 şi Fig.1.16 se prezintă detalii ale unităţilor *lookahead*.
Modelul de defect utilizat presupune că celula defectă rămâne circuit combinaţional.

ILA ce corespunde structurii sumatorului ISBCLA este unilateral, unidimensional fiind compus din celule combinaţionale (supeblocurile sumatorului) al căror model secvenţial prezintă $M \cdot m$ intrări primare (A_i, B_i) şi o intrare de stare (*CY-in*).
Pattern-urile de test pentru testarea completă a acestui tip de sumator sunt aceleaşi cu cele obţinute pentru sumatorul SBCLA.

2.6.4.3 Testarea sumatorului SRCLA

Schema acestui sumator (Fig.1.17.) este obţinută prin interconectarea a N $(N = \frac{n}{M \cdot m})$ supeblocuri formate din M blocuri sumatoare RCLA (ce sunt C-testabile) de m biţi cu două niveluri de anticipare a transportului
Sumatorul în ansamblu are o structură de tip ILA ale cărui celule sunt constituite de supeblocurile sumatorului, ce includ şi ele la rândul lor ILA-uri constituite din blocurile sumatoarelor RCLA.

Modelul de defect utilizat presupune că celula defectă rămâne circuit combinaţional.

ILA ce corespunde structurii sumatorului SRCLA este unilateral, unidimensional fiind compus din celule combinaţionale (supeblocurile sumatorului) al căror model secvenţial prezintă $(M \cdot m)$ intrări primare (A_i, B_i) şi o intrare de stare (*CY-in*).
Pattern-urile de test pentru testarea completă a acestui tip de sumator se pot obţine pe baza:

1. *Pattern*-urilor de testare pseudoexhaustivă a sumatoarelor CLA ce compun blocurile sumatorului, al căror număr este dependent de dimensiunea blocurilor (*m)*
2. Setului complet de teste, obţinut prin metode deterministe (funcţionale), ce permite detecţia tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale logicii de anticipare a transporturilor la nivelul supeblocurilor. Numărul lor este dependent de dimensiunea supeblocurilor (*M*), fiind şi ele obţinute pe baza pattern-urilor de test necesare testării structurilor CLA.

Pattern-urile de test pentru testarea completă a acestui tip de sumator sunt aceleaşi cu cele obţinute pentru sumatorul SBCLA.

2.6.5 Testarea sumatorului PyCLA

Sumatorul PyCLA are şi el o structură regulată şi repetabilă, la care setul complet de teste pentru detecţia defectelor de blocare singulară la 0 sau 1 se obţine recursiv prin parcurgerea următorilor paşi [POPE98G]:

1. Pe baza testelor necesare testării exhaustive a celulei de tip A din Fig.2. 16, şi ţinând cont de condiţia ca *CY-in* să fie egal cu *CY-out*, condiţie ce permite realizarea repetabilităţii stării de pornire pentru circuitul secvenţial ce modelează celula, se determină setul de teste ce realizează testarea completă a acestor celule repetate spaţial pe o direcţie.

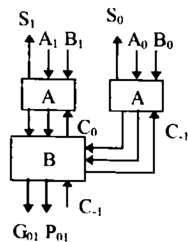


Fig.2. 16

BUPT

Cele 8 teste determinate în maniera prezentată mai sus sunt cuprinse în Tabel 2. 1

2. Pentru testarea completă a celulei compuse din Fig.2. 16, se adaugă la cele 8 teste deja determinate testele ce sunt necesare testării complete a celulei B adăugate la rădăcina arborelui binar dezvoltat pe baza subarborilor formaţi din celule de tip A. Identificarea acestor teste se face sub condiţia repetabilităţii stării de pornire a circuitului secvenţial ce modelează noua celulă compusă.

3. Pentru testarea completă a celulei din Fig.2. 17, se adaugă la testele deja determinate cele necesare testării complete a celulei B adăugate la rădăcina arborelui binar dezvoltat pe baza celulelor din Fig.2. 16. Pentru dezvoltarea testelor suplimentare, din nou trebuie să se ţină cont de condiţia ca *CY-in* să fie restaurat prin valoarea lui *CY-out* (la distanţă maximă 2), condiţie ce permite realizarea repetabilităţii stării de pornire pentru circuitul secvenţial ce modelează celula.

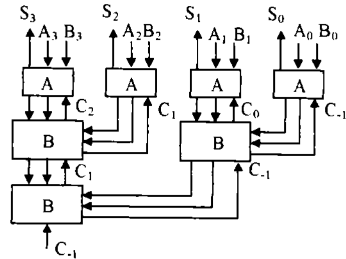


Fig.2. 17

4. Prin conectarea pe baza unei celule de tip B a doi subarbori ce au fost deja generaţi şi pentru testarea completă a cărora au fost determinate seturile de teste necesare, se obţine un nou arbore. Pentru testarea completă a noului arbore (celula din Fig.2. 18), se adaugă la testele deja determinate cele necesare testării complete a celulei B adăugate la rădăcina arborelui binar dezvoltat în maniera prezentată (pe baza celulelor compuse din Fig.2. 17).
Pentru dezvoltarea testelor suplimentare, din nou trebuie să se ţină cont de condiţia ca ca *CY-in* să fie restaurat prin valoarea lui *CY-out* (la distanţă maximă 2), condiţie ce permite realizarea repetabilităţii stării de pornire pentru circuitul secvenţial ce modelează celula.

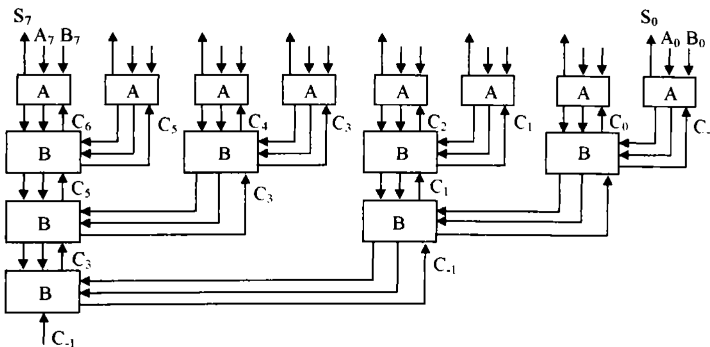


Fig.2. 18

5. Se repetă pasul 4 până la momentul când arborele obţinut este de dimensiunea sumatorului ce se doreşte pentru implementare.

BUPT

Ținând cont de procedura de elaborare a testelor pentru structurile de însumare de tip PyCLA, descrisă mai sus, am enunțat și demonstrat pentru aceste structuri, următoarea teoremă originală [POPE98G]:

Teorema 2.7. Fiind dată structura de sumator PyCLA cu $n = 2^k, (k \in N)$ ranguri, sintetizată pe $\log_2 n$ niveluri logice, numărul total de teste pentru detecția defectelor de blocare singulară la 1 sau 0 la un astfel de sumator este:

$$N_{TPyCLA} = 8 + 14 \cdot \log_2 \frac{n}{2} = 8 + 14 \cdot (k-1). \tag{2.17}$$

Demonstrație:

1. Se pornește de la determinarea testelor pentru testarea celulelor de tip A, ce prezintă cele 2 intrări primare: A_i, B_i și intrarea de stare C_{i-1}. Ele implementează următoarele ecuații logice:
   $S_i = A_i \oplus B_i \oplus C_{i-1}$
   $p_i = A_i \oplus B_i$
   $g_i = A_i \cdot B_i$
   Impunând condiția de repetare spațială a valorii intrării de stare pentru aceste celule de tip A, se obține un număr de 6 teste pentru care *CY-in* = *CY-out* (este vorba de primele 6 teste din Tabel 2. 1), precum și 2 teste pentru care *CY-in* la nivelul unei celule este egal cu $\overline{CY-out}$ generat de această celulă. Testele corespunzătoare acestor situații vor fi aplicate celulelor alternative, așa cum se arată în Fig.2. 9; ele corespund ultimelor două teste din Tabel 2. 1.
   În concluzie, testarea completă pe baza condițiilor impuse de C-testabilitatea structurii iterative obținută prin spațierea pe orizontală a celulelor de tip A este realizată cu ajutorul celor 8 teste date în Tabel 2. 1.

2. Se continuă prin identificarea pentru adăugare la testele deja determinate a celor necesare testării complete pentru detecție a celulelor de tip B ce se adaugă la baza a doi subarbori constituiți în primă fază de celule de tip A, iar în fazele următoare de subarbori deja constituiți și pentru care au fost deja determinate teste necesare testării lor complete.
   Generalizând pentru $\forall j$, cu $i < j, j+1 < k$, avem următoarele ecuații recursive la nivelul celulelor B:
   $C_k = G_{ik} + P_{ik} \cdot C_{k-1}$
   $G_{ik} = G_{j+1,k} + P_{j+1,k} G_{ij}$
   $P_{ik} = P_{ij} \cdot P_{j+1,k}$
   Aceste ecuații exprimă faptul că se generează un CY la nivelul blocului ce conține biții de la i la k (inclusiv) când:
   - el este generat de partea superioară a blocului ce este compusă de biții $(j+1, k)$, sau
   - el este generat de partea inferioară a blocului, compusă de biții (i, j) fiind propagat pe urmă peste partea superioară a blocului.

   Ecuațiile sunt valabile și pentru $i \le j < k$ dacă se setează $G_{ij} = g_i$ și $P_{ij} = p_i$.
   Testarea exhaustivă a celulei B ce are 5 semnale de intrare presupune stimularea ei cu *pattern*-urile date în Tabel 2. 3 (restul *pattern*-urilor nefiind combinații admise ale

BUPT

intrărilor celulei B), unde au fost grupate în mod separat *pattern*-urile ce corespund subarborelui stâng de cele ce corespund subarborelui drept al arborelui obținut prin conectarea celulei de tip B. Pentru identificarea situației prezenței la intrările celulelor de tip B a unor combinații de intrare nepermise, se modifică structura acestora, în sensul adăugării unei logici suplimentare ce să realizeze un autocontrol la nivelul lor. Logica adăugată implementează ecuația:

$$P_{ik} = P_{ij} \cdot P_{j+1,k} + P_{ij} \cdot G_{ij} + P_{j+1,k} \cdot G_{j+1,k};$$

Această modificare a structurii celulelor B evită mascarea defectelor mai sus amintite.

Pentru 8 dintre liniile din Tabel 2. 3 avem $CY - in \neq \overline{CY - out}$; prin urmare, în vederea obținerii unei repetabilități spațiale pentru condițiile de stare ale unor asemenea subarbori, ce se constituie ca și celule ILA repetate spațial și obținute pe baza dezvoltărilor prin conectări cu celule de tip B a subarborilor obținuți în pași anteriori (dezvoltări ce continuă până la atingerea dimensiunii de însumare propuse), testele celor 8 linii vor trebui grupate pentru alternarea semnalelor *CY-in* cu *CY-out* la nivelul celulelor alăturate.

În continuare se urmărește modul în care se face determinarea testelor necesare testării complete a ultimei celule B adăugate în vederea dezvoltării dimensiunii de însumare.

	$P_{j+1,k}$	$G_{j+1,k}$	P_{ij}	G_{ij}	C_{-1}	C_k	Valori de intrare $A_i B_i$ pentru subarbore stâng	Valori de intrare $A_i B_i$ pentru subarbore drept
1	0	0	0	0	0	0	00 00 ... 00 00	00 00 ... 00 00
2	0	0	0	0	*1*	*0*	*00 00 ... 00 00*	*00 00 ... 00 00*
3	0	0	0	1	0	0	00 00 ... 00 00	11 11 ... 11 11
4	0	0	0	1	*1*	*0*	*00 00 ... 00 00*	*11 11 ... 11 11*
5	0	0	1	0	0	0	00 00 ... 00 00	01 01 ... 01 01
6	0	0	1	0	*1*	*0*	*00 00 ... 00 00*	*01 01 ... 01 01*
7	0	1	0	0	*0*	*1*	*11 11 ... 11 11*	*00 00 ... 00 00*
8	0	1	0	0	1	1	11 11 ... 11 11	00 00 ... 00 00
9	0	1	0	1	*0*	*1*	*11 11 ... 11 11*	*11 11 ... 11 11*
10	0	1	0	1	1	1	11 11 ... 11 11	11 11 ... 11 11
11	0	1	1	0	*0*	*1*	*11 11 ... 11 11*	*01 01 ... 01 01*
12	0	1	1	0	1	1	11 11 ... 11 11	01 01 ... 01 01
13	1	0	0	0	0	0	01 01 ... 01 01	00 00 ... 00 00
14	1	0	0	0	*1*	*0*	*01 01 ... 01 01*	*00 00 ... 00 00*
15	1	0	0	1	*0*	*1*	*01 01 ... 01 01*	*11 11 ... 11 11*
16	1	0	0	1	1	1	01 01 ... 01 01	11 11 ... 11 11
17	1	0	1	0	0	0	01 01 ... 01 01	01 01 ... 01 01
18	1	0	1	0	1	1	01 01 ... 01 01	01 01 ... 01 01

Tabel 2. 3

- testarea liniei 1 din tabelul de funcționare a celulei B este realizată de testul T_1 deja determinat
- testarea liniei 2 din Tabel 2. 3, pentru care $CY - in \neq \overline{CY - out}$, presupune obținerea prin combinare a unui test ce să realizeze refacerea valorii intrării de stare cu ajutorul nivelului următor de celule compuse (constituite de arborele format prin adăugarea acestei celule B și care va fi repetat spațial prin dezvoltările ulterioare a dimensiunii celulei compuse obținute până la atingerea dimesiunii *n*) pentru nivelul ce-i urmează, testându-se în felul acesta celulele impare. Testarea celulelor compuse de paritate pară în raport cu intrările ce corespund liniei 2 din Tabel 2. 3 se va face

BUPT

prin utilizarea unui C_{-1} de valoare complementară celei corespunzătoare testului anterior determinat, şi prin inversarea valorilor *pattern*-urilor de test aplicate la nivelul celulelor pare cu cele ale celulelor impare. Alegând pentru combinarea *pattern*-ului înscris în linia 2 un *pattern*-ul de test ce corespunde liniei 9, se obţin următoarele două teste ce realizează testarea atât a celulelor pare cât şi a celor impare atât în raport cu intrările liniei 2 cât şi cu cele ale liniei 9:

T_{8+1} : aplică $C'_{-1} = 0$ având ca şi valori ale intrărilor primare ale celulelor pare $A_l = 0$, $B_l = 0$ $(l = i,k)$ iar pentru cele ale celulelor impare $A_l = 1$, $B_l = 1$ $(l = i,k)$.

T_{8+2} : aplică $C'_{-1} = 1$ având ca şi valori ale intrărilor primare ale celulelor pare $A_l = 1$, $B_l = 1$ $(l = i,k)$ şi pentru cele ale celulelor impare $A_l = 0$, $B_l = 0$ $(l = i,k)$.

- testarea liniei 3 se realizează prin aplicarea în paralel la nivelul tuturor grupărilor de $(k - i + 1)$ ranguri, ce se constituie ca şi celule ale dezvoltărilor ulterioare, a testului:

T_{8+3} : aplică $C'_{-1} = 0$ având ca şi valori ale intrările primare pentru subarborii drept ai celulelor: $A_l = 1$, $B_l = 1$ $(l = i,j)$ şi pentru subarborii stâng $A_l = 0$, $B_l = 0$ $(l = j+1,k)$.

- testarea liniei 4 presupune formarea din nou prin combinare a unui test ce să permită testarea celulelor impare în raport cu linia 4 din Tabel 2. 3, iar a celor pare în raport cu o altă linie (7) a Tabel 2. 3, şi a altui test ce să realizeze testarea celulelor pare în raport cu linia 4 din Tabel 2. 3, şi a celor pare în raport cu linia 7 din Tabel 2. 3.

T_{8+4} : aplică $C'_{-1} = 0$ având ca şi valori ale intrărilor primare

  - pentru subarborii drept ai celulelor pare $A_l = 1$, $B_l = 1$ $(l = i,j)$
  - pentru subarborii stâng ai celulelor pare $A_l = 0$, $B_l = 0$ $(l = j+1,k)$.
  - pentru subarborii drept ai celulelor impare $A_l = 0, B_l = 0$ $(l = i,j)$
  - pentru subarborii stâng ai celulelor impare $A_l = 1, B_l = 1$ $(l = j+1,k)$.

T_{8+5} : aplică $C'_{-1} = 1$ având ca şi valori ale intrărilor primare

  - pentru subarborii drept ai celulelor impare $A_l = 1$, $B_l = 1$ $(l = i,j)$
  - pentru subarborii stâng ai celulelor impare $A_l = 0$, $B_l = 0$ $(l = j+1,k)$.
  - pentru subarborii drept ai celulelor pare $A_l = 0$, $B_l = 0$ $(l = i,j)$
  - pentru subarborii stâng ai celulelor pare $A_l = 1$, $B_l = 1$ $(l = j+1,k)$.

- testarea liniei 5 se realizează prin aplicarea în paralel la nivelul tuturor grupărilor de $(k - i + 1)$ ranguri, ce se constituie ca şi celule ale dezvoltărilor ulterioare, a testului:

T_{8+6} : aplică $C'_{-1} = 0$ având ca şi valori ale intrările primare pentru subarborii drept ai celulelor: $A_l = 0$, $B_l = 1$ $(l = i,j)$ şi pentru subarborii stâng $A_l = 0$, $B_l = 0$ $(l = j+1,k)$.

- testarea liniei 6 presupune formarea din nou prin combinare a unui test ce să permită testarea celulelor impare în raport cu linia 6 din Tabel 2. 3, iar a celor pare în raport cu o altă linie (11) din Tabel 2. 3, şi a altui test ce să realizeze testarea celulelor pare în raport cu linia 6 din Tabel 2. 3, şi a celor pare în raport cu linia 11 din Tabel 2. 3.

T_{8+7} : aplică $C'_{-1} = 0$ având ca şi valori ale intrărilor primare

  - pentru subarborii drept ai celulelor pare $A_l = 0$, $B_l = 1$ $(l = i,j)$
  - pentru subarborii stâng ai celulelor pare $A_l = 0$, $B_l = 0$ $(l = j+1,k)$.

BUPT

- pentru subarborii drept ai celulelor impare $A_l = 0$, $B_l = 1$ $(l = i, j)$
- pentru subarborii stâng ai celulelor impare $A_l = 1$, $B_l = 1$ $(l = j+1, k)$.

T_{8+8} : aplică $C_{-1} = 1$ având ca şi valori ale intrărilor primare

- pentru subarborii drept ai celulelor impare $A_l = 0$, $B_l = 1$ $(l = i, j)$
- pentru subarborii stâng ai celulelor impare $A_l = 0$, $B_l = 0$ $(l = j+1, k)$.
- pentru subarborii drept ai celulelor pare $A_l = 0$, $B_l = 1$ $(l = i, j)$
- pentru subarborii stâng ai celulelor pare $A_l = 1$, $B_l = 1$ $(l = j+1, k)$.

- testarea liniei 8 se realizează prin aplicarea în paralel la nivelul tuturor grupărilor de $(k - i + 1)$ ranguri, ce se constituie ca şi celule ale dezvoltărilor ulterioare, a testului:

T_{8+9} : aplică $C_{-1} = 1$ având ca şi valori ale intrările primare pentru subarborii drept ai celulelor: $A_l = 0$, $B_l = 0$ $(l = i, j)$ şi pentru subarborii stâng $A_l = 1$, $B_l = 1$ $(l = j+1, k)$.

- T_6 este cel ce realizează testarea liniei 10 din Tabel 2. 3
- testarea liniei 12 se realizează prin aplicarea în paralel la nivelul tuturor grupărilor de $(k - i + 1)$ ranguri, ce se constituie ca şi celule ale dezvoltărilor ulterioare, a testului:

T_{8+10} : aplică $C_{-1} = 1$ având ca şi valori ale intrările primare pentru subarborii drept ai celulelor: $A_l = 0$, $B_l = 1$ $(l = i, j)$ şi pentru subarborii stâng $A_l = 1$, $B_l = 1$ $(l = j+1, k)$.

- testarea liniei 13 se realizează prin aplicarea în paralel la nivelul tuturor grupărilor de $(k - i + 1)$ ranguri, ce se constituie ca şi celule ale dezvoltărilor ulterioare, a testului:

T_{8+11} : aplică $C_{-1} = 0$ având ca şi valori ale intrările primare pentru subarborii drept ai celulelor: $A_l = 0$, $B_l = 0$ $(l = i, j)$ şi pentru subarborii stâng $A_l = 0$, $B_l = 1$ $(l = j+1, k)$.

- testarea liniei 14 presupune formarea din nou prin combinare a unui test ce să permită testarea celulelor impare în raport cu linia 14 din Tabel 2. 3, iar a celor pare în raport cu o altă linie (15) din Tabel 2. 3, şi a altui test ce să realizeze testarea celulelor pare în raport cu linia 14 din Tabel 2. 3, şi a celor pare în raport cu linia 15 din Tabel 2. 3.

T_{8+12} : aplică $C_{-1} = 0$ având ca şi valori ale intrărilor primare

- pentru subarborii drept ai celulelor pare $A_l = 0$, $B_l = 0$ $(l = i, j)$
- pentru subarborii stâng ai celulelor pare $A_l = 0$, $B_l = 1$ $(l = j+1, k)$.
- pentru subarborii drept ai celulelor impare $A_l = 1$, $B_l = 1$ $(l = i, j)$
- pentru subarborii stâng ai celulelor impare $A_l = 0$, $B_l = 1$ $(l = j+1, k)$.

T_{8+13} : aplică $C_{-1} = 1$ având ca şi valori ale intrărilor primare

- pentru subarborii drept ai celulelor impare $A_l = 0$, $B_l = 0$ $(l = i, j)$
- pentru subarborii stâng ai celulelor impare $A_l = 0$, $B_l = 1$ $(l = j+1, k)$.
- pentru subarborii drept ai celulelor pare $A_l = 1$, $B_l = 1$ $(l = i, j)$

BUPT

- pentru subarborii stâng ai celulelor pare $A_l = 0$, $B_l = 1$ $(l = j+1, k)$.
- testarea liniei 16 se realizează prin aplicarea în paralel la nivelul tuturor grupărilor de $(k-i+1)$ ranguri, ce se constituie ca şi celule ale dezvoltărilor ulterioare, a testului:

T_{8+14} : aplică $C_{-1} = 1$ având ca şi valori ale intrările primare pentru subarborii drept ai celulelor: $A_l = 1$, $B_l = 1$ $(l = i, j)$ şi pentru subarborii stâng $A_l = 0$, $B_l = 1$ $(l = j+1, k)$.

- T_2, sau T_3 este cel ce realizează testarea liniei 17 din Tabel 2. 3
- T_4, sau T_5 este cel ce realizează testarea liniei 18 din Tabel 2. 3

3. Atâta timp cât dublul dimensiunii celulei arbore obţinută prin dezvoltarea pasului 2 (şi pentru care au fost determinate testele necesare testării ei complete în condiţiile de C-testabilitate ce au fost impuse) nu a atins dimensiunea sumatorului PyCLA pentru care se doreşte determinarea setului complet de teste pentru detecţie în raport cu defectele de blocare singulară la 1 sau 0, se repetă pasul 2. Trecerea la pasul 4 se face după ce au fost identificate un număr de $\left(8 + 14 \cdot \left(\log_2 \frac{n}{2} - 1\right)\right)$ teste.
4. Ultima celulă identificată este dublată în vederea obţinerii subarborilor stâng şi drept ai arborelui binar ce se constituie prin adăugarea celulei de tip B la rădăcina celor doi arbori. Testarea completă a ultimei celule de tip B adăugată presupune adăugarea a încă 14 teste la cele deja determinate, cu observaţia că la determinarea acestora nu mai este necesară respectarea condiţiei de refacere spaţială a valorii semnalelor de stare, întrucât nu se mai fac alte dezvoltări de celule.

În concluzie, prin aplicarea în manieră recursivă a proprietăţii de C-testabilitate celulelor dezvoltate piramidal prin repetare spaţială a două celule compuse pentru care s-au determinat testele implicate de testarea lor completă şi conectarea lor la bază printr-o celulă de tip B pentru testarea căreia se identifică testele necesare, s-a demonstrat că testarea completă pentru detecţia defectelor de blocare singulară la 1 sau la 0 a unui sumator PyCLA de $n = 2^k$, $(k \in N)$ ranguri se realizează cu un număr de $8 + 14 \cdot \log_2 \frac{n}{2} = 8 + 14 \cdot (k-1)$ teste.

În Anaxa A Fig.A.6 se prezintă în pseudocod un algoritm ce permite obţinerea automată a testelor pentru sumatoare PyCLA.
In [POPE98E] este dat un algoritm ce permite generarea testelor pentru un sumator PyCLA hibrid, ce utilizează într-o formă combinată tehnicile *lookahead* şi *ripple*.

2.7 Eficientizarea testării prin C–testabilitate la structurile de însumare cu propagare cu omitere a transportului

2.7.1 Testarea sumatorului CSkA cu un nivel de omitere

Schema acestui sumator (Fig.1.5.) este obţinută prin interconectarea celulelor RCA (care sunt ILA-uri C-testabile), împreună cu o logică externă de obţinere a semnalului *block carry-out*, într-o structură de tip ILA. Modelul de defect utilizat presupune că celula defectă rămâne circuit combinaţional.

BUPT

ILA considerat este unilateral, unidimensional fiind compus din celule combinaţionale al [illegible] m..el s..v.nţial p..zină m intrări primare $A_i = 0$ şi o intrare de stare. *Pattern*-urile de test pentru testarea completă a acestui tip de sumator se obţin pe baza *pattern*-urilor de testare pseudoexhaustivă a ILA-urilor ce formează celulele RCA, la care se adaugă, *attern*-urile de test necesare testării complete a logicii de formare a semnalului *block carry-out* (Fig.2. 19).

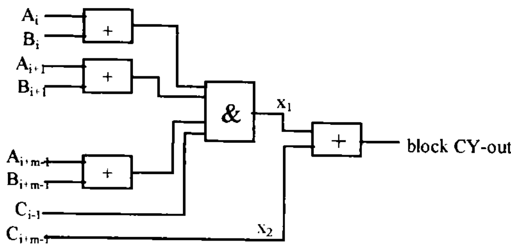


Fig.2. 19

Pe baza celor mai sus menţionate şi a proprietăţii de C-testabilitate, enunţ şi demonstrez următoarea teoremă originală:

Teorema 2.8 Prin adăugarea unei porţi suplimentare la nivelul fiecărui bloc (poarta haşurată în Fig.2. 21) a unui sumator CSkA la care dimensiunea blocurilor este m, se obţine un sumator CSkA C-testabil, testarea completă pentru detecţia defectelor de blocare singulară la 1 sau 0 a acestuia se face prin aplicarea unui număr de pattern-uri de test dat de relaţia:

$$N_{TCSkA} = 10 + 2 \cdot m \quad (2.18)$$

independent de numărul blocurilor înlănţuite prin logica *block carry-out*.

Demonstraţie: Testarea completă a celulelor RCA ce formează blocurile sumatorului se realizează cu cele 8 teste din Tabel 2. 1. Acestor teste li se adaugă cele necesare testării complete a logicii *block carry-out*, care sunt determinate şi ele în condiţiile impuse ale C-testabilităţii.

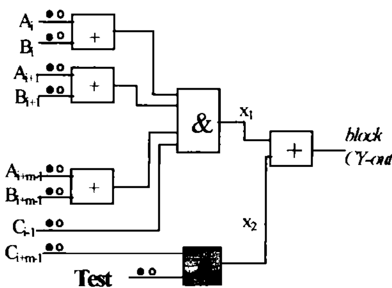


Fig.2. 21

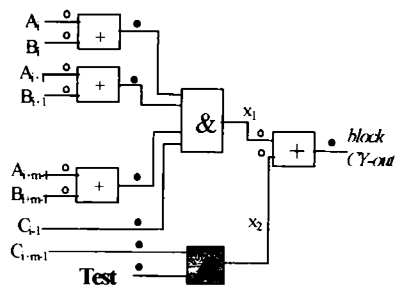


Fig.2. 20

Ecuaţiile de formare a semnalului *block carry-out* este:

$$block\,CY-out = T_i \cdot T_{i+1} \cdot \ldots \cdot T_{i+m-1} \cdot T_{i-1} \cdot C_{i+m-1}$$

unde $(A_i + B_i) \cdot (A_{i+1} + B_{i+1}) \cdot \ldots \cdot (A_{i+m-1} + B_{i+m-1}) \overset{D}{=} P_i \cdot P_{i+1} \cdot \ldots \cdot P_{i+m-1}$

Aplicând concluziile din Teorema 1.1. circuitului combinaţional ce formează logica CY-*skip* a blocurilor, obţinem că testarea lui completă pentru detecţia defectelor singulare de blocare la 1 sau 0 se realizează cu setul de teste ce detectează defectele

BUPT

figurate în Fig.2. 21. Numărul acestor teste poate fi redus prin aplicarea tehnicilor de *fault collapsing*, obținându-se ca și defecte ce trebuiesc urmărite cele din Fig.2. 20.

Continuăm prin identificarea testelor pentru detecția defectelor din Fig.2. 20. Considerăm primul termen al sumei ca fiind x_1, iar al doilea ca fiind x_2.

- Blocarea la 1 a semnalului *Block CY-out* este detectată de testul T_1, indiferent de starea semnalului *Test*
- Blocarea la 0 a lui x_1 este detectată de testele T_4 sau T_5 în condițiile în care semnalul *Test* este 0.
- Blocarea la 0 a lui x_2 este verificată de testele T_7 sau T_8 (în funcție de paritatea lui m) când $Test = 1$
- Blocarea la 0 a unui semnal A_i se detectează cu testul T_5 și $Test = 0$
- Blocarea la 0 a unui semnal B_i se detectează cu testul T_4 și $Test = 0$
- Blocarea la 1 a ieșirii porții OR ce formează semnalul $A_i + B_i$ se detectează prin aplicarea a două teste impuse de condiția de C-testabilitate a sumatorului, ambele teste au $C_{-1}=1$ și aplică porții rangului i valorile $A_i = B_i = 0$, iar pentru porțile ce aparțin rangurilor $j \neq i$, testele aplică următorii stimuli:
  1. Testul 1: pentru blocurile pare $A_j = 0$ și $B_j = 1 (j = 0, m-1$, și $j \neq i)$; valori ce se aplică și blocurilor impare, mai puțin ultimul rang, care va avea atât A_j cât și B_j egale cu 1, generându-se astfel transport inițial blocului par următor; testul realizează testarea porții i a blocurilor pare
  2. Testul 2: pentru blocurile impare $A_j = 0$ și $B_j = 1$, $(j = 0, m-1$, și $j \neq i)$; valori ce se aplică și blocurilor pare, mai puțin ultimul rang, care va avea atât A_j cât și B_j egale cu 1, generându-se astfel transport inițial blocului impar următor; testul realizează testarea porții i a blocurilor impare

Numărul total al acestor teste este $(2 \cdot m)$.

- Blocarea la 1 a lui C_{i+m-1} se detectează prin aplicarea testelor T_7 sau T_8, în funcție de paritatea lui m, când $Test = 1$.
- Aplicarea testelor T_7 sau T_8 (în funcție de paritatea lui m) împreună cu semnalul $Test = 1$, permit detecția defectului de blocare la 0 a semnalului $C_{i+m-1} \cdot Test$

În concluzie, pentru testarea completă pentru detecția defectelor de blocare singulară la 1 sau 0 a sumatorului CSkA cu un nivel de omitere, celor 8 teste necesare pentru testarea unităților RCA (ce trebuiesc aplicate cu *Test*=1) ce aparțin blocurilor sumatorului, li se adaugă cele 2 teste formate din testul T_4 cu *Test*=0 și respectiv testul T_5 cu *Test*=0, precum și cele $(2 \cdot m)$ teste necesare testării la blocare la 1 a ieșirilor porților OR ce formează semnalele de transfer, obținându-se în felul acesta un set complet de teste pentru detecția defectelor de blocare singulară la 1 sau 0 a acestui sumator ce numără $2 \cdot m + 10$ teste, indiferent de numărul blocurilor înlănțuite.

In Anexa A Fig.A7 am prezentat în pseudocod algoritmul de obținere automată al acestor teste pentru un sumator CSkA cu n ranguri organizat în blocuri de dimensiune constantă: m.

BUPT

Corolar 2.8.1. Testarea completă pentru detecția defectelor de blocare singulară la 1 sau la 0 a sumatoarelor CSkA cu un nivel de omitere la care dimensiunea blocurilor este variabilă, se realizează cu un set de teste (unde m_i reprezintă dimensiunea blocului i al sumatorului) al căror număr este independent de numărul rangurilor acestuia, și este dat de următoarea relație:

$$N_{TCSk1} = 10 + 2 \cdot \max(m_i)\ (i = 1, nr.blocuri) \qquad (2.19)$$

Generarea celor $2 \cdot \max(m_i)$, unde m_i reprezintă dimensiunea blocului i al sumatorului se face conform următoarei secvențe (dată în pseudocod):

/ pentru blocuri de dimensiune variabilă m_i, $(i = 1, nr.blocuri)$ */*

For $j = 1, \max(m_i)$ ** unde m_i reprezintă dimensiunea blocului i **

Begin

Asignează ca intrări pentru rangul j al tuturor blocurilor, valorile: $A_j = 0$, $B_j = 0$

Pentru toate blocurile k , $(k = 1, nr.blocuri)$ *ale sumatorului* **Do**

For $(i = 1, m_k), i \neq j$ *asignează pentru rangul i valorile* $A_i = 0$, $B_i = 1$

Asign. pentru ultimul rang al blocurilor pare $(l, l \in 1, nr.blocuri)$ *valorile:* $A_{ml} = 1$, $B_{ml} = 1$

aplică testul obținut

Inversează valorile de intrare ale ultimului rang al blocurilor pare cu al celor impare

aplică testul obținut

end

2.7.2 Testarea sumatorului CSkA cu două niveluri de omitere

Schema acestui sumator (Fig.1.6.) este obținută prin interconectarea blocurilor *CY-skip* (alcătuite din sumatoare RCA ce sunt C-testabile), împreună cu o logică externă de obținere a semnalului *Supeblock carry-out*, într-o structură de tip ILA. Modelul de defect utilizat presupune că celula defectă rămâne circuit combinațional.

Presupunând că avem o dimensiune m pentru blocuri și o dimensiune M pentru supeblocuri am enunțat și demonstrat următoarea teoremă originală referitoare la testarea acestor structuri de însumare:

Teoremă 2.9. Prin adăugarea unei porții suplimentare la nivelul fiecărui supebloc (poarta hașurată în Fig.2. 24) a unui sumator CskA cu două niveluri de omitere, la care dimensiunea blocurilor este m, și numărul blocurilor la nivelul supeblocurilor este M, se obține un sumator C-testabil; testarea completă pentru detecția defectelor de blocare singulară la 1 sau 0 a acestuia se face prin aplicarea unui număr de pattern-uri de test independent de numărul supeblocurilor înlănțuite prin logica supeblock *carry-out* și egal cu:

$$N_{TCSkAmM}(8 + 2m + M + 3) \qquad (2.20)$$

<u>Demonstrație:</u> *ILA* ce corespunde structurii sumatorului CSkA pe două niveluri este unilateral, unidimensional fiind compus din celule combinaționale (supeblocul) al căror model secvențial prezintă $M \cdot m$ intrări primare (A_i, B_i) și o intrare de stare.

Pattern-urile de test pentru testarea completă a acestui tip de sumator se obțin pe baza *pattern*-urilor de testare pseudoexhaustivă a ILA-urilor ce formează blocurilor

BUPT

sumatorului, la care se adaugă, *pattern*-urile de test necesare testării complete a logicii de formare a semnalului *supeblock carry-out* (Fig.2. 22); ele sunt determinate în condițiile impuse de C-testabilitate.

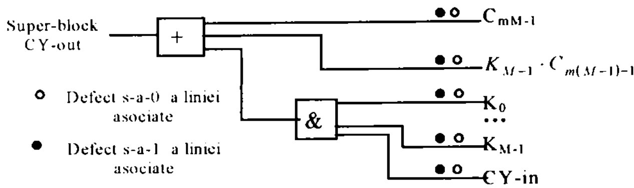


Fig.2. 22

În Fig.2. 22. sunt figurate și defectele ce trebuiesc urmărite la nivelul acestui circuit pentru obținerea unui set complet de test pentru detecția defectelor sale. De notat faptul că numărul acestor defecte urmărite a fost redus prin aplicarea tehnicilor de *fault collapsing*, obținându-se situația din Fig.2. 23.

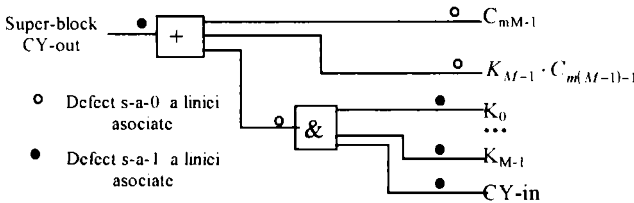


Fig.2. 23

Vom începe cu o analiză a defectelor, detectate la nivelul arborelui din Fig.2. 23, de testele ce au fost deja stabilite pentru testarea completă a ILA-ului format prin interconectarea blocurilor ce formează supeblocul. Pentru defectele neacoperite de aceste teste se vor determina teste suplimentare care să le acopere.

Analiza defectelor determinate de *pattern*-urile de test deja stabilite este prezentată sintetic în tabelul de mai jos:

	C_{mM-1}	$K_{M-1} \cdot C_{m(M-1)-1}$	K_0	...	K_{M-1}	*CY -in*	Defecte detectate
T_1	0	0	0	...	0	0	Supeblock *CY-out* s-a-1
T_2, T_3	0	0	1		1	0	Supeblock *CY-out* s-a-1 *CY-in* s-a-1
T_4, T_5	1	1	1	...	1	1	Supeblock *CY-out* s-a-0
T_6	1	1	1	...	1	1	Supeblock *CY-out* s-a-0
T_7 T_8	0/1 1/0	0 0	0 0		0 0	0 1	Unul din cele două teste, în funcție de paritatea lui Mm detectează defectul C_{mM-1} s-a-0, iar celălalt defectul C_{mM-1} s-a-1
T_9 T_{10}	1/0 0/1	0 0	0 0		0 0	1 1	Unul din cele două detectează C_{mM-1} s-a-0 iar celălalt defectul C_{mM-1} s-a-1
...				...			
$T_{10 \cdot 2m-1}$ $T_{10 \cdot 2m}$	1/0 0/1	0 0	0 0		0 0	1 1	Unul din cele două detectează C_{mM-1} s-a-0 iar celălalt defectul C_{mM-1} s-a-1

Tabel 2. 4

BUPT

În continuare sunt determinate testele suplimentare ce sunt necesare asigurării acoperirii defectelor rămase neacoperite prin aplicarea testelor stabilite la testarea sumatorului CSkA cu un nivel de omitere.

- Pentru detectarea defectului $K_{M-1} \cdot C_{m(M-1)-1}$ s-a-0 (care nu poate fi detectat decât prin adăugarea unei logici adiționale formată din poarta hașurată din Fig.2. 24. și prin activarea semnalului *Test=0* din figură), se poate utiliza la nivelul fiecărui supebloc testul:

  $T_{10+2m+1}$: *CY-in*= 1 și (00 00 .. 00) (11 .. 11) ... (11 .. 11),

  unde în paranteze au fost grupați operanzii sub forma A_iB_i, $i=\overline{0,m-1}$ ai blocurilor 0, 1 și *M-1*.

  Testul a fost determinat pe baza considerentului de asigurare a repetabilității stărilor inițiale pentru toate supeblocurile ce se constituie ca și celule ale ILA ce formează sumatorul CSkA pe două niveluri.

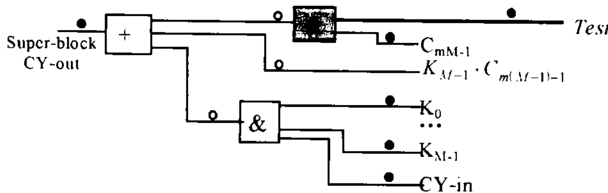


Fig.2. 24

- Detectarea blocării la 1 a semnalelor K_0, .., K_{M-1} se realizează prin aplicarea a două teste impuse de condiția de C-testabilitate a sumatorului, ambele teste au $C_{-1}=1$ și aplică blocului i valorile $A_k = B_k = 0$ $\left(k=\overline{0,m-1}\right)$, iar pentru blocurile cu numărul $j \neq i$, testele aplică următorii stimuli:
  1. Testul 1: pentru supeblocurile pare, la nivelul blocurilor $j \neq i$, se aplică $A_l = 0$ și $B_l = 1$ $(l=\overline{0,m-1})$ valori ce se aplică și blocurilor supeblocurilor impare, mai puțin ultimul bloc, care va avea atât A_l cât și B_l egale cu 1, generându-se astfel transport inițial supeblocului par următor; testul realizează testarea blocului i al supeblocurilor pare
  2. Testul 2: pentru supeblocurile impare la nivelul blocurilor $j \neq i$, se aplică $A_l = 0$ și $B_l = 1$, $(l=\overline{0,m-1})$ valori ce se aplică și blocurilor supeblocurilor pare, mai puțin ultimul rang, care va avea atât A_l cât și B_l egale cu 1, generându-se astfel transport inițial supeblocului impar următor; testul realizează testarea blocului i al supeblocurilor impare

Numărul total al acestor teste este $(2 \cdot M)$. Starea semnalului *Test* este indiferentă.

- Blocarea la 1 a lui C_{mM-1} este detectată prin aplicarea testului T_1 (de exemplu) când *Test=1*.
- Aplicarea testelor T_4 sau T_5 (în funcție de paritatea lui $(m \cdot M)$) împreună cu semnalul *Test 0*, permit detecția defectului de blocare la 1 a semnalului *Test*
- Defectul de blocare la 0 a ieșirii porții AND ce generează semnalul: $K_0 \cdot .. \cdot K_{M-1} \cdot C_{mM-1}$ este detectat de testele T_4, sau T_5, sau T_6 (având *Test=1*), ce

BUPT

vor duce la obţinerea unor rezultate incorecte în prezenţa acestor defecte, datorită nefuncţionării la parametrii a logicii supebloc *CY-out*.

Deci, pentru testarea completă pentru detecţia defectelor de blocare singulară a sumatorului CSkA cu două niveluri de omitere, celor $(10+2m)$ teste necesare pentru testarea blocurilor CSkA cu un nivel de omitere (ce trebuiesc aplicate având starea semnalului *Test* conform cu cele prezentate în secţiunea referitoare la acesta) ce formează supeblocurile sumatorului, li se adaugă testul $T_{10+2m+1}$ (având *Test*=0) necesar testării blocării la 1 a lui $\left(K_{M-1} \cdot C'_{m(M-1)-1}\right)$, precum şi cele $(2 \cdot M)$ teste necesare testării la blocare la 1 a semnalelor K_i $(i = 0, M-1)$, obţinându-se în felul acesta un set complet de teste pentru detecţia defectelor de blocare singulară la 1 sau 0 a acestui sumator ce numără $(11+2 \cdot m+2 \cdot M)$ teste, indiferent de numărul blocurilor înlănţuite.

In Anexa A Fig.A.8. am prezentat în pseudocod algoritmul de obţinere automată al acestor teste pentru un sumator CSkA pe două niveluri ce are n ranguri, ale cărui supeblocuri grupează M blocuri, iar dimensiunea blocurilor este constantă şi egală cu m.

Corolar 2.9.1 Testarea completă pentru detecţia defectelor de blocare singulară la 1 sau la 0 a sumatoarelor CSkA cu două niveluri de omitere la care pe de o parte numărul blocurilor ce formează supeblocurile este variabil - şi anume supeblocul j are M_j blocuri -, iar pe de altă parte este variabil şi numărul de ranguri ce formează blocurile sumatorului - şi anume: blocul i are m_i ranguri -, se realizează cu un set de teste ce însumează:

$$N_{TCSk2mM} = 11 + 2 \cdot \max_{i=1}^{nr.totalblocuri} (m_i) + 2 \cdot \max_{j=1}^{nr.super-blocuri} (M_j) \text{ teste.} \quad (2.21)$$

În concluzie, pentru testarea completă a unui sumator CSkA cu structură regulată ce este organizat pe două niveluri, este necesar un număr de $(11+2 \cdot m+2 \cdot M)$ teste, valoare ce este constantă în raport cu dimensiunea, n, a operanzilor.

2.8 Eficientizarea testării prin C–testabilitate la structurile de însumare cu propagare specială a transportului

2.8.1 Testarea sumatorului LR-CSlA

Schema acestui sumator (Fig.1.24.) este similară celei a sumatorului RCLA, cu deosebirea că pentru această situaţie blocurile CLA sunt înlocuite cu blocuri sumatoare CSlA; prin urmare, schema sumatorului LR-CSlA este obţinută prin interconectarea serială a N $(N = \frac{n}{m})$ blocuri sumatoare CSlA. Fiecare bloc CSlA este format din:

- o unitate CY ce utilizează tehnica *lookahead*,
- o unitate de generare a sumei ce utilizează şi ea tehnica *lookahead*,
- o unitate de selecţie a sumei.

Considerăm că tehnica *lookahead* se foloseşte în forma ei pură, cu observaţia că rezultatele analizei ce urmează sunt aplicabile (prin particularizarea relaţiilor de calcul) şi pentru situaţia utilizării unor tehnici CLA combinate.

BUPT

Sumatorul în ansamblu are o structură de tip ILA ale cărui celule sunt constituite de sumatoarele CSIA, ce includ cele 3 blocuri mai sus amintite.
Modelul de defect utilizat presupune că celula defectă rămâne circuit combinațional.

ILA ce corespunde structurii sumatorului LR-CSIA este unilateral, unidimensional fiind compus din celule combinaționale (blocurile sumatorului) al căror model secvențial prezintă m intrări primare (A_i, B_i), $i = \overline{1,m}$ și o intrare de stare (*CY-in* a blocului respectiv).

Bazându-ne pe considerațiile făcute în secțiunea de analiză funcțională a structurilor de însumare, precum și pe relațiile de generare:

$$S_j^1 = (A_j + B_j) \oplus C_{j-1}^1 \quad j = 0,1,\ldots,n-1$$

$$S_j^0 = (A_j + B_j) \oplus C_{j-1}^0$$

și de selecție:

$$S_i = C_i \cdot S_i^1 + \overline{C_i} \cdot S_i^0$$

$$S_{i+1} = C_i \cdot S_{i+1}^1 + \overline{C_i} \cdot S_{i+1}^0$$

$$\ldots$$

$$S_{i+m-1} = C_i \cdot S_{i+m+1}^1 + \overline{C_i} \cdot S_{i+m-1}^0, \qquad i = \overline{1,N}$$

a biților sumă ce aparțin blocului i (implementate de unitatea de selecție a sumei) ce stau la baza implementării acestui tip de sumator (CY-urile din ecuațiile precedente fiind generate prin *lookahead*), concluzionăm prin a specifica că *pattern*-urile de test pentru testarea completă a acestui tip de sumator, ce reprezintă și el o structură C-testabilă, se pot obține pe baza *pattern*-urilor de testare completă a blocurilor sumatorului, care la rândul lor, se obțin prin reunirea *pattern*-urilor determinate de:

1. setul complet de teste, obținut prin metode deterministe (funcționale), ce permite detecția tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale logicii de anticipare a transporturilor la nivelul blocurilor sumatorului.
2. setul complet de teste, obținut prin metode deterministe (funcționale), ce permite detecția tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale logicii de formare a biților sumă
3. setul complet de teste, obținut prin metode deterministe (funcționale), ce permite detecția tuturor defectelor singulare de blocare la 0 sau 1 a liniilor de semnal ale logicii de selecție a biților sumă

În concluzie, setul complet de teste pentru detecția defectelor singulare de blocare la o valoare logică, a sumatorului LR-CSIA este format din:

- Testele necesare unei testări complete a logicii de anticipare a transporturilor la nivelul blocurilor, al căror număr este dependent de dimensiunea blocurilor *(m)*. Conform teorema 2.3 numărul acestora teste este $\frac{(m+2)\cdot(m+1)}{2} + 1$.
- Testele necesare detecției defectelor singulare de blocare la 0 sau 1 a liniilor de semnale ce aparțin logicii de formare a biților sumă, și care sunt incluse în setul de teste mai sus determinat
- Testele necesare detecției defectelor singulare de blocare la 0 sau 1 a liniilor de semnale ce aparțin logicii de selecție a biților sumă, care sunt și ele incluse în setul de teste mai sus determinat, mai puțin testul ce aplică la nivelul tuturor cifrelor binare ale operanzilor valorile: $A_i = 0, B_i = 0$, $(i = \overline{1,n})$

BUPT

Deci, numărul total de teste pentru detecţia defectelor de blocare singulară la 0 sau 1 a unui sumator LR-CSlA cu n ranguri, având dimensiunea blocurilor m, este constant în raport cu n şi are expresia:

$$N_{T\,LRCSlA} = \frac{(m+2)\cdot(m+1)}{2} + 2 \qquad (2.22)$$

2.9 Evaluare comparativă prin prisma indicatorului de performabilitate a structurilor de însumare

Pe baza expresiilor de performanţă operaţională determinate în cadrul secţiunilor 2.1 ÷ 2.4 se obţin datele din Tabel 2.5 care realizează compararea, sub acest aspect, a sumatoarelor de dimensiuni arbitrare

. De notat faptul că indicatorul performanţei de operare al structurilor de însumare a fost calculat în termenii numărului de întârzieri (τ) introduse prin parcurgerea pe calea de propagare a unei porţi, indiferent de tipul acesteia.

Rezultatele acestui calcul sunt doar orientative, întrucât:

- În funcţie de tehnologia utilizată în cadrul procesului de integrare, apar diferenţe în valorile întârzierilor introduse de propagarea semnalelor prin diferitele tipuri de porţi logice.
- În evaluările făcute nu s-a ţinut cont de restrângerile impuse de mărimile fan-out şi fan-in ale porţilor logice, care şi ele au valori în funcţie de tehnologia de integrare utilizată la implementarea structurilor respective

Pentru anumite cazuri, se poate întâmpla ca valorile teoretice optime ale anumitor parametrii (dimensiunea blocurilor de exemplu) să nu poată fi atinse întrucât ele nu rezultă ca şi valori întregi. Din acest motiv se impune ca şi criteriu de comparaţie valorile lui n folosite pentru implementarea structurilor de însumare.

Sumator	Timp (τ)
CLA pur	4
CSlA	6
PyA	$2\log_2 n + 1$
CdSumA	$2\log_2 n + 2$
SRCLA (m=4, M=4)	$\frac{n}{8} + $ [illegible]
(m=2, M=8)	$\frac{n}{8} + $ [illegible]
(m=4, M=8)	$\frac{n}{16} + $ [illegible]
CCA (media)	$2\log_2 n + 4$
SBCLA	[illegible] $\sqrt[3]{n} + 2$
BCLA	$4\sqrt{n} + 2$
CSkA pe un nivel (optimal)	$\approx 4\sqrt{n}$
PyCLA	$2\log_2 n$
RCLA (m=4)	$\frac{n}{2} + 1$
(m=8)	$\frac{n}{4} + 1$
RCA	$2n + 1$

Tabel 2.5

În continuare se punctează câteva observaţii generale ce trebuiesc luate în considerare la interpretarea datelor ce au fost tabelate.

- Sumatorul RCA pare a reprezenta un proiect de sumator slab. Dar, datorită structurii sale simple şi regulate, este foarte atractiv pentru implementările VLSI, în special în combinaţie cu tehnica *Manchester carry-chain*.
- Sumatoarele CSkA dau cele mai bune performanţe atunci când sunt utilizate blocuri de dimensiune neuniformă; dar, varierea dimensiunii blocurilor înseamnă pierderea regularităţii lor, deci, ele devin mai puţin pretabile pentru anumite implementări tehnologice. Ca şi alternativă de îmbunătăţire a performanţelor lor operaţionale este utilizarea blocurilor de dimensiuni egale în conjuncţie cu tehnica *Manchester carry-*

BUPT

chain, obținându-se atât regularitate, cât și performanțe îmbunătățite. Acest gen de implemetări s-au constituit ca și bază pentru multe implementări VLSI.

- Sumatorul CLA în formă pură are cele mai bune performanțe de viteză, dar el implică pentru realizarea lui fan-out-uri și fan-in-uri ce îl fac realizabil doar pentru dimensiuni de însumare mici. Pentru sumatoare mari, rangurile sumatorului trebuiesc grupate în blocuri și eventual supeblocuri ce permit combinarea tehnicii *CY-lookahead* cu alte tehnici de însumare. O versiune simplă de asemenea sumator este sumatorul RCLA, ce combină tehnca *CY-ripple* cu *CY-lookahead* (cu un nivel de blocuri) și dă performanțe rezonabile pentru sumatoare de dimensiuni mici și medii. Are o structură regulată, întrucât utilizează unități cu aceeași structură și interconectarea între blocuri este mică. Dacă la sumatorul RCLA se adaugă un nivel supebloc, se obține sumatorul SRCLA, care este un sumator foarte bun pentru însumarea cuvintelor de dimensiune mare. Dar, spre deosebire de sumatorul RCLA, sumatorul SRCLA reclamă cel puțin două tipuri de unități de bază și un număr de interconexiuni mult mai mare. Sumatoarele BCLA, SBCLA și ISBCLA sunt sumatoare ce au la bază același principiu.
- Pe baza Tabel 2. 5, se observă că sumatorul piramidal pare a fi un sumator foarte bun, în sensul că timpul lui de operare, este relativ mic. Totuși, datorită problemelor de fan-out și fan-in pe care le ridică, nu este aplicabil în formă pură decât pentru implementarea sumatoarelor de dimensiune mică. La implementarea sumatoarelor mari, tehnica de adunare piramidală se combină cu alte tehnici de însumare care să permită obținerea unor sumatoare mai practice, cu bune performanțe și fără probleme de fan-out și fan-in. Pe de altă parte, un alt avantaj al acestui sumator este dat de faptul că este pretabil la *pipelining* [TING93] [TOMA93] [KUMA94].
- De asemenea, sumatoarele CdSumA și CSIA au performanțe bune pentru întreaga plajă de dimensiune a sumatorelor, dar implică valori mari pentru fan-in (CSIA) și fan-out (CdSumA). Din nou, pentru reducerea acestor inconveniente se poate apela la o tehnică de proiectare hibridă. Pe de altă parte, ambele tipuri de sumatoare se pretează la *pipelining*.
- Sumatorul CCA are un timp de operare mediu bun și o structură regulată. Cu toate acestea, datorită faptului că performanțele lui în cazul cel mai defavorabil sunt similare cu cele ale sumatorului RCA, și a faptului că timpul lui de operare este în funcție de valoare operanzilor, acest tip de sumator este neutilizabil pentru multe implementări. Aceste inconveniente pot fi reduse prin utilizarea tehnicii *Manchester carry-chain*.

Întrucât în ultima perioadă a crescut interesul pentru realizarea mașinilor asincrone, s-ar putea ca pe viitor acest tip de sumator să-și găsescă o utilizare mai largă.

Pe baza expresiilor de calcul al numărului de teste necesar testării complete în raport cu defectele de blocare singulară a structurilor de însumare analizate (în contextul utilizării proprietații de C-testabilitate) și determinate în cadrul secțiunilor 2.5 ÷ 2.8, au fost completate datele din Tabel 2. 6 care realizează compararea sub acest aspect a sumatoarelor de dimensiuni arbitrare.

În continuare se punctează câteva observații generale ce se desprind pe baza datelor ce au fost tabelate.

- Structurile de însumare de tip RCA, respectiv CCA, sunt cele mai ușor testabile
- Testarea structurile de însumare de tip RCLA, respectiv CSIA se face cu același număr de teste, care este mai mic decât cel implicat de testarea structurilor de tip BCLA, și a celor organizate pe superblocuri

BUPT

- Structurile de însumare de tip CSkA necesită pentru testarea lor completă în raport cu defecţiunile de blocare singulară un număr de teste mai mare decât cel implicat de sumatoarele de tip CLA organizate pe blocuri, dar mai mic decât cel implicat de sumatoarele CLA organizate pe superblocuri.

	Sumator	Număr de teste
1.	CCA (media)	8
2.	RCA	8
3	CLA pur	$\frac{(m+1)\cdot(m+2)}{2}+1$
4	RCLA (m)	$\frac{(m+1)\cdot(m+2)}{2}+1$
5	CSIA (m)	$\frac{(m+1)\cdot(m+2)}{2}+1$
6	BCLA (m)	$\frac{(m+1)\cdot(m+2)}{2}+7$
7	CSkA pe un nivel	$(10+2\cdot m)$
8	CSkA multinivel	$(11+2\cdot m+2\cdot M)$
9	PyCLA	$8+14\cdot\log_2\frac{n}{2}$
10	SRCLA (m, M)	$\frac{M\cdot(M+3)}{2}+\frac{m\cdot(m+3)}{2}$
11	SBCLA (m, M)	$\frac{M\cdot(M+3)}{2}+\frac{m\cdot(m+3)}{2}+8$

Tabel 2. 6

Este important de notat faptul că pentru anumite cazuri, valorile teoretice optime ale anumitor parametrii (dimensiunea blocurilor de exemplu) se poate întâmpla să nu poată fi atinse întrucât ele nu rezultă ca şi valori întregi. Din acest motiv se impune ca şi criteriu de comparaţie valorile lui n folosite în implementările practice ale acestor structuri de însumare.
Pe de altă parte, datele din aceste tabele trebuiesc totuşi analizate şi în contextul implementării tehnologice a structurilor de însumare, întrucât la implementarea lor apar probleme legate de fan-out, fan-in, dimensiunea de integrare, etc; aceste probleme nu au fost considerate la momentul completării tabelelor.

Sumator	Performabilitate
CCA (media)	$8\cdot(2\cdot n-1)$
RCA	$16\cdot(\log_2 n+2)$
CLA pur	$4\cdot\left(\frac{(m+1)\cdot(m+2)}{2}+1\right)$
RCLA (m)	$2\cdot\left(\frac{n}{m}+1\right)\cdot\left(\frac{(m+1)\cdot(m+2)}{2}+1\right)$
CSIA (m)	$\frac{(m+1)\cdot(m+2)}{2}+1$
BCLA (m)	$2\cdot\left(m+\frac{n}{m}-1\right)\cdot\left(\frac{(m+1)\cdot(m+2)}{2}+7\right)$
CSkA pe un nivel	$\left(2\cdot\frac{n}{m}+4\cdot m-5\right)\cdot(10+2\cdot m)$
CSkA multinivel	$\left(4\cdot m+4\cdot M+\frac{2\cdot n}{m\cdot M}-12\right)\cdot(11+2\cdot m+2\cdot M)$
PyCLA	$2\cdot\log_2 n\cdot\left(8+14\cdot\log_2\frac{n}{2}\right)$
SRCLA (m, M)	$2\cdot\left(\frac{n}{m\cdot M}+3\right)\cdot\left(\frac{M\cdot(M+3)}{2}+\frac{m\cdot(m+3)}{2}\right)$
ISBCLA (m, M)	$2\cdot\left(\frac{n}{m\cdot M}+m-1\right)\cdot\left(\frac{M\cdot(M+3)}{2}+\frac{m\cdot(m+3)}{2}+8\right)$
SBCLA (m, M)	$2\cdot\left(\frac{n}{m\cdot M}+M+m-1\right)\cdot\left(\frac{M\cdot(M+3)}{2}+\frac{m\cdot(m+3)}{2}+8\right)$

Tabel 2. 7

In vederea unei comparări globale, la nivelul producătorilor, a diferitelor structuri de însumare, se defineşte indicatorul de ***performabilitate*** ca fiind dat de

BUPT

inversul produsului dintre indicatorul performanței lor de operare și numărul de teste necesar unei testări complete al acestora în raport cu detecția defectelor lor de blocare singulară la 1 sau 0.

$$Performabilitate = \frac{1}{(performnta) \cdot (nr.teste)} \tag{2.23}$$

Formulele de calcul ale acestui indicator pentru diferitele structuri de însumare analizate sub aspectul performanței și al numărului de teste sunt tabelate în Tabel 2. 7

2.10 Concluzii

Pe baza analizei funcționale a structurilor de însumare s-au elaboarat în cadrul acestui capitol relațiile de calcul pentru performanțele lor de operare în termenii numărului maxim de porți traversate de semnalele activate în cadrul procesului de însumare.

Pe de altă parte, prin exploatarea proprietății de repetabilitate spațială a structurilor de însumare și aplicând acestor structuri conceptul C-testabilității - realizând eventual (acolo unde este cazul) o reconfigurare în vederea realizării dezideratului C-testabilității – în combinație cu cel al unei testări minimale la nivelul celulor structurilor repetabile, s-a reușit, printr-o analiză minuțioasă a structurilor, obținerea seturilor minimale pentru testarea lor completă în raport cu defectele de blocare singulară la 1 sau 0

Pe baza concluziilor analizelor efectuate am enunțat și demonstrat în cadrul capitolului un număr de 8 teoreme și 2 corolare originale, ce au stat la baza elaborării a 9 algoritmi ce permit generarea automată a seturilor minimale de teste pentru testarea completă în raport cu defectele de blocare singulară a structurilor de însumare analizate.

Pentru caracterizarea globală din punctul de vedere al dependabilității a structurilor de însumare s-a introdus indicatorul de performabilitate pentru care s-au dat relațiile de calcul la nivelul fiecărei structuri de însumare.

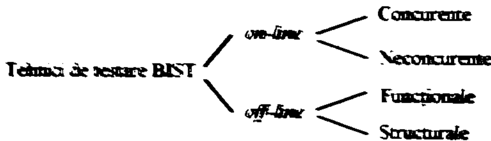
BUPT

3. Reconfigurări ale structurilor de însumare în scopul facilitării testării

Tehnicile *built-in-self-testing* (BIST) sunt tehnici de proiectare prin care anumite părți ale schemei sunt utilizate pentru testarea circuitului în ansamblu.
Aceste tehnici pot fi aplicate atât la nivelul producătorului de circuite (la toate nivelurile de împachetare tehnologică) cât și în exploatare. Utilizarea BIST în testarea la producător, permite ca prin sacrificarea unei suprafețe de integrare să se simplifice anumite aspecte ale testării diagnostice, și să se reducă costul implicat de echipamentele externe, în general sofisticate, ce sunt necesare tehnicilor de testare non-BIST.
De exemplu, în cazul µP I80386, un sacrificiu de *1,8%* din aria de integrare pentru implementarea BIST, permite testarea unor porțiuni de schemă foarte dificil de testat prin alte metode [GELS87].
Dacă se implementează BIST la nivelul chip-urilor în conjuncție cu *boundary-scan*, tehnica BIST poate fi dezvoltată mai departe la toate nivelurile ierarhice ale circuitului.
Avantajul conferit de aceste tehnici constă în faptul că testarea se face în timp real, putând fi folosite cu anumite extinderi și pentru testarea întârzierilor; totuși ele nu permit testarea parametrică. În cazul când tehnicile BIST sunt utilizate pentru testarea în exploatare, sunt reduse costurile implicate de echipamentele de test (devin mai simple) implicate pentru diagnoza defectelor și înlocuirea unităților defecte. Implicațiile sunt benefice pentru mentenabilitate și pentru costurile ciclurilor de viață ale circuitelor digitale complexe.

Built-in self-test exprimă capabilitatea unui circuit (chip sau placă) de a se autotesta. Termenul în sine înglobează conceptele de *built-in test* (BIT) și *self-test* (ST) și este legat de *built-in-test equipment* (BITE), care se referă la hardware-ul și software-ul ce sunt încorporate într-o unitate pentru a-i asigura capabilități DFT și BIST.

Mai jos este dată o clasificare sugestivă a acestor tehnici.



- În cazul tehnicii BIST *on-line*, testarea se realizează pe parcursul operării normale a circuitului, fără a fi necesară trecerea circuitului testat într-o stare specială de test, prin care să se blocheze funcționarea normală a acestuia.

  Tehnica BIST *on-line concurentă* este o formă de testare ce apare simultan cu funcționarea normală a circuitului. Aceasta se realizează de obicei prin utilizarea tehnicilor de codificare, de duplicare sau de comparare [CĂTU89].

  Tehnica BIST *on-line neconcurentă* realizează testarea prin executarea unor rutine software (sau firmware) de diagnoză, atunci când sistemul este în stare de așteptare (*idle*). Procesul de testare poate fi întrerupt la orice moment, continuându-se funcționarea normală.
- În cazul tehnicii BIST *off-line*, testarea se realizează în afara funcționării normale a sistemului, tehnica fiind aplicabilă atât la nivelul producătorului cât și în exploatare. Spre deosebire de tehnicile *on-line*, aceste tehnici nu detectează erorile în timp real, și ele presupun utilizarea unor generatoare de *pattern*-uri de test (*test-pattern*

BUPT

generators - TPG) locale (amplasate pe chip sau placă) şi a unor analizoare ale răspunsurilor de ieşire (*output response analyzers* - ORA).

BIST *off-line funcțional* se realizează prin executarea unui test bazat pe o descriere funcțională a circuitului ce se testează (*circuit under test* – CUT), şi, în general, utilizează un model funcțional de defecte. Acest test este implementat sub forma unui software de diagnoză sau a unui firmware.

Pe de altă parte, BIST *off-line structural* se realizează prin teste bazate pe descrierea structurală a CUT. Poate utiliza un model de defecte structural, acoperirea defectelor fiind bazată pe acoperirea defectelor structurale.

3.1 Reconfigurări BIST on line pentru testarea structurilor de însumare

În capitolul anterior s-a abordat problema detecției defectelor prin observarea răspunsurilor la teste a circuitelor testate. În acest capitol sunt abordate câteva procedee de proiectare ce simplifică diagnoza şi detecția defectelor, şi anume proiectarea specifică sistemelor autotestabile la care defectele pot fi detectate automat prin subcircuitul numit *checker*. Asemenea scheme implică utilizarea intrărilor în formă codificată.

În anumite situații, fără a se cunoaşte valoarea de răspuns aşteptată, doar pe baza urmăririi ieşirilor circuitului, se poate determina faptul că există un anumit defect *f* în circuit. Acest tip de testare ce se desfăşoară *on-line* este bazată pe verificarea anumitor proprietăți invariante ale ieşirilor circuitului.

În acest caz nu mai este necesară o testare explicită a defectului *f*, circuitul fiind numit *autotestabil (self-testing)* în raport cu defectul *f*.

Circuitele ce prezintă capabilități de autoverificare permanentă a bunei lor funcționări se numesc circuite autoverificabile (*self-checking*) (Fig.3. 1).

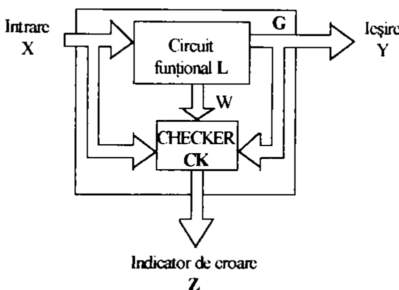


Fig.3. 1

Proiectarea un r [illegible] presupune utilizarea unui circuit suplimentar numit *checker*, care să genereze un semnal de eroare ori de câte ori ieşirile circuitului G semnalează prezența unei erori (dintr-un set specificat de erori) la nivelul lui G.

Prin *eroare* se înțelege prezența unui semnal a cărui valoare este diferită de valoarea lui în [illegible] funcționează corect.

Pentru proiectarea schemelor *self-checking*, acestea se consideră divizate în două blocuri interconectate ca în Fig.3. 1:

1. Blocul L al circuitului funcțional şi
2. Blocul CK al circuitului de verificare

Este preferabil ca atât circuitul funcțional cât şi checker-ul să fie proiectate într-o măsură cât mai mare autotestabile.

Considerăm pentru început că circuitul funcțional L din Fig.3. 1 este combinațional, el are p intrări şi q ieşiri, şi poate avea ca şi intrări toate cele 2^p combinații posibile ale intrărilor. Dacă la ieşirile circuitului pot apare toate cele 2^q combinații

BUPT

posibile ale ieşirilor, nu există nici o posibilitate de detecţie a defectelor din circuit doar pe baza observării valorilor sale de ieşire, fără a se cunoaşte valorile lui de intrare.
Dacă pe timpul funcţionării normale pot apărea doar $k < 2^q$ configuraţii de ieşire, apariţia oricărei alte configuraţii din cele $2^k - q$ rămase va indica prezenţa unei erori în circuit. Asemenea defecte ce duc la asemenea valori ilegale ale ieşirilor vor fi detectate de dispozitivele hardware numite *checker*-e.
Pentru un circuit ce are configuraţii de ieşire ce nu apar pe parcursul operării sale normale, vom numi configuraţiile de ieşire ce apar ca fiind valide (sau cuvinte de cod *(code words)*), iar configuraţiile ce nu apar vor fi numite ca fiind invalide (sau cuvinte noncod (*noncode words*)) Termenii se vor referi atât la codurile de intrare cât şi la cele de ieşire. In vederea definirii terminologiei de autocontrol ce se uzitează cu referire la circuitul funcţional L, facem următoarele consideraţii:

- Presupunem că circuitul combinaţional L produce un vector de ieşire Y(X,f), ce este o funcţie de vectorul de intrare X şi de defectul *f* ce aparţine unui set F de defecte specificate ale schemei.
- Absenţa defectelor indică defectul nul care este notat cu λ.
- Spaţiul de intrare Ω_X este dat de mulţimea tuturor celor 2^p vectori de intrare, iar sSpaţiul Ω_Y al ieşirilor este dat de mulţimea tuturor celor 2^q vectori de ieşire.
- In general, pe parcursul operării normale, circuitele logice vor recepţiona la intrare doar un subset al spaţiului lor de intrare; acest subset este numit spaţiul codurilor de intrare, fiind notat cu N. Spaţiul intrărilor noncod va fi dat de $\Omega_X - N$.
- Considerând că pe durata funcţionării normale la ieşirea circuitului se obţin doar cuvinte de cod, se poate defini un spaţiu al codurilor sale de ieşire, notat cu S, precum şi un spaţiu al ieşirilor noncod, care va fi $\Omega_Y - S$. In [BOND91] se dă un algoritm de calcul pentru S.

Pentru o anumită intrare X, apariţia unui defect poate duce sau nu la obţinerea unor erori la ieşirea circuitului. In situaţia apariţiei unei erori, aceasta va modifica ieşirea fie la un nou cuvânt de cod (eroare nedetectabilă) fie la un cuvânt noncod (eroare detectabilă).

Def.3.1. Un circuit L se numeşte ***fault-secure*** (FS) pentru un set de intrări N şi un set de defecţiuni F, dacă pentru orice intrare $X \in N$ şi pentru orice defect $f \in F$, avem $Y(X,f) = Y(X,\lambda)$ sau $Y(X,f) \notin S$.
Def.3.2. Un circuit L se numeşte ***self-testing*** (ST) pentru un set de intrări N şi un set de defecţiuni F, dacă pentru orice defect $f \in F$ există o intrare $X \in N$ astfel încât $Y(X,F) \notin S$.
Def.3.3. Un circuit se numeşte ***totally self-checking*** (TSC) dacă el este atât *self-testing* cât şi *fault-secure* pentru orice defecţiune singulară $f \in F$.

Iniţial definiţia circuitelor TSC a fost dată pentru circuitele combinaţionale, iar ulterior, ea a fost extinsă acoperind atât circuitele secvenţiale, cât şi sistemele în ansamblu. În [HATE91] este prezentată o metodă de proiectare a circuitelor TSC.
In general, întrucât nu sunt suficiente intrări în N pentru detectarea tuturor defectelor din setul F, condiţia ST este destul de dificil de îndeplinit perfect.

O secvenţă de defecte $< f_1, f_2, ... f_n >$, unde $f_i \in F$, $1 \le i \le n$, este defintă ca reprezentând evenimentul când apare f_1, urmat de apariţia lui f_2 şi aşa mai departe, până la apariţia lui f_n.

BUPT

Se presupune că odată ce apare un defect de blocare a unei linii la 0 sau la 1 logic, aceasta va rămâne blocată la această valoare logică. De asemenea se presupune că defectele apar pe rând, şi că între momentele de apariţie ale defectelor este timp suficient pentru baleierea tuturor combinaţiilor de intrare.

Def.3.4 Dacă circuitul L asigură cuvinte de cod corecte la ieşire pentru o secvenţă de mai puţin de *m* defecte (*m-1* defecte acumulate) $(2 \le m \le n)$ din F, şi, pentru toţi $X \in N$. Aceasta înseamnă că:

$$Y(X,< f_1, f_2,..., f_{m-1} >) = Y(X,\lambda).$$

De asemenea, dacă pentru o secvenţă de *m* defecte $< f_1, f_2,... f_m >$ există un $X \in N$ astfel încât

$$Y(X,< f_1, f_2,..., f_m >) \notin S.$$

Atunci se spune că L este ***strongly fault secure*** (SFS) pentru $< f_1, f_2,... f_m >$.

Un circuit este SFS pentru $F = \{f_1, f_2,..., f_m\}$ dacă el este SFS pentru toate secvenţele de defecte din F.

Toate circuitele SFS satisfac condiţiile TSC, ele constituindu-se ca şi cea mai largă clasă de circuite ce satisfac condiţiile TSC.

Definim perechea (X_a, X_b), unde $X_a \in N$, şi $X_b \in \Omega_X - N$ (X_b reprezintă valoarea eronată a intrării pe care o recepţionează L în locul lui X_a); adică $X_b = X_a + E$, unde $X_a \in N$, $X_b \in \Omega_X - N$ şi E exprimă o eroare la intrare. Se presupune că circuitul L funcţionează corect, şi că X_a şi X_b sunt complet dependente de blocurile de circuite ce preced L.

Def.3.5 Un circuit L cu spaţiul codurilor de ieşire S este ***error secure*** (ES) pentru spaţiul intrărilor noncod, $\Omega_X - N$, dacă pentru fiecare intrare $X_b \in \Omega_X - N$, unde $X_b = X_a + E$, $X_a \in N$, $E \ne 0$, avem:

$$Y(X_b,\lambda) \notin S \text{ sau } Y(X_b,\lambda) = Y(X_a,\lambda).$$

Def.3.6 Un circuit L cu spaţiul ieşirilor de cod S este ***code-disjoint*** (CD) pentru spaţiul intrărilor noncod $\Omega_X - N$ dacă pentru orice intrare X din $\Omega_X - N$ avem

$$Y(X,\lambda) \notin S$$

Un circuit CD este şi ES, reciproca nefind adevărată.

Considerăm situaţia când circuitul L are secvenţa de defecte $< f_1, f_2,... f_n >$, unde $f_i \in N$, $1 \le i \le n$, valorile sale de intrare neaparţinând spaţiului codurilor de intrare.

In [NICO84] se dau următoarele definiţii:

Def.3.7 Dacă înaintea apariţiei oricărui defect, circuitul L este CD şi dacă circuitul L pentru o secvenţă de mai puţin de *m* $(2 \le m \le n)$ defecte din F şi pentru toate intrările noncod $X \in \Omega_X - N$ se comportă astfel încât

$$Y(X,< f_1, f_2,..., f_{m-1} >) \notin S.$$

In condiţiile în care pentru o secvenţă $< f_1, f_2,... f_m >$, circuitul L este ST. Adică:

$$\exists X \in N \text{ astfel încât } Y(X,< f_1, f_2,..., f_m >) \notin S.$$

Atunci L se spune că este ***strongly code disjoint*** (SCD) pentru $< f_1, f_2,... f_m >$.

BUPT

Def.3.8 Circuitul L este SCD pentru $F = \{f_1, f_2, ..., f_m, ..., f_n\}$, dacă el este SCD pentru toate secvențele de defecte din F.
La baza autocontrolului se află redondanța informațională realizată prin codificarea informației.

În general codurile se clasifică în termenii abilității lor de detecție sau corecție a claselor de erori ce afectează un anumit număr de biți ai cuvintelor.

3.1.1 Coduri detectoare /corectoare de erori

Un cod este *e-detector* de erori, dacă el poate detecta orice eroare ce afectează cel mult e biți. Orice asemenea eroare nu va transforma un cuvânt de cod într-un alt cuvânt de cod.

În mod similar, un cod este *e-corector* de erori, dacă el poate corecta orice eroare ce afectează cel mult e biți.
Aceasta implică că pentru oricare două erori e_1, e_2, $(e_1, e_2 < e)$, ce afectează cuvintele w_1 și w_2, se obțin cuvinte de cod diferite [NIKO91].

Distanța Hamming d a unui cod este dată de minimul numărului de biți prin care diferă oricare două cuvinte ale codului. Tabel 3. 1 arată cum poate fi exprimată capacitatea de detecție/corecție a unui cod în termenii lui d.

În general, pentru a avea cuvintele de ieșire ale unui circuit constituite ca și coduri cu capabilități de detecție/corecție a erorilor, este necesar ca schema circuitului să genereze ieșiri adiționale numite biți de control (*check bits*). Cel mai simplu cod de acest gen este codul de verificare a parității. Acestui cod îi corespunde d 2; codul are un singur bit de control, independent de numărul de ieșiri ale circuitului original. Există două tipuri de coduri de verificare a parității, și anume:

- pentru paritate pară
- pentru paritate impară

Pentru codul de paritate pară bitul de control este calculat astfel ca pentru fiecare cuvânt de cod numărul biților rgali cu 1 şă fie par. Similar, pentru codul de paritate impară bitul de control este calculat astfel ca pentru fiecare cuvânt de cod numărul unităților de biți să fie impar.
Prin urmare, orice eroare ce afectează un singur bit va afecta paritatea unităților binare ale cuvintelor de ieșire, și astfel ea va putea fi detectată. De notat că într-un circuit arbitrar o defecțiune singulară poate, datorită prezenței *fanout*-urilor în circuit, să afecteze mai mult de o poziție binară la nivelul cuvintelor de ieșire. Prin urmare, proiectarea schemelor circuitelor la care detecția erorilor este bazată pe un cod cu capabilități limitate de detecție a erorilor, trebuie să se facă cu multă atenție.

d	Capabilitatea
1	niciuna
2	detecție - 1 eroare; corecție - 0 erori
3	detecție - 2 erori; corecție - 1 erori
•••	
e+1	detecție - e erori; corecție - $\left\lceil \frac{e}{2} \right\rceil$ erori
•••	
2e+1	detecție - 2e erori; corecție -e erori

Tabel 3. 1

Biții de control implicați de un anumit cod sunt de fapt redundanți întrucât ei sunt necesari doar pentru detecția erorilor. Ceilalți biți ai cuvântului sunt numiți *biți de informație*.

BUPT

Se poate defini o clasă generalizată a codurilor de verificare a parității ce au capabilități mărite de detecție/corecție a erorilor.
Un cod de detecție a erorilor singulare pentru q biți de informație reclamă c biți de control, unde $2^c \geq q+c+1$.
Cei c biți de control împreună cu cei q biți de informație formează un cuvânt de cod cu $(c+q)$ biți: $b_{c+q}, \dots, b_2, b_1$. În codul Hamming convențional biții de ontrol ai parității apar pe pozițiile b_{2^i}, $0 \leq i \leq c-1$. Valorile acestor biți de control sunt definite de ecuațiile de verificare a parității.
Fie p_i setul de întregi a căror reprezentare binară are valoarea 1 pe poziția b_i (adică $p_i = \{l / b_i(l) = 1\}$, unde $b_i(n)$ indică valoarea bitului i a reprezentării binare a lui n. Astfel, valorile celor c biți de control sunt definite de cele c ecuații de paritate și control a căror formă este:

$$\sum_{k \in p_i}{}^{\oplus} b_k = 0, \quad i = 1,\dots,c$$

unde suma este modulo 2.
O eroare a bitului b_i va duce la obținerea unei parități necorespunzătoare exact în ecuațiile pentru care i este în p_i Deci, pe baza acestor c ecuații de verificare a parității (care permit calcularea sindromului ca diferență între valorile corecte și cele eronate ale răspunsului), se poate face corecția bitului respectiv.

Utilizarea acestor coduri permite proiectarea hardware-ului astfel ca anumite clase de erori din circuit să poată fi detectate și corectate automat pe baza utilizării unui checker hardware, așa cum se arată în Fig.3. 2.b. De observat că erorile ce apar la nivelul checker-ului se poate întâmpla să nu fie detectate.

Au fost dezvoltate multe coduri ce pot fi utilizate în proiectarea unor asemenea circuite autotestabile. Alegerea codului folosit se face în funcție de tipul circuitului. De exemplu, pentru transmisia pe magistrale de date este adecvată alegerea codurilor de verificare a parității. Totuși, pentru alte tipuri de funcții este de dorit să se utilizeze coduri pentru care biții de control ai rezultatului pot fi determinați pe baza biților de control ai operanzilor.

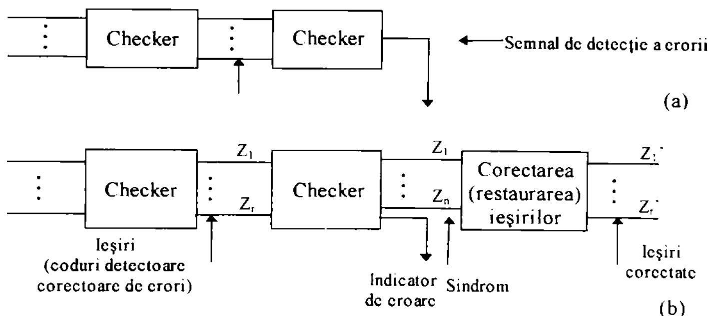


Fig.3. 2

Considerăm proiectarea unui sumator aritmetic ce să utilizeze codul de verificare a parității pare. Pentru cele două adunări ilustrate în Fig.3. 3, în ambele cazuri biții de control ai operanzilor sunt 1 și respectiv 0, în timp ce bitul de control al sumei $A+B_1$ este 0, iar cel al sumei $A+B_2$ este 1. Prin urmare, prin calcularea bitului de control al sumei după fiecare adunare nu se pot detecta erorile procesului de adunare.

BUPT

O altă clasă de coduri numite *coduri reziduale* au proprietatea că pentru operațiile aritmetice de adunare, scădere, și înmulțire, biții de control ai rezultatului pot fi determinați pe baza biților de control ai operanzilor [ZARO90]. Proprietatea este numită verificare independentă (*independent checking*). Au fost formulate mai multe tipuri diferite de coduri reziduale. Vom considera doar unul din această clasă.

A	=	0001	A	=	0001
B_1	=	0101	B_2	=	0011
$A+B_1$	=	0110	$A+B_2$	=	0100

$C(A)$ 1, $C(B_1)$ $C(B_2)$ 0, $C(A \cdot B_1)$ 0, $C(A \cdot B_2)$ 1,
unde $C(X)$ este bitul de control al lui X

Fig.3. 3

În codul considerat, biții de control se află pe cele p poziții mai puțin semnificative ale cuvântului. Biții de verificare definesc un număr binar C, iar biții de informație definesc un alt număr binar N. Valorile biților de control sunt definite astfel încât $C=(N) \bmod m$, unde m este un parametru al codului numit reziduu (*residue*), iar numărul biților de control este $p=\lceil \log_2 m \rceil$. In [STAN94] se dă o metodă de proiectare a generatorului de coduri reziduale a unităților aritmetice multi-operand.
În [ABRA96] avem dată demonstrația următoarei teoreme:

Teorema 3. 1

Fie $\{a_i\}$ un set de operanzi. Biții de control ai sumei sunt dați de $\left(\sum a_i\right) \bmod m$ iar biții de control ai produsului sunt dați de $\left(\prod a_i\right) \bmod m$. Atunci,

1. Biții de control ai sumei sunt egali cu suma modulo m a biților de control ai operanzilor
2. Biții de control ai produsului sunt egali cu produsul modulo m ai biților de control ai operanzilor

În Fig.3. 4 este ilustrată utilizarea codurilor reziduale [PIES87] pentru verificarea adunării într-un sistem. Pentru suma $A+B$ sunt calculați biții de control $C(A+B)$ și rezultatul calculului este comparat cu suma modulo m a biților de control: $\left(C(A)+C(B)\right) \bmod m$. Acest circuit va detecta orice eroare ce cauzează rezultat neegal în urma comparării.

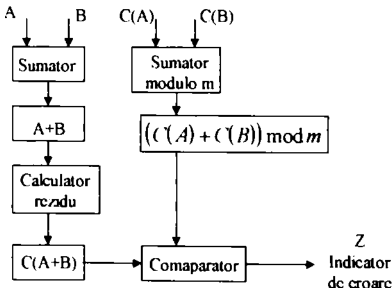


Fig.3. 4

T[illegible] determinată clasa de erori ce poate fi detectată prin aceste coduri reziduale.
Dacă un cod rezidual definește cuvinte cu $s=p+q$ biți, atunci se poate defini un *pattern* de eroare ca fiind un vector binar cu s biți în care dacă bitul i este eronat $e_i=1$, iar dacă bitul i este corect, $e_i=0$. Pentru un număr N având biții de control C, cu $C=(N) \bmod m$, o asemenea eroare va schimba N în N' și/sau C în C'. O astfel de eroare va fi detectată doar dacă se asigură $C' \neq N' \bmod m$.

Teorema 3. 2

Într-un cod rezidual cu m impar sunt detectate toate erorile singulare [ABRA96].

BUPT

Când m este impar, anumite erori singulare la nivel de bit pot fi nedistinctibile.
În schimb dacă m este par, anumite erori singulare pot fi chiar nedetectate.
În concluzie, pentru ca acest cod să detecteze erorile singulare trebuie ca parametrul m să fie impar. Pe măsură ce creşte valoarea lui m, creşte şi numărul biţilor de control, iar capabilitatea de detecţie a erorilor, respectiv de corecţie a acestora are o variaţie complexă în funcţie de valorile specifice ale lui m.

3.1.2 Erori multiple la nivelul biţilor

Pentru codurile detectoare/corectoare de erori este importantă şi capabilitatea acestora în raport cu erorile ce afectează mai mulţi biţi [BOSE91] [BASS94A]. În cazul codurilor de verificare a parităţii, acestea vor detecta orice erori ce afectează un număr impar de biţi.
Este important de notat faptul că în multe tehnologii cele mai probabile erori multiple la nivelul biţilor nu au un caracter aleator; ele au anumite proprietăţi speciale de genul:

- erorilor unidirecţionale – la care toţi biţii eronaţi au aceeaşi valoare [BOSE86] [BASS94B] [CHOW94]
- erorilor de biţi adiacenţi – toţi biţii afectaţi de o eroare sunt contigui

Pentru detectarea acestor tipuri de erori sunt utilizate în general următoarele coduri:

1. Codul k n (*k out of n*) ce constă în cuvinte de cod de n biţi în care numărul unităţilor este exact k. Aceste coduri detectează toate erorile unidirecţionale, întrucât aceste erori duc fie la creşterea, fie la descreşterea numărului unităţilor din cuvinte. Este un cod neseparabil, la care nu se pot separa biţii de control de cei de informaţie. Circuitele ce utilizează aceste coduri trebuie să realizeze calcule asupra întregului cuvânt, nu doar asupra părţii de informaţie a acestuia (care nu poate fi separată).
2. Codurile *Berger* sunt coduri separabile [STAN87] [CHUN94]. Pentru codificarea a I biţi de informaţie se folosesc $C = \lceil \log_2(I+1) \rceil$ biţi de control, formându-se cuvinte cu $n = I + C$ biţi. Cei C biţi de control definesc un număr binar ce reprezintă complementul Boolean al numărului biţilor de informaţie de valoare I. În Tabel 3. 2 este dat codul complet pentru $I = 3$ şi $C = 2$. Codul Berger detectează toate erorile unidirecţionale. Comparativ cu alte coduri ce detectează erorile unidirecţionale [POPE95B], codurile Berger sunt optimale din punct de vedere al numărului mic de biţi de control implicaţi pentru I biţi de informaţie. Totuşi, pentru realizarea a r cuvinte valide de cod pentru r având valori mari, codurile m n implică un număr de biţi de control mai mic.
3. În codul *rezidual modificat*, se definesc $m-1$ biţi astfel încât numărul total al unităţilor din cuvântul de cod să fie multiplu de parametru m. Acest cod, ce este o generalizare a codului de verificare a parităţii, detectează toate erorile unidirecţionale ce afectează un număr de biţi mai mic decât m. Pentru m 3, dacă biţii de informaţie au valoarea 110, biţii de control vor avea valorile 01 sau 10.

I_1	I_2	I_3	C_1	C_2
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0

Tabel 3. 2

4. Dacă se dă un set de $k \cdot n$ biţi dispuşi în k cuvinte de n biţi, prin adăugarea unui bit de paritate la fiecare cuvânt rezultă k cuvinte de $(n+1)$ biţi, pentru care pot fi detectate toate erorile singulare la nivel de bit, precum şi orice eroare ce afectează mai mulţi biţi atâta timp cât fiecare cuvânt are cel mult un bit eronat. Ca şi alternativă se poate defini un cuvânt

BUPT

de control C cu n biţi, în care pentru toţi i, $(1 \le i \le n)$, bitul i din C este un bit de verificare a parităţii pentru toţi biţii i ai celor k cuvinte. Se obţin astfel $(k+1)$ cuvinte de n biţi pentru care sunt detectate toate erorile singulare, toate erorile ce afectează mai mulţi biţi la nivelul unui cuvânt, precum şi toate erorile ce afectează mai mulţi biţi din cuvinte diferite. Această tehnică poate fi utilizată în conjuncţie cu codurile detectoare/corectoare de erori, obţinându-se astfel *multiple-bit correction.*

3.1.3 Problematica construirii checker-elor de erori

Detectoarele de erori sunt circuite care detectează şi semnalizează apariţia erorilor şi care stau la baza testării *on line* a sistemelor. Avantajele acestei testări în raport cu cea *off line* este asigurat de următoarele elemente:

- detectează erorile intermitente
- previne propagarea erorilor la ieşirea sistemului
- permite localizarea poziţiei erorii
- elimină (sau cel puţin reduce substanţial) software-ul de diagnoză la nivelul sistemului

Pentru prevenirea propagării erorii în sistem există două posibilităţi:

1. una din posibilităţi constă în controlul datelor recepţionate în urma transferurilor de date, iar în cazul detecţiei unei erori, fie că se cere retransmisia, fie că se face corectarea datelor transmise cu ajutorul unui cod corector de erori (ca în cazul memoriilor RAM)
2. cealaltă posibilitate constă în controlul rezultatului unei operaţii aritmetice şi în cazul detectării unei erori la nivelul rezultatului se face repetarea operaţiei.

În mod uzual, sistemele mari includ detectoare de erori, precum şi circuite pentru înregistrarea locaţiei (a poziţiei) şi frecvenţei evenimentelor de eroare; acestea fiind utilizate pentru facilitarea întreţinerii rapide şi exacte a sistemului.

Pe de altă parte, testarea *on line* cu ajutorul hardware-ului prezintă şi câteva dezavantaje:

- implică un hardware suplimentar, inclusiv checker-ele care şi ele sunt hardware
- Hardware-ul adiţional trebuie verificat sau testat (*testing the checker problem*)

Checker-ele pot fi clasificate ca fiind:

- De verificare a codurilor (pentru verificarea parităţii, a reziduurilor, a codurilor M din N, a codurilor Brger, etc.) - Se folosesc când ieşirea circuitului funcţional se află în formă codificată.
- De predicţie – se folosesc pentru verificarea circuitelor funcţionale ale căror ieşiri nu sunt în formă codificată. In categoria checker-elor de predicţie se-ncadrează: checker pentru duplicare (*duplicate checker)*, checker cu regenerarea intrărilor (*input regeneration checker)* şi checker-ul de predicţie a parităţii

Def. 3.9 Un circuit este un ***self-testing checker*** dacă el este *self-testing* şi *code-disjoint.*

Dacă checker-ul CK (Fig.3. 1) este un checker ST, şi circuitul L este TSC, atunci întreg circuitul G din Fig.3. 1 este TSC.

Dar, întrucât pe parcursul operării normale a circuitului G intrările checker-ului CK sunt asigurate de ieşirile circuitului funcţional L, este dificil de asigurat pentru checker

BUPT

îndeplinirea condiției ST fără a se adopta nişte tehnici de proiectare speciale [JHA84], [KHAK84], [FUJI87].
Este important de păstrat condiția CD chiar şi în prezența erorilor la nivelul checker-ului; pentru aceasta checker-ul trebuie să fie SCD.
Dacă circuitul verificat L este SFS iar checker-ul CK este SCD, atunci circuitul global G este SFS.

3.1.4 Tehnici de proiectare pentru checker-ele de erori încorporate testabile

Cele mai familiare detectoare de erori sunt *checkere-le de paritate*. Ele detectează la nivelul cuvintelor de *n* biți prezența oricărui număr impar de erori. La baza funcționării detectoarelor de erori stă codificarea în cuvinte de cod a informației din sistemele de calcul.

Detectoarele de erori conțin checker-e (circuite de verificare) ce primesc la intrări informația în formă codificată şi determină dacă aceste cuvinte de informație sunt cuvinte de cod sau nu.

Din moment ce funcția unui circuit de verificare (checker) este cea de furnizare a unui semnal de ieşire activ atunci când cuvintele de intrare nu aparțin codului, pentru testarea lui trebuiesc aplicarea la intrările sale inclusiv cuvinte ce nu aparțin codului verificat. În cazul sistemelor fără erori checker-ele vor recepționa la intrările lor doar cuvinte de cod. Deci, testarea unui checker încorporat în sistem nu este posibilă decât dacă s-a avut în vedere să se prevadă în mod special această abilitate suplimentară la nivelul sistemului.
Dacă sistemul include o facilitate de tip *scan-path* ca şi caracteristică de proiectare pentru testabilitate, atunci orice checker ale cărui intrări provin numai de la bistabile (*latches* sau *flip-flops*) poate fi testat complet prin scanarea (serializarea) în bistabile a unui set corespunzător de *pattern*-uri de test. Pentru checker-ele la care nu toate intrările provin de la bistabilele *scan-path*, este necesară modificarea proiectării checker-ului pentru permiterea testării complete a defecțiunilor singulare de tip blocare la 1 sau 0 (*stuck-at*). În continuare se prezintă tehnici de proiectare pentru checker-ele încorporate ce nu pot fi testate cu ajutorul bistabilelor *scan-path*.

Structura unui detector de erori este determinată în mod special de codul detector de erori utilizat în cuvintele ca vor fi controlate. Codurile pot fi clasificate ca fiind separabile sau nu.
Un cod controlor de paritate pară este un cod separabil cu un singur bit de paritate egal cu paritatea părții controlate din cuvântul de cod. Un alt exemplu de cod separabil este codul rezidual la care partea de control este egală cu restul modulo M al părții de date. Un cod separabil foarte utilizat este duplicarea la care partea de control este identică cu partea de date. La un cod neseparabil nu este posibil să se determine subsetul biților din cuvântul de cod ce este utilizat pentru codificarea părții de date. Codul cu pondere fixă (M din N) este un exemplu de cod neseparabil.

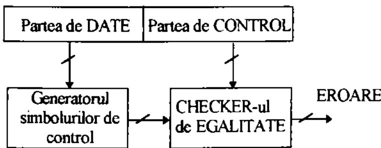


Fig.3. 5

Orice cod separabil poate fi controlat printr-o structură ca cea din Fig.3. 5.
În situația obişnuită când în partea de date pot să apară toate combinațiile posibile de biți nu este nici o problemă în testarea generatorului simbolurilor de control. Dar pentru checker-ul de egalitate situația nu este la fel de simplă, din moment ce combinațiile de intrare corespunzătoare situațiilor de erori nu pot să apară atunci când

BUPT

nu sunt erori. Pentru testarea acestor checker-e se folosec tehnici ce vor fi discutate în continuare.

3.1.4.1 Checker-e de paritate testabile

Codul de paritate este un cod separabil având caracteristic faptul că partea sa de control este formată dintr-un singur bit. Din acest motiv structura checker-ului de paritate este mai simplă decât cea generală a checker-elor pentru coduri separabile. De fapt ea nu implică decât o schemă ce să calculeze suma modulo 2 a biților din cuvântul de cod.

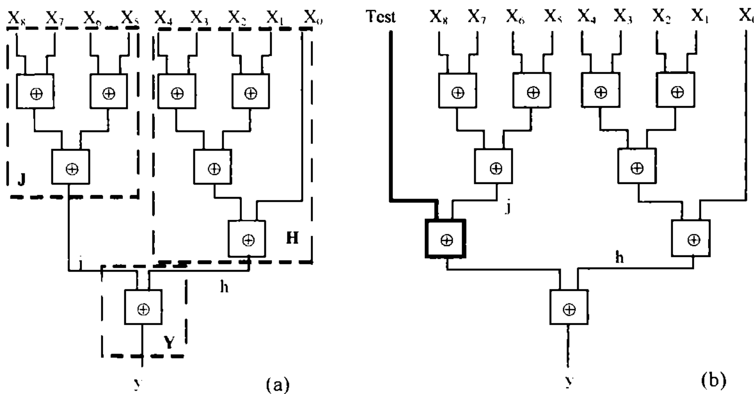


Fig.3. 6

În Fig.3. 6.a. este dat un arbore de porți XOR cu 2 intrări, ce controlează paritatea unui octet codificat după un cod de paritate pe 9 biți. Când este utilizat un cod de paritate impară, ieșirea y este egală cu 1 pentru toate cuvintele valide de cod.

Schema din Fig.3. 6.a. este formată din 3 subscheme: subschema H, intrările și porțile conectate pentru a forma semnalul h; subschema J, intrările și porțile conectate pentru a forma semnalul j; și subschema Y, intrările și porțile conectate pentru a forma semnalul y; Orice eroare singulară de tip s-a-0 (1) din subschemele H sau J poate fi testată, întrucât aceste subscheme recepționează toate combinațiile de intrare posibile. De fapt, aceste subscheme au proprietatea de autotestare (*self-testing*).

O subschemă a unui checker se numește *self-testing* dacă și numai dacă orice defecțiune singular de tip s-a-0 (1) din subschemă cauzează o indicare de eroare la ieșirea checker-ului pentru cel puțin un cuvânt de cod valid prezent la intrarea lui, cu observația că uneori pot fi utilizate alte modele de defecțiune singulară în locul modelului de defecțiune de tip s-a-0 (1) singular.

În general, checker-ele sunt proiectate să aibă proprietatea de autotestare ce garantează indicarea erorii în cazul apariției unei defecțiuni chiar la nivelul schemei lor în cazul funcționării normale.

3.1.4.2 Checker-e cu autotestare

Un checker (pentru un anumit cod) este definit în general ca fiind un circuit cu o ieșire singulară ce ia valoare 0 pentru orice combinații de intrare ce corespund codului, și valoare 1 pentru cuvinte de intrare ce nu aparțin codului. Semnalele ce au o singură valoare logică posibilă în timpul operării normale sunt numite *pasive*

BUPT

Întrucât o defecţiune de tip s-a-0 (1) care indică NO-ERROR la nivelul unei ieşiri nu poate fi detectată decât dacă circuitul are cel puţin două ieşiri, pentru proiectarea checker-elor autotestabile se procedează la înlocuirea tuturor semnalelor pasive cu câte o pereche de semnale, fiecare dintre acestea putând lua pe parcursul operării normale atât valori 0 cât şi valori 1.

Acest concept de implementare a unei funcţii pasive printr-o pereche de funcţii nepasive poate fi formalizat în felul următor: Se consideră o funcţie binară *f* cu valori 0 şi 1, ce sunt mapate în două funcţii binare z_1 şi z_2 pentru care $(z_1,z_2)=(0,0)$ sau $(1,1)$ corespund lui *f* 1 şi $(z_1,z_2)=(0,1)$ sau (1,0) corespund lui *f* 0. Setul funcţiilor (z_1,z_2) este numit *funcţia morfică* ce corespund lui *f*. Anumite funcţii morfice pot fi proiectate să fie total autotestabile.

În general pentru o funcţie cu *n* variabile există 2^n variabile a_i definite în funcţiile morfice corespondente. Se pune problema dacă se pot selecta aceste valori pentru a_i astfel încât z_1 şi z_2 să fie autotestabile.

Orice circuit de verificare cu autotestare completă (*completely self-testing*) trebuie să aibă cel puţin două ieşiri. Aceste circuite sunt circuite pentru care întregul checker este cu autotestare. Practica obişnuită presupune proiectarea unui checker cu autotestare cu două ieşiri ale cărui semnale 01 şi 10 indică funcţionarea fără erori, iar semnalele 00 şi 11 apar ca şi răspuns la o defecţiune singulară de tip s-a-0 (1) din checker, sau la un cuvânt de intrare ce nu aparţine codului. Acestea sunt semnalele care apar la liniile h şi j din Fig.3. 6.a, astfel acest circuit este cu autotestare dacă pentru indicarea erorii este acceptabilă utilizarea a două semnale în loc de unul.

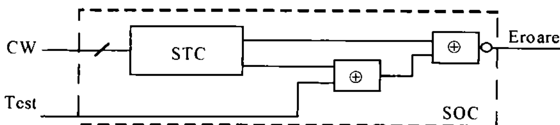
Prin urmare, orice circuit pentru controlul parităţii care are două ieşiri, fiecare egală cu paritatea unuia dintre cele două sub-seturi de intrări, este cu autotestare completă.

Această proprietate de autotestare furnizează o tehnică generală de proiectare a checker-elor încorporate. Există proiectări pentru toate checker-ele importante cu autotestare completă.

Majoritatea sistemelor cu checker-e încorporate necesită câteva indicaţii de operare, atunci când un checker a detectat o eroare (Operarea sistemului poate fi oprită şi se verifică condiţiile checker-elor individuale pentru diagnosticare). Trebuie să fie furnizată o facilitate de combinare a ieşirilor checker-elor individuale într-un singur indicator. În acest sens, vor fi discutate structurile pentru combinarea ieşirilor dublă cale (*two-rail outputs*) ale checker-elor cu autotestare.

Dacă se cere un checker de paritate cu o singură ieşire, circuitul din Fig.3. 6.a poate fi făcut complet testabil prin adăugarea circuitului adiţional desenat cu linii groase din Fig.3. 6.b. În timpul operării normale, intrarea *Test* adăugată este ţinută pe *0* logic, şi operarea circuitului nu este afectată de poarta adăugată.

Pentru testarea porţii de ieşire, intrarea *Test* se poziţionează pe 1 logic, pentru a plasa combinaţiile de 00 şi 11 la intrările acestei porţi. De observat că poarta XOR adăugată nu este cu autotestare (*self testing*) din moment ce intrarea de test este tot timpul pe 0 logic în timpul operării normale, dar ea poate fi testată complet prin poziţionarea semnalului *Test* pe 1.



STC =*Self-Testing Checker* (Circuit verificator cu autotestare)
SOC = *Single-Output Checker* (Circuit verificator cu o singură ieşire)
CW = *Code word* (Cuvânt legal de cod)

Fig.3. 7

BUPT

Această tehnică din Fig.3. 6.b este o tehnică generală şi ea poate fi folosită pentru a converti orice circuit STC (*Self-Testing Checker*) cu două ieşiri într-unul testabil cu o singură ieşire, aşa cum se arată în Fig.3. 7.

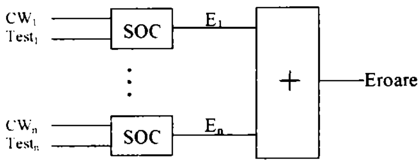


Fig.3. 8

Fig.3. 8 prezintă o structură pentru combinarea ieşirilor a *n* checker-e cu o singură ieşire într-un singur semnal de eroare. Acest circuit necesită *n* semnale de test ($Test_1$, ... , $Test_n$), câte unul pentru fiecare checker cu o singură ieşire.

Poarta OR de la ieşire nu este cu autotestare, deoarece pe tot timpul funcţionării nor-male ea are la toate intrările ei *0* logic. Poarta este totuşi testabilă, deorece semnalele de test individuale pot fi utilizate pentru a aplica toate configuraţiile cu un singur 1, ce sunt necesare pentru a testa complet poarta la toate defecţiunile singulare de tip s-a-0 (1).

3.1.4.3 Checker-e dublă cale

Un dezavantaj al structurii din Fig.3. 8 este faptul că ea necesită *n* semnale adiţionale de test, câte unul pentru fiecare checker individual. Fig.3. 9 arată o structură de combinare a ieşirilor a *n* checker-e cu autotestare (STC) utilizând un singur semnal de test.

Pentru a converti cele *n* perechi de semnale într-o singură pereche de semnale complementare, este folosit un circuit de verificare dublă cale (*two-rail checker*), adică un circuit care controlează dacă fiecare pereche de intrări are valori complementare. Cele două ieşiri sunt apoi convertite într-un singur semnal de ieşire testabil utilizând circuitul din Fig.3. 7.

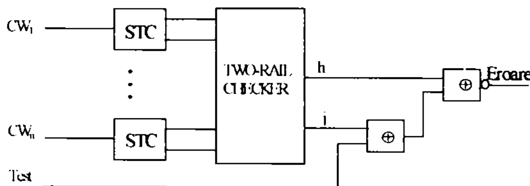


Fig.3. 9

Într-un sistem dublă cale, valorile logice a două linii reprezintă o singură variabilă şi ele au valori complementare. Variabila a_1 poate fi indicată prin două linii care au semnalele corespunzătoare a_1 şi b_1 $(\overline{a_1})$. Semnalele $a_1b_1 = 11$ şi 00 sunt semnalele invalide (ilegale) şi reprezintă o eroare. Pentru a determina dacă sunt valide semnalele de la două variabile, reprezentate prin liniile a_1b_1 şi a_2b_2, se utilizează circuitul *code-disjoint* din Fig.3. 10, descris de următoarele ecuaţii: $f = a_1b_2 + a_2b_1$, $g = a_1a_2 + b_2b_1$.

Fig.3. 10 arată o schemă cu un circuit de verificare dublă cale care converteşte două perechi de semnale de intrare într-o singură pereche de semnale de ieşire. Acest circuit

BUPT

este cu autotestare, dacă şi numai dacă, în timpul operării normale apar la intrările lui toate cele patru cuvinte valide de cod.

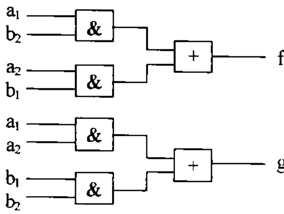


Fig.3. 10

Un arbore de circuite ca cel din Fig.3. 10 poate fi utilizat pentru a converti *n* perechi de intrări într-o singură pereche de ieşire [ITO91]. Autotestabilitatea unui asemenea arbore depinde de prezenţa sau absenţa, pe parcursul operării normale, a tuturor combinaţiilor posibile de valori ale intrărilor sale.

3.1.4.4 Checker-e de egalitate

Checker-ul de egalitate este componenta cheie a checker-elor pentru coduri separabile (Fig.3. 5) şi pentru compararea ieşirilor circuitelor duplicate. Este un tip foarte important de checker, el verificând de fapt egalitatea prin determinarea coincidenţei biţilor corespunzători ai cuvintelor comparate.
În Fig.3. 11 este ilustrată cea mai simplă schemă de checker de egalitate ce compară două cuvinte *X* şi *Y* de *4* biţi. Fiecare pereche de biţi este conectată la o poartă XOR a cărei ieşire trebuie să fie tot timpul pe 0 dacă sistemul operează corect. Ieşirile tuturor porţilor XOR sunt conectate împreună la o poartă OR a cărei ieşire va fi 0 atâta timp cât toate ieşirile porţilor XOR sunt 0.

Acest circuit fiind foarte simplu, el nu este cu autotestare şi este foarte dificil de testat, întrucât prin nici o intrare a circuitului nu se pot detecta defecţiunile singulare sau multiple de tip s-a-0 de la o ieşirile oricărei porţi XOR.
Schema circuitului poate fi făcută testabilă prin adăugarea a câte unei porţi XOR şi a câte unui semnal de test la ieşirile fiecăreia din porţile XOR existente. În felul acesta este posibilă complementarea selectivă a câte uneia din intrările porţilor XOR originale (ca la intrarea *j* din Fig.3. 6). Totuşi, această modificare care dublează de fapt complexitatea circuitului nu rezolvă problema autotestării checker-ului obţinut.

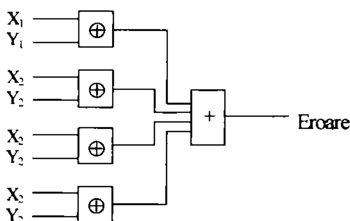


Fig.3. 11

Pentru a obţine un checker de egalitate cu autotestare trebuie complementaţi toţi biţii din unul din cuvintele ce trebuiesc comparate, formându-se astfel un cod dublă cale (*two-rail code*).
Astfel, se pot utiliza pentru realizarea checker-elor de egalitate testabile checker-ele dublă cale di t m .
O altă posibilitate de implementare a checker-elor de egalitate autotestabile este cea de concatenare a cuvântului comple-mentat şi necomplementat, rezultând un cuvânt de cod la care jumătate din biţii săi sunt egali cu 1. Deci, pentru formarea unui checker de egalitate testabil se poate utiliza un checker *k* din *2k*.
De observat că este discutabil avantajul primei implementări în raport cu cea de-a doua.

BUPT

3.1.4.5 Checker-e M din N

Aceste checker-e au o utilizare mai restrânsă comparativ cu cea a celorlalte tipuri de checker-e [DIMA95].

Codurile cu ponderea fixă M din N se utilizează atunci când se dorește nu numai detectarea tuturor defectelor singulare de tip s-a-0(1), ci și detectarea tuturor defectelor unidirecționale..

Codul zecimal *2* din *5* este cel mai cunoscut exemplu al unui cod M din N. Codurile k din $2k$ sau k din $2k-1$ sunt implementările uzuale din moment ce ele conțin numărul maxim de cuvinte de cod pentru o lungime de cuvânt dată.

Andersom și Metze (1973) au dat următoarea soluție de proiectare a checker-elor STC $k/2k$. Checker-ul propus are două ieșiri f și g. Cele $2k$ intrări sunt partiționate în două seturi disjuncte cu k intrări fiecare: X_A și X_B. Funcția f este definită să aibă valoarea 1 dacă și numai dacă cel puțin i dintre variabilele din setul X_A au valoarea 1, și cel puțin k-i dintre variabilele din X_B au valoarea 1, pentru i par. Similar, funcția g este definită să aibă valoarea 1 dacă și numai dacă cel puțin i dintre variabilele din setul X_A au valoarea 1, și cel puțin k-i dintre variabilele din X_B au valoarea 1, pentru i impar.

Adică:

$$f = \sum_{i=0}^{k} T(m_A \geq i) \cdot T(m_B \geq k - i), \qquad i = \text{întreg par}$$

$$g = \sum_{i=0}^{k} T(m_A \geq i) \cdot T(m_B \geq k - i), \qquad i = \text{întreg impar}$$

unde m_A și m_B reprezintă numărul unităților ce apar în subseturile X_A și respectiv X_B, iar $T(m_A \geq i)$ reprezintă funcția Booleană ce are valoare 1 dacă și numai dacă numărul unităților din subsetul X_A este mai mare sau egal decât valoarea lui i. Suma semnifică operația logică OR.

Pentru cuvinte de intrare de cod (k dintre cele $2k$ variabile de intrare au valoarea 1) ieșirile circuitului sunt $f=1$ și $g=0$, sau $f=0$ și $g=1$, în timp ce pentru cuvinte de intrare noncod la care numărul unităților este mai mic decât k ieșirile circuitului sunt $f=g=0$, iar pentru cuvinte de intrare la care numărul unităților este mai mare decât k, ieșirile circuitului sunt $f=g=1$.

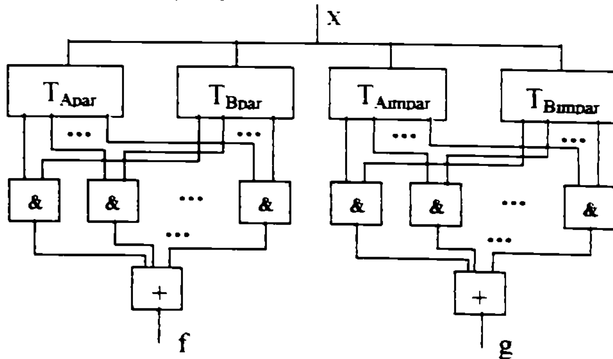


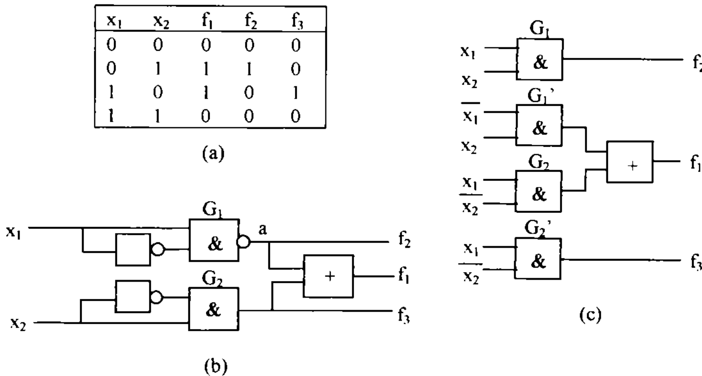
Fig.3. 12

BUPT

În Fig.3. 12 este dată forma generală a unui asemenea circuit, în care T_{Apar} are ieşiri pentru fiecare valoare pară a lui i, şi ieşirea i are valoare 1 dacă cel puţin i din intrările sale au valoare 1. Subcircuitele T_{Bpar}, T_{Aimpar} şi T_{Bimpar} sunt obţinute în mod similar. Se poate arăta că acest circuit este atât ST cât şi FS, deci el este un circuit TSC (chiar checker TSC).
Checker-ele autotestabile pentru codurile $k/2k$ pot fi utilizate şi pentru sinteza checker-elor mai generale pentru coduri k/n, unde $n \neq 2k$.

3.1.5 Realizarea circuitelor combinaţionale autotestabile

Vom considera problema generală de proiectare a circuitelor autotestabile pentru funcţii arbitrare. Este evident că intrările unui circuit autotestabil trebuiesc codificate cu un cod cu distanţă d, unde d trebuie să fie cel puţin 2, pentru ca circuitul să fie FS în raport cu defectele singulare de tip s-a-0 (1) ale intrărilor.
Pentru ca circuitul să fie FS în raport şi cu defectele singulare de tip s-a0 (1) de la ieşire, trebuie ca şi ieşirile să fie codificate în mod similar.
Dacă presupunem că nu există defecţiuni ale intrărilor, şi că intrările sunt necodificate, autotestarea este realizată automat, şi circuitul trebuie proiectat doar pentru a fi FS.
Dacă se definesc ieşirile circuitului ca fiind de paritate pară, ele definesc o distanţă de cod 2. Totuşi, aşa cum se arată în Fig.3. 13 pentru defectul a s-a-0 şi intrările $(x_1, x_2) = (0,1)$, este posibil ca un singur defect să afecteze două ieşiri. În acest caz circuitul nu este FS, întrucât ieşirea normală $(f_1, f_2, f_3) = (1,1,0)$ şi ieşirea cu defect $(f_1, f_2, f_3) = (0,0,0)$ sunt ambele ieşiri valide posibile ale circuitului pentru o anumită combinaţie de intrare. Pentru prevenirea unui asemenea *pattern* de eroare, circuitul poate fi reproiectat aşa cum se arată în Fig.3. 13.c, unde a fost asigurată duplicarea anumitor semnale în vederea asigurării erorilor de paritate impară.
De observat că fiecare din porţile G_1 şi G_2 ale Fig.3. 13.b. au fost înlocuite cu două porţi, şi se presupune mai departe că nici unul din defectele de intrare nu poate afecta simultan cele două porţi. Prin urmare, dacă nu sunt considerate asemenea defecţiuni ale intrărilor, este posibilă proiectarea unor realizări autotestabile ale circuitelor combinaţionale arbitrare.



x_1	x_2	f_1	f_2	f_3
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	0	0

Fig.3. 13

BUPT

În Fig.3. 14 este dată schema de principiu pentru indicarea erorii printr-un singur semnal pasiv pentru circuitele TSC.

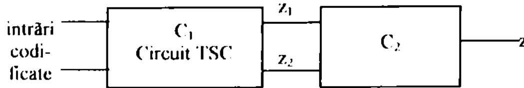


Fig.3. 14

Circuitul C_2 reprezintă *hardcore*-ul ansamblului. Se pune problema dacă logica circuitului C_1 poate fi simplificată în mod substanțial, prin creşterea într-o oarecare măsură a complexității circuitului C_2. Pare probabil că teoria generală a circuitelor autotestabile nu este restricționată la funcțiile morfice cu două ieşiri şi trebuie să recunoască interdependența între dimensiunea *hardcore*-ului şi gradul de autotestabilitate.
În acest context se poate considera o clasă de circuite ce sunt TFS (*totally fault secure*), dar numai parțial *self-testing* şi o clasă de circuite ce sunt TST (*totally self testing*), dar sunt FS doar pentru un subset al tuturor intrărilor posibile. Cu toate acestea încă nu s-au dezvoltat proceduri generale de proiectare pentru asemenea circuite.

3.1.6 Autotestarea circuitelor secvențiale

Teoria codurilor, precum şi proiectarea autotestabilă sunt aplicabile şi circuitelor secvențiale. Considerăm modelul circuitului secvențial din Fig.3. 15, în care sunt codificate atât ieşirile cât şi varibilele de stare. Se poate obține autotestarea prin proiectarea logicii combinaționale astfel încât:

1. Pentru orice defect intern al lui C şi pentru orice intrare, fie ieşirea şi starea următoare sunt corecte, fie ieşirea şi/sau starea următoare sunt cuvinte noncod.
2. Pentru orice stare corespunzătoare unui cuvânt noncod rezultat printr-un defect f a lui C, şi pentru orice intrare, starea următoare generată de C cu defectul f este un cuvânt noncod şi ieşirea este şi ea un cuvânt noncod.

În Fig.3. 16 este ilustrat conceptul unei astfel de proiectări. Ieşirile circuitului secvențial C_1 codificate în cod k/n, sunt intrări pentru checker-ul k/n. Checker-ul generează un semnal R_s ce este necesar pentru generarea următorului impuls clock (iar în cazul circuitelor asincrone pentru generarea următorului semnal de intrare). Astfel, pentru cele mai multe defecte sistemul se va opri odată ce a fost generat un semnal eronat.

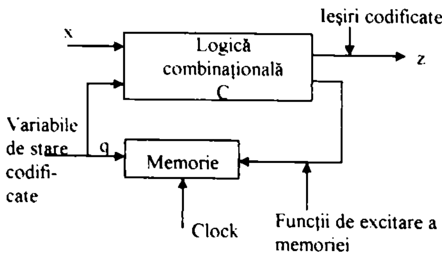


Fig.3. 15

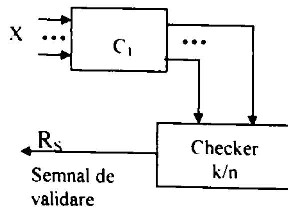


Fig.3. 16

BUPT

Un număr considerabil de cercetări din domeniu s-au concentrat asupra aplicării conceptelor de bază ale autotestării în proiectarea sistemelor complexe, inclusiv a unităţilor de control microprogramate [NICO87][PSCH87].

3.1.7 Proiectarea unui sistem TSC

Cu toate că un volum mare al cercetărilor din domeniu [GAST83], [NIKO88], [JHA89] au fost direcţionate spre dezvoltarea unor circuite TSC cât mai eficiente pentru majoritatea codurilor uzuale, încă nu s-a ajuns la stabilirea unei teorii generale de proiectare valabilă pentru circuite arbitrare de tip combinaţional, respectiv secvenţial.

O situaţie interesantă apare când checker-ul are o defecţiune în sine. Din moment ce se presupune că unitatea are numai o singură defecţiune la un moment dat, se poate considera că dacă checker-ul are o defecţiune, unitatea funcţională nu are nici o defecţiune. Prin urmare, unitatea funcţională nu va produce nici o ieşire ilegală. Astfel, avem nevoie de un mecanism prin care fiecare defect din interiorul checker-ului să se releve ca o eroare la ieşirea lui. Astfel, se presupune implicit că toate intrările de test ale sitemului TSC sunt aplicate în modul operării normale; adică checker-ul este *self-testing*.

De asemenea, trebuie să se codifice ieşirile checker-ului printr-un cod detector de erori unidireţionale [FUJI91]. Checker-ul trebuie să producă cât mai puţine ieşiri deorece ele trebuie să fie monitorizate din exterior. Cel mai mic cod nesistematic cunoscut, numit cod dublă cale (*dual rail-code*), este $\{01,10\}$. O eroare unidirecţională transformă acest cod în $\{00 \text{ } sau \text{ } 11\}$ şi astfel ieşirile checker-ului vor indica o eroare. Prin urmare, prin observarea unei ieşiri ilegale la un checker se poate detecta o defecţiune/eroare.

Relativ la checker-ul TSC se punctează faptul că ieşirea lui devine 00 sau 11, dacă şi numai dacă, este detectată o eroare la ieşirea unei unităţi funcţionale, sau dacă checker-ul în sine are o defecţiune hardware. Deci, un checker TSC trebuie să fie *code-disjoint*. Există de asemenea situaţii în care chiar dacă un checker este proiectat pentru *code-disjointness*, nu sunt disponibile de la unitatea funcţională TSC toate intrările de test necesare pentru testarea lui.

Pentru exemplificare considerăm un sistem bazat pe un cod dublă cale, ce este des utilizat în sistemele cu autoverificare (*self-checking*).

a_1	b_1	a_2	b_2	f	g
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1

Tabel 3. 3

a_1	b_1	a_2	b_2
1	0	1	0
0	1	0	1
0	1	1	0

Tabel 3. 4

Tabel 3. 3 arată testele pentru Fig.3. 10. Ieşirile la funcţionare normală sunt $fg = 01$ sau 10. O intrare eronată cauzează o ieşire eronată. De exemplu, dacă a_1b_1 = 11, atunci fg = 11 şi dacă a_1b_1 = 00, atunci $fg = 00$.

Pentru ca acest circuit de verificare dublă cale să fie cu autotestare, el are nevoie de intrările din Tabel 3. 3; dar să presupunem că recepţionează numai 3 din cele 4 teste de care are nevoie, şi anume cele din Tabel 3. 4.

În această situaţie, nu poate fi detectat defectul b_2 s-a-0 pentru ieşirea f din Fig.3. 10, întrucât nu sunt disponibile semnalele de intrare $a_1 = 1$ şi $a_2 = 0$. În mod similar, pentru

BUPT

ieşirea g nu pot fi detectate defectele a_2 s-a-1 sau b_1 s-a-1. Deci, cu toate că acest checker este *code-disjoint*, el nu este cu autotestare fără cele patru combinaţii date în Tabel 3. 3.

Pentru a face *self-testing* circuitele care generează f şi g, noile ecuaţii pentru f şi g sunt: $f = a_2 b_1$ şi $g = a_1 + b_2$.
Pentru aceste noi ecuaţii, dacă $a_2 = 1$ şi $b_2 = 1$ când $a_1 = 1$ şi $b_1 = 0$, obţinem $f = 0$ şi $g = 1$. Astfel, acest circuit nu mai este *code-disjoint*.
Ne aflăm deci într-o situaţie fără soluţie. Dacă circuitul este făcut *self-testing*, atunci el nu este *code-disjont*. Astfel, în acestă situaţie nu se va obţine niciodată un checker TSC, adică un circuit care să fie atât *code-disjoint* cât şi *self-testing*.

Deci, pentru a face un circuit să fie atât *self-testing* cât şi *code-disjoint*, ar tebui ca din când în când să se suspende operarea normală a circuitului pentru a alimenta checker-ul cu intrări de test care la operare normală nu sunt disponibile. Apare însă problema faptului că în acestă situaţie circuitul nu mai este *on-line* şi există dificultăţi practice de alimentare a checker-ului. Este preferată soluţia de proiectare a checker-elor ce să fie atât *self-testing* cât şi *code-disjoint* şi care să nu aibă nevoie de acces din exterior, fiind astfel uşor de încorporat.

Fig.3. 17 arată schema bloc a unui sistem TSC. De obicei, biţii de control se aleg în aşa fel încât ieşirile blocului funcţional să fie un cod detector de erori unidirecţionale. În proiectarea propusă de Kundu şi Reddy, este necesar ca biţii de control să fie selectaţi pentru a permite proiectarea checker-elor TSC. Această necesitate poate implica partiţionarea setului de ieşiri funcţionale în aşa fel încât biţii din fiecare bloc de partiţie să se codifice într-un cod nesistematic. În felul acesta se poate proiecta un checker TSC cu două ieşiri. Checker-ul TSC descris utilizează o schemă de arbore de comparatoare dublă cale cu două intrări. Intrările unui comparator dublă cale se realizează cu două perechi de intrări (a_1, b_1) şi (a_2, b_2).
Cele două ieşiri f şi g ale comparatorului cu două intrări sunt:

$$f = a_1 b_2 + a_2 b_1, \quad g = a_1 a_2 + b_2 b_1.$$

Majoritatea checker-elor TSC pentru coduri detectoare de erori au două ieşiri care produc în timpul funcţionării normale fie *01*, fie *10*. Prin urmare, ieşirile a două checker-e TSC pot fi monitorizate prin utilizarea checker-ului dublă cale cu două intrări din Fig.3. 17.

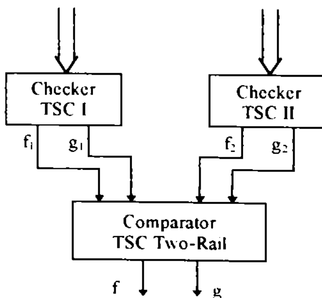


Fig.3. 17

f_1	g_1	f_2	g_2
0	1	0	1
1	0	1	0

Tabel 3. 5

Întregul circuit din figură poate fi TSC, dacă şi numai dacă, $f_1 g_1$ şi $f_2 g_2$ furnizează toate cele 4 teste din Tabel 3. 3. De exemplu, dacă $f_1 g_1$, $f_2 g_2$ iau numai cele două valori din Tabel 3. 5, este clar că checker-ul din Fig.3. 17 nu este TSC.

BUPT

În acest caz se pot genera toate testele pentru comparatorul dublă cale prin inserarea unei întârzieri de tact (*clock-delay*) sau a bistabilelor D la ieşirile unuia dintre checker-ele TSC, aşa cum este ilustrat în Fig.3. 18 (este vorba de ieşirea fiecărui checker de la primul nivel).

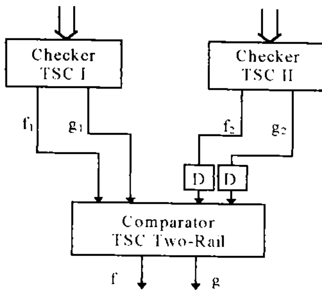


Fig.3. 18

Se verifică uşor că în această situaţie sunt disponibile toate testele pentru comparatorul dublă cale. Necesitatea utilizării bistabilelor D din Fig.3. 18 nu ridică nici un fel de problemă întrucât bistabilele sincrone sunt componente obişnuite ale sistemelor de calcul digitale.

In concluzie, dacă $y_1, y_2, y_3, \ldots, y_k$ sunt ieşirile funcţionale ale ansamblului din Fig.3. 17, pentru construirea unei unităţi TSC şi a unui checker TSC se aplică următoare procedură:

1. Se partiţionează ieşirile funcţionale $y_1, y_2, y_3, \ldots, y_k$ în blocuri astfel încât biţii din fiecare bloc să poată fi codificaţi printr-un cod nesistematic pentru care există un checker normal TSC cu două ieşiri.
2. Se proiectează checker-ele cu două ieşiri pentru codurile de la punctul 1
3. Ieşirile checker-elor TSC se monitorizează printr-un arbore de comparatore dublă cale cu două intrări

[KAPU92] propune o altă procedură de proiectare a checker-elor TSC.

3.1.8 Reconfigurări BIST ale structurilor de însumare bazate pe controlul parităţii

O serie de cercetări au fost investite în direcţia obţinerii unor reconfigurări BIST ale structurilor de însumare, importante fiind în acest sens cele ce permit detecţia erorilor pe baza codurilor de paritate şi a codului sumă de control (*checksum code*), precum şi cele ce permit corecţia erorilor prin utilizarea codurilor combinate, respectiv a codurilor bazate pe paritate [HUAN84], [TOHM86].

Pentru stabilirea caracteristicilor de eroare ale structurilor de însumare, vom denumi prin:

- circuit de însumare – subcircuitul ce realizează implementarea ecuaţiei de generare a sumei la nivelul fiecărui bit ce se însumează
- circuit CY – subcircuitul ce realizează implementarea ecuaţiei de generare a lui CY

Astfel, punctăm următoarele caracteristici ale acestor structuri:

1. Erorile circuitului de însumare nu se propagă, cauzând astfel doar erori singulare.
2. O eroare la nivelul unui bit CY se propagă totdeauna la următorul bit sumă pe care-l afectează.
3. Orice eroare din circuitul de propagare în cascadă a lui CY poate provoca pachete de erori (*error bursts*), în timp ce pentru situaţia circuitului de anticipare a lui CY, doar erorile ce afectează semnalele P_i, respectiv G_i pot provoca pachete de erori.
4. Intr-un circuit de anticipare a transportului semnalele CY nu depind de cele anterioare lor.

BUPT

A=	0	0	1	0	p_A =1
B=	0	1	1	0	P_B =0
(C=	1	1	0	0)	P_C =0
					P_S =1
S=	1	0	1	0 →	P_S =0

Tabel 3. 6

Pe baza precizărilor de mai sus, analizăm implicațiile verificării parității la structurile de însumare.

Fie p_A şi p_B biţii de verificare a parităţii pentru operanzii A şi B ce se însumează, iar p_C paritatea semnalelor CY interne sumatorului. Paritatea sumei este dată de:

$$
\begin{aligned}
p_S &= S_{n-1} \oplus S_{n-2} \oplus \ldots \oplus S_1 \oplus S_0 = \\
&= (A_{n-1} \oplus B_{n-1} \oplus C_{n-2}) \oplus (A_{n-2} \oplus B_{n-2} \oplus C_{n-3}) \oplus \ldots \oplus (A_0 \oplus B_0 \oplus C_{-1}) = \\
&= (A_{n-1} \oplus A_{n-2} \oplus \ldots \oplus A_0) \oplus (B_{n-1} \oplus B_{n-2} \oplus \ldots \oplus B_0) \oplus \qquad (3.1) \\
&= \oplus (C_{n-2} \oplus C_{n-1} \oplus \ldots \oplus C_{-1}) = \\
&= p_A \oplus p_B \oplus p_C
\end{aligned}
$$

Verificarea definită de ecuaţia (3.1) este numită verificarea parităţii pentru întreaga sumă (*full-sum parity check*) şi ea presupune compararea valorii prezise pentru paritate: $p_A \oplus p_B \oplus p_C$ cu valoarea actuală a parităţii sumei: p_S (Fig.3. 19).

Pentru exemplificarea metodei considerăm în Tabel 3. 6 situaţia însumării operanzilor $A = (0010)$ cu $B = (0110)$.

Dezavantajul acestei metode constă în faptul că ea nu permite detecţia erorilor pentru semnalele CY. Aceasta se datorează faptului că întotdeauna o eroare CY cauzează o eroare a unui bit sumă, rezultând astfel că numărul erorilor ce apar este întotdeauna par.

Pentru surmontarea acestei probleme se apelează la:

- Tehnica de dublare a logicii semnalelor CY obţinân-du-se astfel un sumator cu verificarea parităţii prin dublarea lui CY (*Duplicate Carry with Parity Check Adder* -DCPCA)
- Tehnica de însumare prin condiţionarea biţilor sumă de cei ai semnalelor CY ce se generează pentru rangul următor obţinându-se sumatorul cu suma dependentă de CY (*Carry-Dependent Sum Adder* – CDSA)

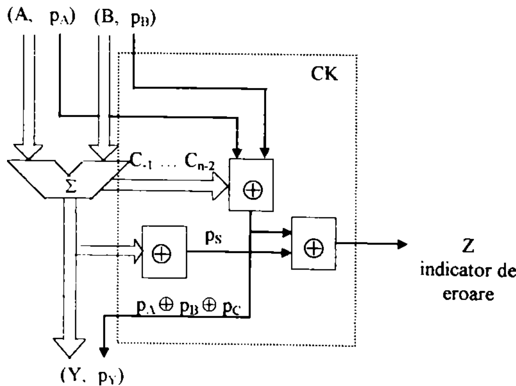


Fig.3. 19

BUPT

3.1.8.1 Sumator cu verificarea parității prin dublarea lui CY

În Fig.3. 20 este dat un exemplu de asemnea sumator în care este dublat circuitul de generare a lui CY, obținându-se prin dublare semnalul p_{cd}.

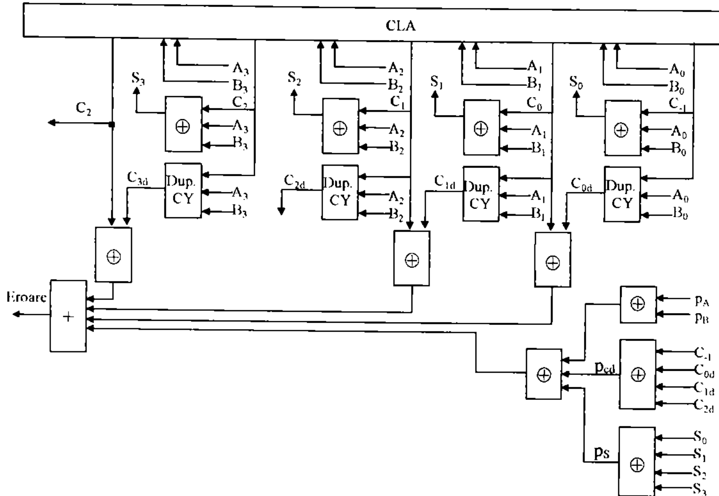


Fig.3. 20

Prin urmare, o verificare a parității pe baza relației:

$$p_S = P_A \oplus p_B \oplus p_{cd}$$

duce la detectarea tuturor erorilor CY cauzate de:

- un defect singular – în cazul sumatoarelor cu propagare serială a transportului
- erori ale semnalelor P_i, respectiv G_i – în cazul sumatoarelor cu anticiparea transportului.

3.1.8.2 Sumator cu suma dependentă de CY

Această tehnică de autoverificare a structurilor de însumare se bazează pe observația că pentru a putea detecta prin controlul parității erorile ce apar în lanțul CY a unui sumator, trebuie ca aceste erori să producă un număr impar de erori în cadrul pachetelor de erori.

A_i	B_i	C_{i-1}	S_i	C_i	$f_i = S_i \oplus C_i$
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	0

Tabel 3. 7

Sumatorul CDSA prezintă această proprietate (Fig.3. 21).

Pentru a realiza dezideratul unui număr impar de erori în cadrul pachetelor de erori provocate de eroarea semnalului C_i (care în condițiile structurilor de însumare prezentate provoacă un pachet cu număr par de erori: C_i, S_{i+1}, ..., C_{i+t-1}, S_{i-t}), trebuie forțată dependența valorii bitului

BUPT

sumă S_i de valoarea bitului eronat C_i.
Aceasta se realizează utilizând ca şi ecuaţie de generare a biţilor sumă o ecuaţie de forma:

$$S_i = f_i \oplus C_i \qquad (3.2)$$

unde f_i este o funcţie de A_i, B_i şi C_{i-1}.
Expresia funcţiei f_i poate fi dedusă din Tabel 3. 7. Astfel, pentru f_i se obţine următoarea ecuaţie:

$$f_i = \overline{A_i \cdot B_i \cdot C_{i-1} + \overline{A_i} \cdot \overline{B_i} \cdot \overline{C_{i-1}}}$$
$$= \overline{G_i \cdot C_{i-1} + \overline{T_i} \cdot \overline{C_{i-1}}}$$

Prin urmare, ecuaţia (3.2) devine:

$$S_i = G_i \cdot C_{i-1} \cdot C_i + \overline{G_i} \cdot C_{i-1} \cdot \overline{C_i} + T_i \cdot \overline{C_{i-1}} \cdot \overline{C_i} + \overline{T_i} \cdot \overline{C_{i-1}} \cdot C_i + \overline{G_i} \cdot T_i \cdot \overline{C_i}$$

Considerăm situaţia când există o eroare la intrarea sumatorului (nu şi la nivelul sumatorului). Pe baza ecuaţiei (3.2) se prezice paritatea sumei p_S. Fig.3. 22 prezintă circuitul ale cărui ieşiri sunt codificate pe baza codului de paritate (de forma (S, p_S)).

Dacă la intrările (A, p_A) sau (B, p_B)ale circuitului din Fig.3. 21 există o singură eroare, circuitul nefiind afectat de erori, atunci întotdeauna eroarea de la intrare va fi propagată spre ieşirea circuitului.
Aceasta se explică în felul următor:

- Dacă există o eroare pe linia p_A (sau p_B); această eroare este propagată totdeauna numai la ieşirea de paritate p_S şi nu şi la ieşirea S. Prin urmare, ieşirea (S, p_S) nu este un cuvânt de cod.
- Dacă există o eroare la nivelul bitului i a lui A (rezultatele analizei sunt valabile şi pentru operandul B), atunci acestă eroare se va propaga întotdeauna la S_i, întrucât aceasta este o funcţie liniară de A_i ($S_i = A_i \oplus B_i \cdot C_{i-1}$). Pe de altă parte, propagarea acestei erori la bitul C_i este condiţionată de valoarea lui B_i şi C_{i-1}. Dacă această eroare este propagată la C_i, atunci ea se propagă şi la ieşirea bitului sumă a rangului următor (S_{i+1}).

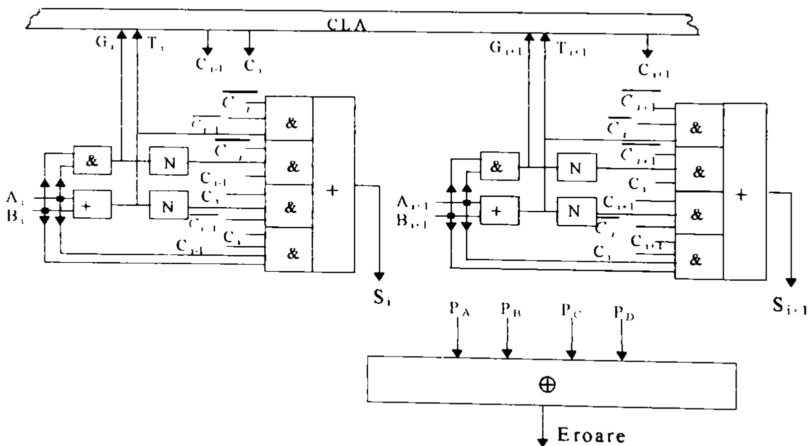


Fig.3. 21

BUPT

În mod similar, dacă eroarea este propagată la un CY a altui rang (de exemplu C_{i+t+1}), atunci şi S_{i+t+1} va fi eronat.

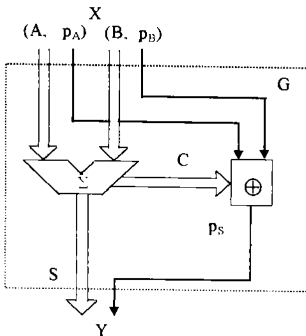


Fig.3. 22

Rezultă că eroarea singulară poate fi propagată printr-o cale de forma: $S_i, (C_i, S_{i+1}), \ldots, (C_{i+t}, S_{i+t+1}), \ldots,$; astfel, întotdeauna rezultă un număr impar de erori la nivelul biţilor sumă şi ai biţilor CY. Aceasta înseamnă că dacă la nivelul biţilor sumă apare un număr par de erori, la nivelul biţilor CY vom avea un număr impar de erori şi invers. Prin urmare, ieşirea (S, p_S) nu este un cuvânt de cod, circuitul G din Fig.3. 22 fiind CD pentru erori singulare ale spaţiului de intrare.

3.1.9 Reconfigurări BIST ale structurilor de însumare bazate pe codul sumă de control

Codurile sumă de control (*checksum code)*, numite şi coduri de paritate la nivel de digit, pot fi utilizate pentru detecţia defectelor singulare la nivelul *byte*-ilor.
Un cod sumă de control este dat de un set de $(n+1)$ simboluri din mulţimea Z_q (a întregilor modulo q). Fiecare vector are o componentă numită simbol de verificare, care este egală cu suma modulo q a celorlalte componente, numite simboluri de informaţie ale vectorului.

Def.3.10. Un cod sumă de control este dat de mulţimea:

$$\left\{ (x_c, x_{n-1}, \ldots, x_0) \,\middle|\, (x_i, x_c \in Z_q)\ si\ \left(x_c = \sum_{0 \le i \le n-1} x_i \bmod q \right) \right\}$$

Pe baza acestei definiţii este evident că minimul distanţei Hamming a acestui cod este 2.

Prezintă un interes deosebit codurile pentru care q este egal cu 2^b, cu $b \in Z, b > 1$ (pentru $b = 1$ codul degenerează în codul de imparitate). În aceste coduri, fiecare simbol din Z_{2^b} poate fi codificat ca şi *byte* cu b cifre binare. Prin urmare, fiecare cuvânt de cod are $n \cdot b$ biţi de informaţie şi b biţi de verificare. Sunt detectate toate erorile limitate la un singur *byte* de b cifre binare.

Considerăm adunarea modulo 2^b a vectorilor cu orice număr de componente din Z_{2^b}. Intr-un sumator autoverificabil ce operază suma modulo 2^b a operanzilor se pot detecta toate erorile singulare la nivel de *byte*. Prin urmare va fi detectat orice defect la nivelul sumatorului ce cauzează erori singulare la nivelul *byte*- ilor. Aceste defecte aparţin mulţimii defectelor F.
Considerăm doi vectori de intrare ce aparţin codului sumă de control şi a căror formă este:

$$A = (A_c, A_{n-1}, A_{n-2}, \ldots, A_0)$$
$$B = (B_c, B_{n-1}, B_{n-2}, \ldots, B_0)$$

BUPT

unde A_c şi B_c sunt simbolurile de control, şi $(A_{n-1}, A_{n-2}, \dots, A_0)$, respectiv $(B_{n-1}, B_{n-2}, \dots, B_0)$ sunt partea de informaţie utilă a codului,

$$A_d = (A_{n-1}, A_{n-2}, \dots, A_0)$$
$$B_d = (B_{n-1}, B_{n-2}, \dots, B_0)$$

Părţile de informaţie utilă A_d şi B_d sunt reprezentările întregilor $[A_d],[B_d]$ astfel încât:

$$[A_d] = \sum_{0 \le i \le n-1} A_i \cdot 2^{bi}$$
$$[B_d] = \sum_{0 \le i \le n-1} B_i \cdot 2^{bi}$$

Adunarea obişnuită a părţilor de informaţie a cuvintelor de cod este definită de:

$$S_d = A_d + B_d + C'$$
$$[S_d] = ([A_d] + [B_d]) \bmod M$$
$$C' = (C'_{n-2}, \dots, C'_0, C'_{-1})$$
$$C'_i = \begin{cases} 0 & daca \ A_i + B_i + C'_{i-1} < 2^b \\ 1 & daca \ A_i + B_i + C'_{i-1} \ge 2^b \end{cases} \qquad 0 \le i \le n-2$$

Pentru ecuaţiile de mai sus:

- Pentru adunare în complement faţă de 2 avem $M = 2^b$ şi $C'_{-1} = 0$
- Pentru adunare în complement faţă de 1 avem $M = 2^b - 1$ şi $C'_{-1} = C'_{n-1}$

Simbolul de verificare a sumei a două părţi de informaţie este egal cu rezultatul adunării simbolurilor sumă obţinute:

$$S_c = \sum_{0 \le i \le n-1} S_i \bmod 2^b \qquad (3.3)$$

Pe de altă parte, S_c poate fi prezis pe baza simbolurilor de verificare ale intrărilor şi a lui CY, pe baza relaţiei:

$$S_c = (A_c + B_c + C'_c) \bmod 2^b \qquad (3.4)$$

unde $C'_c = \sum_{0 \le i \le n-1} C'_{i-1} \cdot \bmod 2^b$

Pentru exemplificare considerăm următoarea adunare în complement faţă de 2:

- Fie $n = 4$, $b = 3$, $A = (7,2,2,4,7)$, $B = (5,3,3,4,3)$ şi $M = 2^3$

  Atunci, $C' = (0,1,1,0)$ şi $S_d = ((2,2,4,7) + (3,3,4,3) + (0,1,1,0)) \bmod 2^3$ din care obţinem: $S_c = (S_3 + S_2 + S_1 + S_0) \bmod 2^3 = 6$.
- Pe de altă parte, pe baza ecuaţiei (3.4) avem $A_c = 7$, $B_c = 5$ şi $C'_c = 2$. S_c poate fi prezis ca fiind: $S_c = (A_c + B_c + C'_c) \bmod 2^3 = 6$.

Rezultatele ecuaţiilor (3.3) şi (3.4.) sunt comparate şi rezultatul comparării este semnalizat de indicatorul de eroare din Fig.3. 23. Checker-ul din Fig.3. 23 aparţine clasei checker-elor de predicţie. Circuitul format din sumator împreună cu circuitul de verificare a simbolului prezis este CD în raport cu erorile singulare ale intrărilor. Nici acest checker nu detectează defectele de pe circuitul de generare a lui CY, întrucât acestea produc erori compensate atât în S_c cât şi în C'_c, şi cu toate că ieşirea este incorectă nu se generează semnal de indicare a erorii. Această problemă poate fi surmontată prin apelare la dublarea logicii CY.

Schema de verificare din Fig.3. 23 poate fi aplicată efectiv la sumatorul format din sumatoare *byte-sliced*.

BUPT

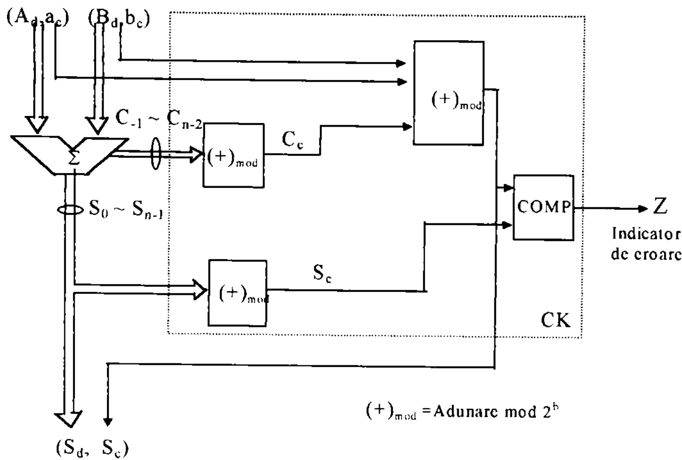


Fig.3. 23

3.1.10 Reconfigurări BIST ale structurilor de însumare bazate pe coduri combinate

Codurile corectoare de erori pot fi împărțite în două clase:

- Clasa codurilor bazate pe controlul parității – s-au dovedit foarte eficiente pentru memorii și operații de transfer a datelor
- Clasa codurilor reziduale – s-au dovedit eficiente în controlul operațiilor aritmetice

Pentru a realiza o corecție eficientă la nivelul unităților aritmetice ce includ structuri de însumare, se pot utiliza coduri combinate ce include atât control bazat pe paritate, cât și control bazat pe reziduuri.
Un astfel de cod combinat (Fig.3. 24) aplicat informației X duce la obținerea unui cuvânt de cod de forma: $[X,P,[X]_m]$, unde $[X]_m$ reprezintă reziduul lui X modulo m $(=2^l-1)$, iar P este format din biți de paritate calculați pentru *byte*-ii lui X.
Mecanismul de corecție a unui asemenea cod îl vom descrie prin următorul exemplu:

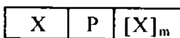


Fig.3. 24

Considerăm informația X ca fiind compusă din 7 *byte*-i, $B_6, B_5, \ldots, B_0$, fiecare *byte* având 3 biți. Obținem pentru X forma: $X=(x_{20}, x_{19}, \ldots, x_0)$.
Pentru X se calculează 3 biți de control a parității (P) și un reziduu de verificare ($[X]_7$).

Partea de control a parității, $P=(P_0,P_1,P_2)$, este calculată pe baza *byte*-ilor lui X după cum urmează:

$$P_0 = B_0 \oplus B_2 \oplus B_4 \oplus B_6$$
$$= (x_0 \oplus x_1 \oplus x_2) \oplus (x_6 \oplus x_7 \oplus x_8) \oplus (x_{12} \oplus x_{13} \oplus x_{14}) \oplus (x_{18} \oplus x_{19} \oplus x_{20})$$

BUPT

$$P_1 = B_1 \oplus B_2 \oplus B_5 \oplus B_6$$
$$= (x_3 \oplus x_4 \oplus x_5) \oplus (x_6 \oplus x_7 \oplus x_8) \oplus \overbrace{(x_{15} \oplus \underline{x_{16}} \oplus x_{17})}^{B_5} \oplus (x_{18} \oplus x_{19} \oplus x_{20})$$
$$P_2 = B_3 \oplus B_4 \oplus B_5 \oplus B_6$$
$$= (x_9 \oplus x_{10} \oplus x_{11}) \oplus (x_{12} \oplus x_{13} \oplus x_{14}) \oplus \overbrace{(x_{15} \oplus \underline{x_{16}} \oplus x_{17})}^{B_5} \oplus (x_{18} \oplus x_{19} \oplus x_{20})$$

S-a considerat cazul în care cifrele binare alese pentru formarea biţilor de control a parităţii $P_i, (i = 0,1,2)$ sunt cei daţi de codul Hamming SEC.

Partea de verificare a reziduului se obţine adunând cu sumatoare modulo 7 toţi *byte*-ii de informaţie.

Presupunem că în prezenţa unei erori singulare la nivelul bitului x_{16}, a cărui valoare corectă $x_{16} = 0$ a fost transformată în $x_{16}' = 1$, cuvântul X a devenit X'.

Calculând ecuaţiile de verificare a parităţii pentru cuvântul eronat se obţine sindromul de paritate: $S_p = (011)$ ce indică o eroare la nivelul *byte*-ului B_5 a lui X'. Pentru a identifica cifra binară eronată se calculează sindromul reziduului:

$$S_r = [X' - X]_7 = [[X']_7 - [X]_7]_7,$$

obţinându-se $[2^{17-1}]_7 = 2$, ceea ce indică ca şi cifră binară eronată pe cea dea-a doua din *byte*-ul localizat pe baza sindromului de paritate.

Pe de altă parte, dacă $x_{16} = 1$ a fost transformat în $x_{16}' = 0$, controlul reziduului indică:

$$[X' - X]_7 = [-2^{16}]_7 = 5 = [-2]_7,$$

ceea ce indică din nou cea de-a doua cifră binară a *byte*-ului localizat pe baza sindromului de paritate, dar tipul erorii indicate de astă dată este eroare de alterare a valorii 1 – referită prin 1-*eroare*.

Este important faptul că:

- sindromul de paritate este unic pentru fiecare din cei 7 *byte*-i, pe când sindromul reziduului este unic pentru fiecare bit şi pentru fiecare tip de eroare al *byte*-ului.
- dacă partea de informaţie utilă a cuvântului de cod este afectată de eroare, atunci eroarea este detectată atât de verificarea parităţii, cât şi de verificarea reziduului şi corelând informaţiile furnizate de cele două verificări, se poate face corecţia erorii.
- Dacă eroarea afectează P sau $[X]_7$, atunci doar unul din cele două sindromuri este diferit de zero.

Pe baza acestor observaţii se poate defini o procedură de localizare şi corecţie a erorii redată de următoarele ecuaţii:

$S_p = 0,\ S_r = 0$: nu există eroare

$S_p = 0,\ S_r \neq 0$: eroare în $[X]_7$

$S_p \neq 0,\ S_r = 0$: eroare în P

$S_p \neq 0,\ S_r \neq 0$: eroare în X

$S_p = i \neq 0,\ S_r = j \neq 0 \rightarrow B_{i+1}$ este eronat $(i = 1, 2, ..., 7)$

$j = 1$; primul bit eronat, eroare de tip 0→1

$j = 6$; primul bit eronat, eroare de tip 1→0

$j = 2$; al doilea bit eronat, eroare de tip 0→1

$j = 5$; al doilea bit eronat, eroare de tip 1→0

BUPT

$j = 4$; al treilea bit eronat, eroare de tip $0 \rightarrow 1$

$j = 3$; al treilea bit eronat, eroare de tip $1 \rightarrow 0$

Rezultă că orice eroare singulară la nivelul părţii de informaţie X poate fi detectată şi corectată.

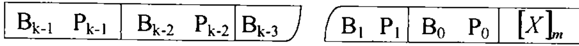


Fig.3. 25

Modificând structura codului combinat utilizat pentru codificare, în sensul calculării parităţii la nivelul fiecăruia din cei k *byte-i* (Fig.3. 25), se obţine o simplificare a implementării structurilor de însumare ce păstrează capabilităţile mai sus menţionate.
Astfel, pentru un operand X de n biţi $(n = k \cdot l)$, cuvântul de cod ce-i corespunde este $[X, P_X, [X]_m]$, unde P_X este format din k biţi de paritate, câte unul pentru fiecare *byte* al lui X, iar $[X]_m$ reprezintă reziduul lui X modulo m $(= 2^l - 1)$.

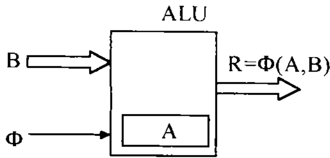


Fig.3. 26

Considerăm pentru început un model simplu al unei unităţi aritmetice, ca cel din Fig.3. 26. A este un operand intern, B este un operand de intrare (extern), şi Φ este codul operaţiei ce se execută, toate fiind specificate la momentul t. Se notează cu R rezultatul operaţiei Φ asupra operanzilor A şi B; R fiind disponibil o unitate de timp mai târziu.

În acest model avem: $R(t+1) = \Phi(A(t), B(t))$. Pentru a aplica codul combinat din Fig.3. 25 unităţii aritmetice, modelul acesteia trebuie modificat prin adăugarea unei unităţi de calcul a reziduului şi a unei unităţi de decodificare (Fig.3. 27).
Operaţia Φ din Fig.3. 26 este înlocuită în Fig.3. 27 cu operaţia combinată (Φ, Φ_m). Logica de paritate este proiectată în ALU astfel încât pentru fiecare operaţie Φ asupra lui A şi B se generează la ieşirea ALU perechea (R, P_R). P_R constă din k biţi de paritate numiţi biţi de paritate prezişi.
De asemenea, pentru fiecare operaţie Φ, se proiectează la nivelul unităţii de reziduu o operaţie Φ_m ce operează în paralel asupra reziduurilor $[A]_m$ şi $[B]_m$.

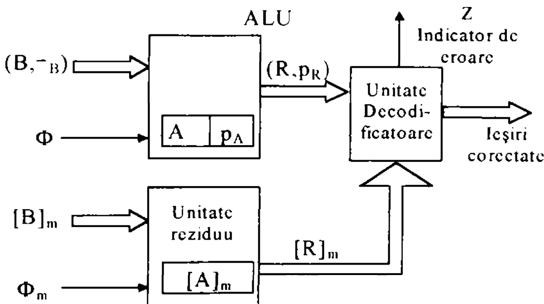


Fig.3. 27

BUPT

Rezultatul acestei operații este notat cu $[R]_m = \Phi_m([A]_m,[B]_m)$.
Pe baza acestei metode, dacă nu există erori, se obține la ieșirea circuitului un cuvânt de cod combinat. Totuși, datorită defectelor, ar putea apărea erori la nivelul acestui rezultat combinat.
Unitatea de decodificare generează sindroamele (s_p, s_r) pe care le utilizează pentru localizarea și corectarea erorilor.

BUPT

3.2 Reconfigurări BIST off line pentru testarea structurilor de însumare

În general generarea testelor şi comprimarea răspunsurilor se realizează prin utilizarea anumitor forme de registre de deplasare cu reacţie (*linear feedback shift register* - LFSR).

TPG-urile pot fi clasificate ca fiind de două tipuri:

1. Generatoare paralele de *pattern*-uri pseudoaleatoare (*pseudorandom pattern generator* - PRPG) - sunt dispozitive cu ieşiri multiple implementate în general cu LFSR [HORT89].
2. Generatoare de *pattern*-uri seriale (*shift register pattern generator* - SRPG) - sunt realizate cu LFSR autonom cu o singură ieşire.

ORA-urile pot fi şi ele clasificate în:

1. Registre de semnătură cu intrări paralele (*multiple-input signature register* - MISR)
2. Registre de semnătură cu o singură intrare (*single-input signature register* - SISR)

La nivelul unui circuit este foarte important de delimitat *hardcore*-ul acestuia, adică partea de circuite ce trebuie să fie operaţională pentru a fi posibilă autotestarea (*self-test* - ST). Este de dorit ca *hardcore*-ul să fie de complexitate minimă, întrucât testarea lui se face prin utilizarea unor echipamente exterioare de test, sau în caz contrar, el este făcut să fie autotestabil prin diferite forme de redondanţă (duplicarea, checker-ele cu autoverificare).

3.2.1 Arhitecturi BIST off-line generice

Aceste arhitecturi generale sunt aplicabile la nivelul CI sau a plăcilor de CI ce conţin blocuri de logică combinaţională interconectate cu celule de memorare.

Arhitecturile BIST *off-line*, indiferent de nivelul la care se referă pot fi clasificate după următoarele criterii [ZIQ83]:

1. Arhitecturile BIST *off-line* cu circuitele BIST centralizate sau distribuite
2. Arhitecturile BIST *off-line* cu elementele BIST separate sau încorporate

Elementele unei arhitecturi BIST sunt următoarele:

- generatoarele de *pattern*-uri de test
- analizoarele răspunsurilor de ieşire
- circuitul ce se testează
- sistemul de distribuţie (DIST) pentru transmiterea datelor de la TPG-uri la CUT şi de la CUT la ORA. Acesta constă în fire de interconectare directă, magistrale, multiplexoare şi căi de scanare (SP).
- un controler BIST ce controlează pe parcursul procesului de autotestare circuitele BIST şi CUT. În general acesta se află în întregime sau numai în parte în afara chip-ului.

În Fig.3. 28 este prezentată forma generală a unei arhitecturi BIST, în care, mai multe CUT-uri partajează circuitele TPG şi ORA. Acest tip de arhitectura este eficientă din

BUPT

punct de vedere al suprafeței consumate, dar inficientă din punct de vedere al duratei testării.

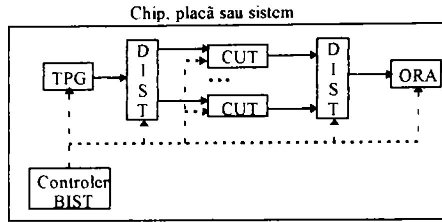


Fig.3. 28

Pe parcursul testării controlerul BIST realizează următoarele:

- selectează CUT ce se testează şi îi aplică procedura de testare
- inhibă *clock*-ul sistem şi controlează *clock*-ul de test
- comunică cu alte controlere de test, în general prin magistrale de test
- Controlează operația de autotest: alimentează registrele de deplasare cu datele de inițializare, contorizează numărul operațiilor de deplasare necesare operațiilor *scan*, şi urmăreşte numărul *pattern*-urilor de test ce au fost procesate

Informații legate de proiectarea controlerelor BIST se găsesc în [BREU88].
În Fig.3. 29 este dată arhitectura BIST distribuită, ce se caracterizează prin faptul că fiecare CUT are TPG şi ORA propriu. Diagnoza obținută este mai acurată, timpul de test este redus în comparație cu cel al primei arhitecturi, dar suprafața consumată pentru implementare este mai mare.
Arhitecturile din Fig.3. 28 şi Fig.3. 29 sunt exemple de arhitecturi separate, întrucât circuitele TPG şi ORA sunt exterioare CUT-ului, nefiind părți ale circuitului funcțional.

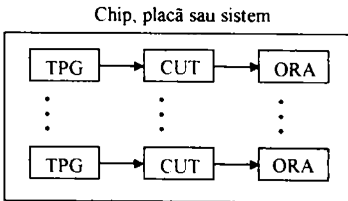


Fig.3. 29

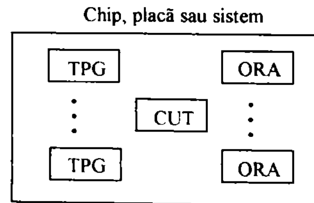


Fig.3. 30

În Fig.3. 30. avem forma generală a unei arhitecturi BIST distribuite şi încorporată. Aici TPG şi ORA sunt configurate din elementele funcționale ale CUT (de genul registrelor). În felul acesta creşte complexitatea părții de control, dar hardware-ul implicat este mai mic decât la arhitecturile separate.
Optarea pentru o anumită arhitectură BIST se face pe baza analizei următorilor factori:

1. Gradul de paralelism al testării. Arhitectura BIST distribuită asigură un grad de paralelism al testării mai mare decât cel al arhitecturii centralizate.
2. Acoperirea defectelor. Arhitecturile BIST distribuite asigură în general o acoperire mai mare a defectelor întrucât permite adaptarea circuitelor ORA şi TPG la cerințele CUT-ului ce se testează. De exemplu o tehnică BIST pentru testarea unui bloc combinațional nu este cea mai potrivită în cazul testării memoriei RAM.

BUPT

3. Gradul de împachetare. La nivel înalt BIST-ul centralizat devine mai natural. De exemplu testarea microdiagnostică este aplicabilă la nivelul la care există microcontrolere microprogramabile, când controlerul poate fi folosit să testeze multe componente ale sistemului.
4. Timpul de test. De obicei BIST distribuite duc la o reducere a timpului de test.
5. Constrângeri fizice. Proiectarea este influențată de factori ca dimensiunea, greutatea, puterea, etc. De obicei arhitecturile BIST încorporate și separate implică hardware mai mult și degradează performanțele.
6. Complexitatea unităților înlocuibile. Dacă cea mai mică unitate înlocuibilă este sistemul, atunci plăcile lui constituente nu este necesar să conțină circuite TPG și ORA, putându-se utiliza o arhitectură BIST centralizată.
7. Strategiile de testare și reparare la nivelul producătorului și în exploatare. Tipul ATE și măsura în care este acesta utilizat pentru testarea și diagnoza defectelor influiențează și este influențat la rândul său de BIST.
8. Degradarea performaței. Adăugarea elementelor BIST în anumite puncte critice din punct de vedere al timpului de operare poate duce la necesitatea reducerii frecvenței tactului.

Generatoarele de *pattern*-uri pseudoaleatoare se implementează cu ajutorul LFSR-urilor.

Mai jos sunt date diferitele forme de testare BIST împreună cu generatoarele de *pattern*-uri de test ce le sunt asociate:

• Testare exhaustivă	• Generatoare de *pattern*-uri de test exhaustive
• Testare pseudoaleatoare	• Generatoare de *pattern*-uri de test cu alocare de ponderi (*Weighted test generator* - WG)
	• Generatoare adaptive de *pattern*-uri de test (*Adaptive test generator* - AG)
• Testare pseudoexhaustivă	• Numărător comandat de sindrom (*Syndrome driver counter* - SDC) [BARZ181]
	• Numărător cu pondere constantă (*Constant weight counter* - CWC) [WANG87]
	• LFSR combinat cu registru de deplasare (*Combined LFSR and shift register*) [BARZ83] [TANG84]
	• LFSR combinat cu porți XOR (*Combined LFSR and XOR gates*)
	• LFSR condensat (*Condensed LFSR*) [WANG84], [WANG86]
	• LFSR ciclic (*Cyclic LFSR*) [WANG87].

3.2.2 Testarea exhaustivă

Testarea exhaustivă a unui circuit combinațional cu n intrări presupune aplicarea la intrarea acestuia a tututror celor 2^n combinații posibile ale valorilor sale de intrare. Ca și TPG pentru acest tip de testare se poate folosi fie un numărător binar, fie un LFSR autonom de lungime n, modificat de așa manieră încât să genereze inclusiv *pattern*-ul de test *(00 ... 0)* [TANG84B]. Testarea exhaustivă garantează detectarea tuturor defectelor detectabile care nu produc o comportare secvențială pentru circuitul combinațional testat. În general acestă metodă nu este aplicabilă pentru valori ale lui n ce depășesc 22 (limita lui n fiind de fapt determinată de frecvența tactului sistem). Pentru circuite ce au număr de intrări mare se apelează la alte tehnici de testare. De asemenea conceptul testării exhaustive nu este aplicabil, în general, circuitelor secvențiale.

BUPT

3.2.3 Testarea pseudoaleatoare

Testarea pseudoaleatoare realizează testarea unui circuit cu *pattern*-uri de test ce au multe din caracteristicile *pattern*-urilor aleatoare, dar care sunt generate în mod determinist, ele fiind deci repetabile. Generarea *pattern*-urilor de test pseudoaleatoare se poate face cu sau fără înlocuire.
Generarea cu înlocuire implică posibilitatea generării repetate a unui *pattern* de test, pe când cea fără înlocuire implică unicitatea fiecărui *pattern* de test. Trebuiesc generate toate cele 2^n *pattern*-uri. Ca şi generatoare de *pattern*-uri de test fără înlocuire pot fi folosite LFSR-urile autonome. Testarea pseudoaleatoare este aplicabilă atât circuitelor combinaţionale cât şi celor secvenţiale. Acoperirea defectelor se poate determina prin procesul de simulare a defectelor. Lungimea testelor este aleasă în funcţie de nivelul de acoperire al defectelor dorit.

3.2.3.1 Generarea aleatoare a testelor

În această secţiune se analizează metode statistice de:
- estimare a calităţii unui set de teste format din vectori aleatori, pe baza probabilităţilor de detecţie a defectelor.
- determinare a numărului de vectori generaţi aleator, necesari atingerii unei anumite calităţi a testării [WAGN87].

Iniţial presupunem că:
- vectorii de intrare sunt uniform distribuiţi, adică fiecare din cei 2^n vectori de intrare pentru un circuit cu n intrări primare sunt echiprobabili. Adică pentru fiecare PI avem probabilitate egală ca ea să fie 1L, respectiv 0L.
- vectorii de intrare sunt generaţi independent. Deci, acelaşi vector poate apare de mai multe ori în secvenţa de test generată. Totuşi, cele mai multe generatoare de vectori aleatori lucrează astfel încât un vector ce a fost generat să nu mai fie repetat.

Acest mecanism duce la seturi de teste mai mici decât cele generate sub presupunerea vectorilor independenţi.

Calitatea unui set de teste aleator trebuie să exprime nivelul de încredere acordat în interpretarea rezultatelor aplicării sale.
În mod cert, dacă cel puţin unul din testele aplicate cade, circuitul testat este defect. Dar, dacă toate testele trec trebuie determinat gradul nostru de încredere că circuitul este bun (nu este prezent nici unul din defectele singulare detectabile).
Acest nivel de încredere, poate fi măsurat prin probabilitatea ca testele aplicate să detecteze toate SSF-urile detectabile.

Astfel, pe o secvenţă de test de lungime N definim *calitatea sa de testare*, t_N, ca fiind probabilitatea ca toate defectele detectabile SSF să fie detectate prin aplicarea a N vectori aleatori. Cu alte cuvinte t_N este probabilitatea ca cei N vectori aleatori să conţină un set complet de teste pentru SSF.

O altă modalitate de măsurare a nivelului de încredere a rezultatelor testării aleatoare este considerarea defectelor individuale. Adică, dacă trec toate cele N teste aplicate, câtă încredere avem că circuitul nu conţine nici un defect f? Aceasta se măsoară prin probabilitatea d_N^f ca f să fie detectat (cel puţin odată) prin aplicarea a N

BUPT

vectori aleatori. d_N^f este numită probabilitatea ca defectul f să fie detectat prin aplicarea a N teste.

Calitatea detecției d_N a unei secvențe de test de lungime N este dată de:

$$d_N = \min d_N^f \tag{3.5}$$

Diferența dintre *calitatea testării* t_N unei secvențe de test de lungime N şi *calitatea detecției* ei, d_N, este că:

- t_N este probabilitatea de detecție a oricărui defect în timp ce
- d_N este probabilitatea de detecție a defectului cel mai greu de detectat

$$\Rightarrow t_N < d_N \tag{3.6}$$

În mod uzual nivelul de încredere de valoare minimă C este corelat cu calitatea detecției, astfel lungimea unui set de teste se alege astfel încât N să satisfacă relația:

$$d_N \geq C \tag{3.7}$$

Aceasta este justificat de faptul că o secvență de test suficient de lungă pentru a detecta cu probabilitate C defectul cel mai greu detectabil, va detecta oricare alt defect, f, cu probabilitatea:

$$d_N^f \geq C \tag{3.8}$$

Se defineşte e_N^f ca fiind probabilitate ca defectul f să rămână nedetectat prin aplicarea a N vectori aleatori. Avem:

$$d_N^f + e_N^f = 1 \tag{3.9}$$

Fie T_f setul tuturor testelor ce detectează f într-un circuit cu n PI.

Probabilitatea ca un vector aleator să detecteze defectul f este:

$$d_1 = |T_f| / 2^n \tag{3.10}$$

Ea reprezintă d_1^f, probabilitatea de detecție într-un pas a lui f; ne vom referi la aceasta prin: probabilitatea de detecție a lui f

Probabilitatea ca f să rămână nedetectat după un vector este:

$$e_1^f = 1 - d_f.$$

Întrucât vectorii de intrare sunt independenți, probabilitatea ca defectul f să rămână nedetectat după N paşi este:

$$e_N^f = (1 - d_f)^N \tag{3.11}$$

Dacă $d_{\min}$ este cea mai mică probabilitate de detecție a SSF-urilor din circuit, atunci pentru a se atinge o calitate a testării de valoare cel puțin C, este necesar un număr de teste, N, dat de următoarea relație:

$$1 - (1 - d_{\min})^N \geq C \tag{3.12}$$

Cea mai mică valoare N ce satisface această inegalitate este:

$$N_d = \left\lceil \frac{\ln(1 - C)}{\ln(1 - d_{\min})} \right\rceil \tag{3.13}$$

Pentru valori $d_{\min} << 1$, $\ln(1 - d_{\min})$ poate fi aproximat prin $d_{\min}$.

Calculul lui N presupune cunoscută probabilitatea de detecție $d_{\min}$ a defectului cel mai dificil de detectat.

BUPT

Dacă corelăm nivelul de încredere dorit C cu calitatea testării, trebuie ca N ales să satisfacă relația:

$$t_N \geq C \quad (3.14)$$

Savir și Bardell [SAVI84] au obținut următoarea formulă pentru estimarea limitei superioare a valorii celei mai mici pentru N ce atinge o calitate a testării de valoare cel puțin C:

$$N_t = \left[\frac{\ln(1-C) - \ln k}{\ln(1-d_{min})} \right] \quad (3.15)$$

unde k este numărul defectelor a căror probabilitate de detecție este cuprinsă în domeniul $[d_{min}, 2d_{min}]$.

Defectele a căror probabilitate de detecție este $2d_{min}$, sau mai mare, nu afectează în mod semnificativ lungimea de test cerută.

Tabel 3. 8 prezintă câteva valori pentru N_d și N_t pentru diferite valori pentru C și k în care d_{min} = 0,01 (valoare care trebuie înțeleasă ca și o valoare aproximativă mai degrabă decât ca una exactă)

C	N_d	k	N_t
0,95	300	2	369
		10	530
0,98	392	2	461
		10	622

Tabel 3. 8

De exemplu pentru a detecta orice defect cu o probabilitate de cel puțin 0,95 este necesar să se aplice cel puțin N_d = 300 de vectori aleatori

Dacă circuitul are numai $k = 2$ defecte ce sunt greu detectabile (adică probabilitatea ca ele să fie detectate este în domeniul [0,01, 0,02]) trebuiesc aplicați cel puțin N_t = 369 vectori asigurându-se astfel o probabilitate de detecție a oricărui defect de cel puțin 0,95.

Pentru $d_{min} = 0{,}001$, toate valorile N_d și N_t sunt crescute cu un factor de cel puțin 10.

Se concluzionează prin a afirma că pentru estimarea lungimii unei secvențe de test aleatoare, necesară pentru atingerea unei anumite calități a testării este necesar să se cunoască d_{min}, probabilitatea de detecție a defectului cel mai dificil.

În plus, pentru estimarea lungimii necesare pentru atingerea unei anumite calități a testării trebuie cunoscut și numărul k de defecte a căror probabilitate de detecție este apropiată de d_{min}.

Intr-un circuit combinațional cu mai multe ieșiri, având n numărul maxim de intrări primare ce alimentează o PO, limita inferioară pentru d_{min} este dată de următoarea relație:

$$d_{min} \geq 1/2^{n_{max}} \quad (3.16)$$

Limita inferioară se obține pentru PO al cărei con are cele mai multe PI.

Această limită inferioară este totuși de obicei prea conservativă întrucât dacă utilizăm o valoare a lui d_{min} dată de relația (3.13), de multe ori se obține $N_d > 2^n$, ceea ce înseamnă că sunt necesari mai mulți vectori pentru testarea aleatoare decât pentru testarea pseudoexhaustivă.

Determinarea probabilităților de detecție pe baza probabilităților de semnal este tratată în [ABRA96].

BUPT

3.2.3.2 Reducerea măsurilor de testabilitate la problema generală a probabilităților circuitelor logice

În această secțiune se prezintă o procedură de reducere a măsurilor de fiabilitate și testabilitate la problema generalizată a probabilităților circuitelor booleene [POPE94B].

Procedura se bazează pe utilizarea a 4 variabile logice, pe inserarea unor pseudoporți în scopul reducerii diverselor măsuri la problema generalizată a probabilităților și pe construirea unor tabele de funcționare bazate pe acele 4 variabile pentru porțile circuitului.

Pentru analiză se adoptă modelul SSF.

Admitem că C este un circuit boolean realizat cu porți AND, OR și NOT, având intrările $x_1 \dots x_n$. Fie l o linie a lui C. Linia l este blocată la b, $(b \in \{0,1\})$, dacă l este menținută permanent la valoarea b în prezența defectului F. Un vector de intrare este un vector de test pentru defectul F dacă și numai dacă există o ieșire Y a lui C astfel încît valoarea lui Y la aplicarea lui T în absența defectului este a, $(a \in \{0,1\})$ și în prezența defectului este b, $(b \in \{0,1\})$, unde $a \neq b$.

Pentru calculul mărimilor ce interesează presupunem următoarele:

1. Pentru toate liniile din circuit l∈C, se cunosc probabilitățile: $q_b(l) = \Pr\{l\ s-a-b\}$ unde $b \in \{0,1\}$,
2. Liniile de intrare pot lua valorile 0 și 1 cu probabilitate 1/2.
3. Seturile de evenimente $\{x_i = 0\}$, $\{x_i = 1\}$ sunt mutual independente.
4. Pentru orice două linii $l, p \in C$ și oricare ar fi $a, b \in \{0,1\}$, evenimentele l s-a-a și p s-a-b sunt independente (a și b nu neapărat distincte).

Problema generalizată a probabilităților – PGP – pentru circuite booleene se definește astfel: se dau setul de valori logice $V = \{V_1, \dots, V_k\}$ unde $\{0,1\} \in V$, setul de porți Q definite peste V, $Q = \{Q_1, \dots, Q_t\}$unde {AND, OR, NOT} ∈ Q, circuitul aciclic C compus din porți din Q cu o ieșire Y și n intrări $x_1, x_2, \dots, x_n$ și probabilitatea fiecăruia din evenimentele mutual independente $E\ \ \{\{x_i\ \ c\}, c \in V, 1\ i\ n\}$ și se pune problema calculului probabilității evenimentelor $\{y = a\}, a \in V$.

[CHAK91] definește următoarele măsuri de fiabilitate:

1. Calculul *probabilității unui semnal* (CPS) se referă la calculul probabilității ca un vector aleator de intrare să seteze o ieșire Y a lui C la 1. [MARK87] și [KAPU92] dau o procedură de calculare a limitei inferioare pentru probabilitatea de semnal. Acest calcul (abordat în [POPE94A]) este o instanță a problemei generalizate a probabilităților pentru circuite booleene și problema este echivalentă cu calculul măsurilor de testabilitate utilizate în testarea circuitelor booleene.
2. Calculul *probabilității de detecție* (CPD) se referă la a calcula probabilitatea ca un vector de intrare ales aleator, să fie un vector de test pentru defectul F de blocare la 0 sau 1 a liniilor din circuitul C. CPD este folosit pentru determinarea lungimii *pattern*-urilor aleatoare de test în testarea circuitelor booleene.
3. Calculul *fiabilității semnalului* (CFS), realizează calculul pentru fiecare ieșire Y_i, $(i = 1 \dots k)$ a lui C, a probabilității următoarelor evenimente:

   Y_i este corect la 1

   Y_i este corect la 0

   Y_i este incorect la 1

BUPT

Y_i este incorect la 0

Probabilitatea acestor 4 evenimente reprezintă fiabilitatea semnalului Y_i.

Calculul probabilităţii de semnal

Este evident că CPS este o instanţă a lui PGP.

Calculul acestei mărimi se poate face şi pe baza utilizării cuburilor Miller şi a operatorilor cu aceste cuburi ce sunt prezentaţi în [POPE95C].

Răspunsul circuitului fără defect	Răspunsul circuitului defect	Codificare
0	0	0
0	1	$\bar{d}$
1	0	d
1	1	1

Tabel 3. 9

Reducerile la PGP pentru celelalte probleme se bazează pe utilizarea a 4 valori logice $\{0,1,d,\bar{d}\}$, definite în Tabel 3. 9.

O asemenea codificare cu 4 vari-abile poate fi folosită în scrierea tabelelor de adevăr a porţilor AND, OR şi NOT, tabele ce se utilizează pentru calculul vec-torilor de test pentru defectele de tip s-a-0 sau s-a-1; un vector de intrare T, aplicat circuitului devine vector de test pentru circuit, dacă orice ieşire a circuitului este setată în prezenţa defectului F la d sau $\bar{d}$. Pentru modelarea defectelor de tip s-a-1 (sau s-a-0) de pe liniile circuitului, se inserează pe aceste linii pseudoporţi α (respectiv β) (Fig.3. 31), cărora le corespund tabelele de funcţionare din Tabel 3. 10 (respectiv Tabel 3. 11).

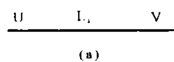
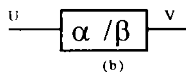



Fig.3. 31

α	
U	V
0	$\bar{d}$
1	1
d	1
$\bar{d}$	$\bar{d}$

Tabel 3. 10

β	
U	V
0	0
1	d
d	d
$\bar{d}$	0

Tabel 3. 11

γ		
γ_1	γ_2	γ_3
X	d	d
X	$\bar{d}$	d
d	X	d
$\bar{d}$	X	d
0	0	0
0	1	0
1	0	0
1	1	0

Tabel 3. 12

Calculul probabilităţii de detecţie

Se realizează utilizând pseudoporţi γ avînd tabelul de adevăr din Tabel 3. 12.

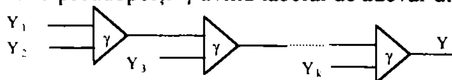


Fig.3. 32

Pentru reducere se procedează astfel:

- pentru toate liniile din circuit s-a-1 se inserează pseudoporţi α, iar pentru cele ce sunt s-a-0 se inserează pseudoporţi β;
- se ataşează lanţul de pseudoporţi γ (Fig.3. 32) la cele k ieşiri ale circuitului C modificat prin inserarea pseudoporţilor α şi β

Acest lanţ de pseudoporţi γ asigură valoarea d la ieşire, dacă oricare din intrările lanţului are d sau $\bar{d}$ la intrare.

BUPT

Bazându-ne pe aceste modificări din circuit, problema CPD a oricăruia din defectele F considerate, se reduce la calculul $P_r\{Y = d\}$

Calculul fiabilității semnalului

In [POPE94C] este prezentat un algoritm algebric pentru calculul fiabilității. Dacă se dorește calculul fiabilității semnalelor de ieșire ale unui circuit logic, se procedează la inserarea unor pseudoporți δ pe fiecare linie l susceptibilă la defect (pentru care $q_1(l) + q_0(l) \neq 0$).

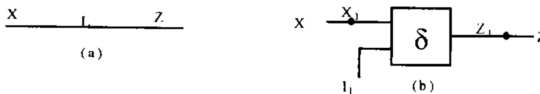


Fig.3. 33

O pseudopoartă δ prezintă pseudointrarea I_l, intrarea X_l și ieșirea Z_l (Fig.3. 33).

δ			
eveniment	I_l	X_l	Z_l
fără defect	d	0	0
	d	1	1
	d	d	d
	d	$\bar{d}$	$\bar{d}$
s-a-0	0	0	0
	0	1	d
	0	d	d
	0	$\bar{d}$	0
s-a-1	1	0	d
	1	1	1
	1	d	1
	1	$\bar{d}$	$\bar{d}$

Tabel 3. 13

Tabelul de funcționare al pseudoporții δ este prezentat în Tabel 3. 13:

Urmărind CFS pentru o ieșire Y_i, se face inserarea pseudoporților δ pe toate liniile ce prezintă $(q_1(l) + q_0(l) \neq 0)$, fiecare pseudopoartă avînd pseudointrarea asociată ei: I_l. Pentru fiecare pseudointrare I_l seminficația va-lorilor sale este dată în Tabel 3. 14.

În acest sens, evenimentele:

Y_i = 0 este echivalent cu evenimentul Y_i e corect la 0

Y_i = 1 este echivalent cu evenimentul Y_i e corect la 1

Y_i = d este echivalent cu evenimentul Y_i e incorect la 0

$Y_i = \bar{d}$ este echivalent cu evenimentul Y_i e incorect la 1

Eveniment	Eveniment echivalent	Probabilitatea evenim
$\{I_l=1\}$	linie l este s-a- 1	$q_1(l)$
$\{I_l=0\}$	linie l este s-a- 0	$q_0(l)$
$\{I_l=d\}$	linie l este fără defect	$1-q_1(l)-q_0(l)$

Tabel 3. 14

3.2.3.3 Algoritm de rezolvare a Problemei generalizate a probabilităților pentru circuite logice

Fie $< R_1,...,R_k >$ o ordonare topologică a punctelor de fanout ale unui circuit logic cu ieșirea X, la care valorile logice ale fiecărei linii de circuit $\in V$. Definim următoarele elemente:

- *H-Asignarea.* $< R_1 = a_1,...,R_k = a_k >$ este o H-Asignare dacă și numai dacă pentru toți $i \in (1,k)$, valorile $a_i \in V$.

BUPT

- *H-Asignare consistentă.* O H-Asignare este consistentă dacă şi numai dacă $\Pr\{R_1 = a_1,...,R_k = a_k\} > 0$.
- *H-Asignare parțial consistentă.* $< R_1 = a_1,...,R_i = a_i >$ este o H-Asignare parțial consistentă dacă şi numai dacă:
  1. $i \le k$
  2. $a_j \in V$ pentru $1 \le j \le i$
  3. $\Pr\{R_1 = a_1,...,R_i = a_i\} > 0$.

În prezentarea algoritmului se face uz de următoarele notații:

C(X)	Circuitul ce are ieşirea X.
X^T	Un arbore de subcircuit ce alimentează X
X	Iesirea lui X^T
$< R_1,...,R_k >$	O ordonare topologică a punctelor de fan-out pentru C(X)
FP	Setul de puncte de fanout ale circuitului împreună cu ieşirea circuitului
R^T	Arbore de circuit maximal (ACM) cu iesirea R, pentru fiecare $R \in FP$.

Pentru obținerea relațiilor ce stau la baza algoritmului considerăm $H_1, H_2,...,H_N$ ca fiind un set de H-Asignări ale circuitului cu ieşirea X. Atunci, pentru toate valorile $a \in V$ avem:

$$\Pr\{X = a\} = \sum_{i=1}^{N} \Pr\{(X = a) \wedge H_i\} x \Pr\{H_i\} \tag{3.17}$$

Fie $H_i =< R_1 = a_1,...,R_k = a_k >$. Atunci:

$$\begin{aligned}\Pr\{H_i\} &= \Pr\{R_1 = a_1,...,R_k = a_k\} = \\ &= \Pr\{R_k = a_k \| R_1 = a_1,...,R_{k-1} = a_{k-1}\} x \Pr\{R_1 = a_1,...,R_{k-1} = a_{k-1}\} = ... = \\ &= \Pr\{R_1 = a_1\} x \prod_{j=2}^{k} \Pr\{R_j = a_j \| R_1 = a_1,...,R_{j-1} = a_{j-1}\} = \\ &= \beta_1 x \prod_{j=2}^{k} \beta_j\end{aligned} \tag{3.18}$$

unde $\beta_1 = \Pr\{R_1 = a_1\}$

şi pentru toți $j \ge 2$ avem $\beta_j = \Pr\{R_j = a_j \| R_1 = a_1,...,R_{j-1} = a_{j-1}\}$

Algoritmul H-ASSIGN utilizează relațiile (3.17) şi (3.18), şi începe prin enumerarea unei H-Asignări parțial consistente de lungime 1; în continuare, gaseşte extensii pentru asignarea parțială până a enumerat o H-Asignare consistentă.
În pseudocodul ce urmează, probabilitatea de distribuție a punctului Y este memorată într-un tablou 1-dimensional PY de lungime *k*; şi pentru toți $1 \le i \le k$, $PY[i] = \Pr\{Y = i\}$.

```
PROGRAM H-ASSIGN;
TYPES
        TREE_PTR: Pointer la un arbore de subcircuit
        LOGIC_VALUES: {V1,...,Vk}
        MTS_LIST: constă din două câmpuri
                TR: TREE_PTR;
                NEXT: CONST_LIST_PTR;
```


BUPT

__MTS_LIST_PTR__:Pointer la MTS_LIST;
__CONST_LIST__: constă din două câmpuri
VAL: LOGIC VALUES;
NEXT: CONST_LIST_PTR;
__CONST_LIST_PTR__: Pointer la CONST_LIST;
Variabile globale
***PROB_DIST**. Un ARRAY[1..k] de reali. Conține probabilitatea de distribuție a lui Y ce se calculează, unde Y este ieşirea circuitului.*
***ASSIGN**. O variabilă de tip CONST_LIST_PTR*
Asignarea parțială ce se trasează, este menținută ca o listă şi ASSIGN pointează la capul listei. Dacă ASSIGN conține I intrări, şi câmpul VAL al intrării j este a_j, atunci $< R_1 = a_1, \ldots, R_i = a_i >$ este asignarea parțial consistentă ce este în curs de trasare.
***BEGIN** (* Main *)*
__FP__ O variabilă de tip MTS_LIST_PTR
Presupunem că FP pointează la capul listei ce conține setul de ACM ai circuitului Ordonarea totală a setului ce constă din punctele de fanout şi ieşirea circuitului, este $< R_1, \ldots, R_k, Y >$. Prin urmare, campul TR al intrării i a lui FP pointează la subcircuitul R_i^T ce are iesirea R_i.

```
    FOR I = 1 TO K DO PROB_DIST[i]=0;
    ASSIGN=NIL;
    CALL C_PDIST(FP,1,ASSIGN);
END

PROCEDURE C_PDIST (PTR:MTS_LIST_PTR; PR_ASSIGN:REAL; ASSIGN: CONST_LIST_PTR);
EXT_ASSIGN: CONST_LIST; P_DIST: ARRAY[1..k] of REAL;
BEGIN
    IF PTR ≠ NIL THEN
      BEGIN
        TREE(PTR, P_DIST, ASSIGN);
        IF(PTR.NEXT = NIL) THEN
          FOR I=1 TO K DO
          PROB_DIST[I] = P_DIST[I] • PR_ASSIGN + PROB_DIST[I]
        ELSE
        BEGIN
          FOR J=1 TO K DO
          BEGIN
           IF (P_DIST[j]>0) THEN
             BEGIN
               EXT_ASSIGN.VAL= V[J]
               Adaugă EXT_ASSIGN la sfârşitul lui ASSIGN
               C_PDIST(PTR.NEXT,PR_ASSIGN • P_DIST[J],ASSIGN);
               Extrage EXT_ASSIGN din ASSIGN;
             END
          END
        END
      END
END

PROCEDURE TREE (PTR:MTS_LIST_PTR; PR_DIST:ARRAY[1..K] OF REAL;
      ASSIGN:CONST_LIST_PTR);
BEGIN
```

***IF** ASSIGN = NIL **THEN** PTR este FP si PTR.TR pointează la R_1^T care este un circuit fără fanout. Probabilitatea de distribuție R_1 este calculată printr-o traversare directă a lui R_1^T şi este returnată programului apelant în PR_DIST.*
***IF** ASSIGN ≠ NIL **then** realizează următoarele:*

BUPT

- *Presupunem că la apelul procedurii PTR pointează la intrarea I,* $(I > 0)$ *a listei pointate de FP; adică PTR.TR pointează la subcircuitul* R_i^T.
- *ASSIGN reprezintă asignarea parțială* $< R_1 = a_1 ..., R_{I-1} = a_{I-1} >$.
- *Se calulează, pe baza relațiilor (3.17) și (3.18), pentru toate valorile* $a \in V$ *probabilitatea* $\Pr\{R_I = a\}$ *și se returnează programului apelant distribuția probabilităților lui* R_I.

END

Din păcate anumite circuite conțin defecte *random-pattern-resistant*, şi astfel pentru asigurarea unei înalte acoperiri a defectelor în aceste situații sunt necesare teste de lungime mare. LFSR-urile produc *pattern*-uri de test la care probabilitatea de apariție a valorilor *0* sau *1* pe fiecare linie de ieşire este *0.5*. Uneori este necesară generarea unor *pattern*-uri de test ce să aibă diferite distribuții pentru valorile 0 sau 1 ale liniile de test, utilizându-se în acest sens:

1. Generarea testelor cu alocare de ponderi (*Weighted test generation* - WTG). Pentru generare se utilizează un generator realizat cu ajutorul unui LFSR autonom la care se adaugă logică combinațională. La testarea unui circuit stimulat cu astfel de vectori este necesară o procedură de preprocesare care să determine unul sau mai multe seturi de ponderi (*weights*) ce să permită testarea cât mai eficientă a diferitelor părți ale unui circuit. Aceste ponderi stau la baza construirii generatoarelor de astfel de *pattern*-uri de test.
2. Generarea adaptivă a testelor. Generarea adaptivă a testelor utilizează şi ea WTG. În acest caz rezultatele simulării defectelor sunt utilizate pentru modificarea ponderilor, rezultând, prin urmare, una sau mai multe distribuții pentru *pattern*-urile de test. Pe baza acestor distribuții se face proiectarea TPG-urilor, foarte eficiente din punct de vedere al lungimii testelor, dar complexe din punct de vedere al hardware-ului implicat.

3.2.4 Testarea pseudoexhaustivă

Testarea pseudoexhaustivă prezintă multe din avantajele testării exhaustive, şi de obicei implică mai puține teste [UDEL86]. In [RAJS93] este dată o procedură recursivă de generare pseudoexhaustivă a testelor. La baza acestei testări stă partiționarea prin diferite forme a circuitului testat, fiecare segment astfel obținut fiind testat exhaustiv [PATA83] [JONE89A]. Segmentul este de fapt un subcircuit al circuitului C (segmentele nu trebuie să fie disjuncte). Există mai multe forme de segmentare, şi anume:

1. Segmentare logică
    a. Segmentare prin activarea căilor
    b. Segmentare conică
2. Segmentare fizică

De multe ori, când se aplică acestă metodă de testare unui circuit cu *n* intrări, este posibilă reconfigurarea liniilor de intrare astfel încât să fie necesară generarea testelor doar pe *m* linii ($m < n$) [SALU88] [JONE89B]. Aceste *m* linii ce comandă prin fanout cele *n* linii ale circuitului sunt numite semnale de test ale circuitului respectiv. Testarea pseudoexhaustivă poate fi deseori realizată cu ajutorul *pattern*-urilor de test de pondere constantă.

BUPT

3.2.4.1 Pattern-uri de pondere constantă

Se consideră $n,k \in N$, cu $k \leq n$. Fie T un set de vectori binari cu n componente. Se spune că T acoperă exhaustiv toate cele k-subspații dacă pentru toate subseturile de k poziții de bit toate cele 2^k *pattern*-uri binare apar cel puțin odată printre vectorii binari de n componente ai lui $|T|$. Este evident că dacă $|T|_{\min}$ este cea mai mică dimensiune a unui asemenea set T, atunci $2^k \leq |T|_{\min} \leq 2^n$.

Un vector binar se zice că este de pondere k dacă el conține k unități. Există C_n^k vectori binari cu n componente ce au ponderea k.
Trang și Won [TANG83] au demonstrat următoarele teoreme:

Teorema 3.3

Pentru n și k dați, T acoperă exhaustiv toate k-subspațiile binare dacă el conține toți vectorii binari cu n componente ce au ponderea w, astfel încât $w = c \bmod (n-k+1)$, pentru un c întreg constant, unde $0 \leq c \leq n-k$.

Corolar

Pe baza teoremei 3.3 se obțin $(n-k+1)$ seturi de soluții disjuncte T_i, $0 \leq i \leq n-k$ ce partiționează în clase disjunte setul tuturor celor 2^n vectori binari de n componente.

Prin urmare cel mai mic astfel de set va fi mai mic decât media acestor seturi, putându-se determina o limită superioară a dimensiunii $|T|_{\min}$, care este: $|T|_{\min} \leq \dfrac{2^n}{n-k+1} = B_n$.

Teorema 3.4

Fie T_i un set generat prin aplicarea teoremei 3.3. Atunci T_i este minimal (adică el nu conține un subset ce să testeze și el exhaustiv cele k-subspații) dacă $c \leq k$ sau $c = n-k$.

Pentru valori ale lui k apropiate de n, limita superioară a lui $|T_{\min}|$ este apropiată de B_n, iar pentru valori ale lui k mici ea este mult diferită de B_n. Pe de altă parte B_n are o variație exponențială în raport cu n și independentă de k.

Pentru $k \leq n/2$, dimensiunea setului minim apare când $w = \lfloor k/2 \rfloor$ și $\lfloor k/2 \rfloor + (n-k+1)$, rezultând $|T_{\min}| = C_n^{\lfloor k/2 \rfloor} + C_n^{k - \lfloor k/2 \rfloor - 1}$.

Sunt importante următoarele două situații ce pot apărea:

1. $n = k$ Pentru acest caz avem $w = 0 \bmod 1$, deci $w = 0,1,2,...,n$. Prin urmare, T_0 este dat de mulțimea tuturor celor 2^n vectori binari.

$T_0=$ (paritate pară)

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

$T_1=$ (paritate impară)

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

BUPT

2. $n = k + 1$ Pentru acest caz avem $w = c \bmod 2$, deci, T_0 este dat de toţi vectorii binari de paritate pară, iar T_1 de toţi cei de paritate impară. Mai jos este ilustrată această situaţie pentru cazul când $n = 4$ şi $k = 3$

Teorema 3.5

Într-o matrice cu 2^k rânduri distincte, fiecare rând având $(k+1)$ valori binare, în care fiecare rând are aceeaşi paritate, fiecare set de k coloane are toate ce 2^k combinaţii ale celor k valori.

3.2.4.2 Segmentarea logică la nivelul structurilor de însumare

Segmentare prin activarea căilor

Anumite circuite pot fi segmentate pe baza conceptului de activare a căii. În Fig.3. 34 este dat un exemplu trivial de segmentare bazată pe această metodă. Pentru testarea exhaustivă a circuitului C_1 se aplică intrării A 2^{n1} *pattern*-uri de test, în timp ce intrarea B este setată la o valoare astfel încât D=1. În felul acesta a fost stabilită o cale de activare de la C la F. Circuitul C_2 este testat şi el într-o manieră similară. Poarta AND este testată şi ea complet prin acest proces. Prin urmare testarea completă a acestui circuit se realizează cu numai $2^{n1} + 2^{n2} + 1$ *pattern*-uri de test (în loc de 2^{n1+n2} *pattern*-uri).

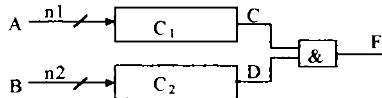


Fig.3. 34

Segmentarea conică

Un circuit cu m ieşiri este segmentat logic în m conuri (*cones*), fiecare con constând din întrega logică asociată unei ieşiri. Fiecare con este testat exhaustiv şi toate conurile sunt testate concurent.Această formă de testare a fost propusă de către McCluskey [MCCL84] şi este numită *verification testing*.

Considerăm un circuit combinaţional C cu intrările $X = \{x_1, x_2, ..., x_n\}$ şi ieşirile $X = \{x_1, x_2, ..., x_n\}$. Fie $y_i = f_i(X_i)$, unde $X_i \subseteq X$ şi fie $w = \max_i \{|X_i|\}$. O *verification testing* produce toate cele 2^w *pattern*-uri de intrare pe fiecare din cele C_n^w subseturi de n intrări ale circuitului C. Circuitul ce se testează este notat prin (n, w)-CUT, unde $w < n$.

Întrucât la nivelul sumatoarelor cu operanzi de n biţi, conul asociat ieşirii S_n prezintă ca şi variabile de intrare toate variabilele de intrare ale sumatorului, obţinem $w = 2 \cdot n + 1$; adică (n, n) -CUT. Tehnica *verification testing*, la nivelul acestora va degenara într-o testare exhaustivă a sumatoarelor, impracticabilă pentru valori mari ale lui n.

În continuare, se vor determina pe baza segmentării logice bazată pe activarea căilor, seturile complete de teste pentru detecţia defectelor singulare de tip stuck-at la 0 sau 1 la nivelul sumatoarelor ce prezintă prin structura lor o partiţionare în blocuri.

3.2.4.2.1 Testare bazată pe partiţionare logică prin activarea căilor la CSkA

Aplicarea acestei tehnici la nivelul sumatoarelor CSkA de n biţi presupune testarea individuală în manieră exhaustivă a fiecărui bloc de dimensiune m_i, prin aplicarea

BUPT

tuturor celor $2^{2 \cdot m_i}$ *pattern*-uri de intrare atât cu C_{-1}=0, cât şi cu C_{-1}=1. Pe durata testării unui anumit bloc, celelalte blocuri vor avea pe intrare $A_i B_i = 01$, combinaţie ce asigură activarea căilor în vederea segmentării logice. În plus, pentru testarea completă a sumatorului trebuie adăugat *pattern*-ul de intrare 00 00 .. 00 cu C_{-1}=0.

Pentu generarea acestor teste, există, în principiu, două soluţii de implementare a TPG-urilor:

1. bazate pe utilizarea PROM-urilor
2. bazate pe utilizarea automatelor secvenţiale sincrone realizate cu numărătoare (sau LFSR-uri), şi multiplexoare.

1. Implementarea TPG pe bază de PROM simplifică testarea CSkA ce au blocuri de dimensiuni diferite. Această implementare presupune utilizarea unui numărător NUMPROM ce permite adresarea PROM-ului şi a unor multiplexoare pentru selectarea intrărilor sumatorului. NUMPROM va număra de la 0 la NT, unde NT reprezintă numărul total al testelor pentru testarea completă a sumatorului; deci $NT = \sum_{i=1}^{nr.bloc} \left(2 \cdot 2^{2 \cdot m_i}\right) + 1$. Numărarea se va face sub condiţia: $Count = CLK \cdot Test$, iar iniţializarea o va face $\overline{CLR} = \overline{Test}$. Structura de însumare de n biţi va avea intrările conectate la n circuite ca cele din Fig.3. 35; Notaţia FN din Fig.3. 35 indică conexiunea la funcţionare normală a structurii de însumare.

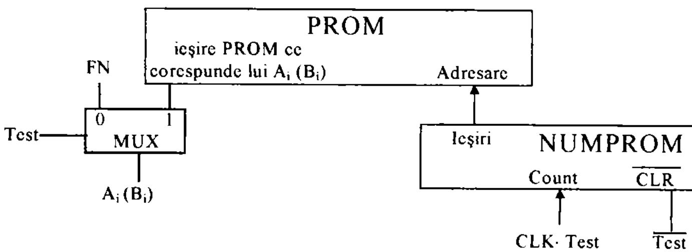


Fig.3. 35

Pe de altă parte, sinteza TPG realizată cu numărătoare şi multiplexoare presupune utilizarea următoarelor elemente:

Numărătorul NB, pentru identificarea blocului testat, ce are $\log_2 M$ ranguri, unde M reprezintă numărul blocurilor sumatorului, şi care numără sub condiţia: $Count = CLK \cdot EndNUM_i \cdot Test$ de la 0 la $(M-1)$.

Semnalul $EndNUM_i$ este un semnal derivat prin întârziere cu un tact din semnalul EndNUM, ce indică sfârşitul numărării stărilor la nivelul blocurilor testate. Deci, $EndNUM_i = \sum_{i=1}^{M} \left(NB_i \cdot \left(combinatie\ binară\ ce\ identifică\ ultimul\ test\ al\ blocului\ i\right)\right)$.

Este necesară această întârziere cu un tact pentru a asigura testul 00 00 ... 00 cu C_{-1}=0; acesta va fi generat după fiecare testare completă a unui bloc (se creşte numărul testelor aplicate cu M-1 în favoarea unei simplificări considerabile a TPG. Iniţializarea numărătorului o face $\overline{Test}$.

Numărătorul NUM ce generează, când $Test = 1$, testele pentru blocul ce se testează, el fiind iniţializat sub condiţia:

BUPT

$$\overline{Test} + \sum_{i=1}^{M} \left(NB_i \cdot comb.bin.deultim\ test\ pt.bloc\ i \right).$$

În cazul utilizării blocurilor de dimensiune variabilă, numărătorul trebuie să funcţioneze în regim comandat, ceea ce complică sinteza acestuia. NUM are un număr de $2 \cdot m_{max}$ ranguri. Comanda lui C_{-1} se face cu rangul mai semnificativ de numărare ce corespunde fiecărui bloc în parte, adică prin semnalul $\sum_{i=1}^{M} \left(NUM_j \cdot NB_i \right)$.

- Multiplexoare MUX_{Ai} şi MUX_{Bi} pentru comanda intrărilor sumatorului. Considerând biţii A_i şi B_i ce aparţin blocului *j*, multiplexoarele asociate rangului *i* al sumatorului, împreună cu comenzile asociate lor sunt date în Fig.3. 36. FN indică conexiunea la funcţionare normală, iar NUM_{2i}, conexiunea la rangul corespondent al numărătorului NUM.

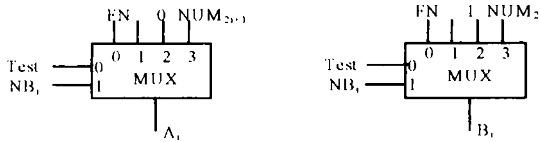


Fig.3. 36.

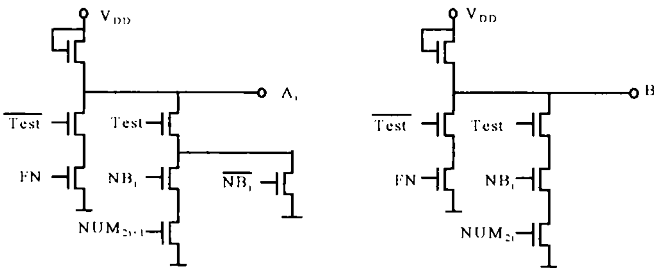


Fig.3. 37

Pentru creşterea vitezei de operare la nivelul acestor multiplexoare se apelează la metode de intervenţie în circuit la nivelul integrării acestuia. În Fig.3. 37. este ilustrată implementarea n-MOS a acestor multiplexoare. Conceptul poate fi aplicat şi altor tipuri de operatori, precum şi altor tehnologii.

Aceleaşi teste sunt valabile şi pentru sumatorul CSkA pe mai multe niveluri. În concluzie, Tabel 3. 16 prezintă numărul testelor determinate prin aplicarea acestei tehnici pentru sumatoare CSkA de diverse dimesiuni, la care blocurile au dimensiune constantă.

3.2.4.2.2 Testare bazată pe partiţionare logică prin activarea căilor la structurile de însumare cu anticiparea transportului

Datorită structurii complexe a sumatorului CLA în formă pură, nu este realizabilă o partiţionare logică a acestuia. Prin urmare testarea lui nu se poate face decât fie în manieră exhaustivă, fie în manieră deterministă. Pentru testarea exhaustivă a unui CLA cu *m* ranguri este necesară generarea a $2^m \cdot 2$ teste.
Jumătate din teste se aplică cu C_{-1}=0 şi jumătate cu C_{-1}=1.

BUPT

Testare bazată pe partiţionare logică prin activarea căilor la RCLA şi BCLA

Considerând un sumator RCLA pentru operanzi cu n biţi, organizat în N blocuri, fiecare bloc având m ranguri $(N = n/m)$, testarea exhaustivă a fiecărui bloc se face prin aplicarea celor 2^{2m} teste de intrare atât cu C_{-1}=1 cât şi cu C_{-1}=0.

Celelalte blocuri în afara celui testat vor avea pe intrare *pattern*-ul 01 01 ... 01, permiţându-se astfel activarea căilor la nivelul circuitului. La aceste $N \cdot 2 \cdot 2^{2m}$ teste se adaugă testul: 00 00 .. 00 ce are C_{-1}=0, rezultând pentru testarea completă a sumatorului un necesar total de teste de valoare: $(N \cdot 2 \cdot 2^{2m} + 1)$. La fel se face şi testarea sumatorului BCLA.

Testare bazată pe partiţionare logică prin activarea căilor la SBCLA, ISBCLA şi SRCLA

Considerând un sumator SBCLA pentru operanzi cu n biţi, organizat în N superblocuri, fiecare superbloc având M blocuri cu m ranguri $(N = n/M)$, testarea fiecărui bloc se face exhaustiv prin aplicarea celor 2^{2m} teste de intrare atât cu C_{-1}=1, cât şi cu C_{-1}=0. Pentru fiecare din aceste teste, blocurile mai semnificative blocului testat vor avea pe intrare *pattern*-urile din Tabel 3. 15, iar cele mai puţin semnificative vor avea *pattern*-ul: 01 01 .. 01.

Primul bloc al SBCLA	Al doilea bloc al SBCLA		Al doilea bloc ce urm. bloc testat	Primul bloc ce urm. bloc testat
01 01 ... 01	01 01 ... 01	...	01 01 ... 01	01 01 ... 01
01 01 ... 01	01 01 ... 01	...	01 01 ... 01	*11 11 ... 11*
01 01 ... 01	01 01 ... 01	...	*11 11 ... 11*	01 01 ... 01
...				
01 01 ... 01	*11 11 ... 11*	...	01 01 ... 01	01 01 ... 01
11 11 ... 11	01 01 ... 01	...	01 01 ... 01	01 01 ... 01

Tabel 3. 15

La aceste teste se adaugă testul ce aplică fiecăruia din cele $M \cdot N$ blocuri *pattern*-ul 00 00 ... 00 având C_{-1}=0. În felul acesta a fost realizată activarea căilor la nivelul blocurilor netestate, precum şi testarea completă a logicii asociată superblocului, rezultând pentru testarea completă a sumatorului SBCLA un necesar total de teste de valoare:

$$\left((n/m)+(n/m)^2-\sum_{k=1}^{n/m}k\right)\cdot 2\cdot 2^{2m}+1=\left(\frac{(n/m)\cdot(n/m+1)}{2}\right)\cdot 2\cdot 2^{2m}+1.$$

O altă posibilitate de partiţionare logică prin activarea căilor la acest tip de structuri de însumare este cea bazată pe testarea exhaustivă a superblocurilor, cu activarea căilor pentru superblocurile netestate, cărora li se aplică pe intrare *pattern*-ul 01 01 ... 01. Acestor teste li se adaugă testul ce aplică pe intrările sumatorului *pattern*-ul 00 00 ... 00 având C_{-1}=0, realizându-se astfel testarea completă a logicii de superbloc.

Numărul testelor rezultate pentru testarea completă a sumatorului este însă foarte mare: $(N \cdot (2 \cdot 2^{2 \cdot mM}) + 1)$, soluţia nefiind aplicabilă în practică.

Testarea sumatoarelor ISBCLA şi SRCLA se face în mod similar, obţinându-se aceleaşi *pattern*-uri de test.

În Tabel 3. 16 se dă numărul testelor necesare unei testări complete în raport cu defectele de blocare singulară pentru diferite configuraţii şi tipuri de sumatoare (obţinute prin această metodă)

BUPT

	Dimens.grup		Număr de teste		
Lățime (n)	m	M	CSkA	RCLA, BCLA	SBCLA, ISBCLA, SRCLA
16	2	2	65	65	289
		4	65	65	289
	4	2	129	129	321
32	2	2	129	129	1089
		4	129	129	1089
		8	129	129	1089
	4	2	257	257	1153
		4	257	257	1153
	8	2	2049	2049	5121
64	2	2	257	257	4225
		4	257	257	4225
		8	257	257	4225
		16	257	257	4225
	4	2	513	513	4352
		4	513	513	4352
		8	513	513	4352
	8	2	4097	4097	18 433
		4	4097	4097	18 433
	16	2	$2^{21}+1=2\ 097\ 153$	$2^{21}+1=2\ 097\ 153$	1 310 721

Tabel 3. 16

3.2.4.3 Segmentarea fizică la nivelul structurilor de însumare

De multe ori, în cazul circuitelor mari, tehnicile anterior descrise pentru testarea pseudoexhaustivă duc la obținerea unor teste lungi. Pentru a aplica eficient testarea pseudoaexhaustivă pentru acest tip de circuite, este necesar să se apeleze la conceptul segmentării fizice. În acest sens, circuitul este împărțit în subcircuite prin utilizarea tehnicilor de segmentare hardware [FUJI90].
Acest aspect a fost considerat de Bozorgui-Nesbat & McCluskey (1980), care au prezentat o metodă generalizată pentru realizarea unui număr arbitrar de partiționări pentru un circuit dat [MCCL81]; în Fig.3. 38 este prezentată metoda de partiționare a unui circuit în 3 subcircuite.

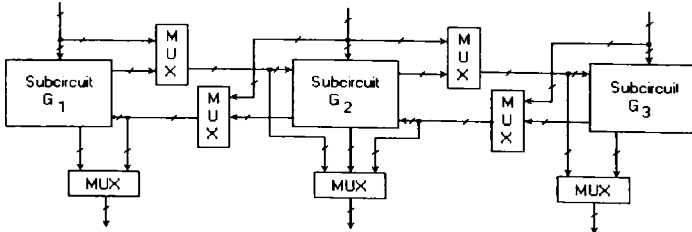


Fig.3. 38

Segmentarea fizică poate fi realizată și prin adăugarea celulelor *bypass* de memorare (*bypass storage cell*) în diferite puncte ale liniilor de semnal. O celulă *bypass* de memorare este o celulă de memorie care în funcționare normală se comportă ca și simplă legătură, dar în mod test poate fi parte a unui circuit LFSR. Este similară celulelor utilizate în *boundary-scan design* (Fig.3. 39) [GLOS89].

BUPT

Dacă o asemenea celulă este inserată la nivelul liniei x, atunci LFSR-ul asociat poate fi folosit ca şi MISR şi astfel să detecteze erorile ce apar pe linia x, sau poate fi folosit ca şi PRPG ce să genereze *pattern*-uri pe linia x [POPE98F].

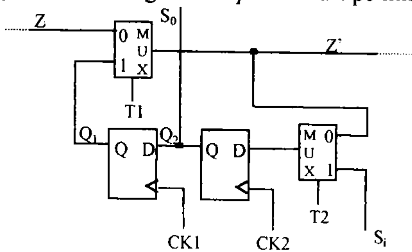


Fig.3. 39

Pentru creşterea testabilităţii circuitelor, Oklobodzija şi Ercegovac [OKLO82] au propus o metodă de intervenţie în circuit la nivelul structurii porţilor acestuia, realizând astfel o îmbunătăţire a controlabilităţii şi observabilităţii circuitului în ansamblu.

Metoda constă în schimbarea anumitor porţi dintr-un circuit logic dat, în porţi controlabile, ce să realizeze partiţionarea circuitului în subcircuite realtiv mici.

O astfel de poartă controlabilă este definită ca fiind o poartă cu o intrare de control c, a cărei ieşire g ia valoarea $f(x_1, x_2, x_3, \dots x_n)$, dacă $c = k$ şi ia valoarea $g(x_p)$, dacă $c = \bar{k}$; unde f este funcţia porţii în mod de operare normal, şi g este o funcţie de intrarea de prioritate x_p, $(1 \le p \le n)$ şi k este o constantă 0, sau 1.

Intrucât $g(x_p)$ este fie x_p, fie $\overline{x_p}$, valoarea intrării de prioritate va fi transmisă direct, sau în formă negată la ieşirea porţii controlabile. Aceasta face posibilă blocarea tuturor căilor spre intrarea porţii, cu excepţia lui x_p, creându-se o cale prin x_p.

Deci, aceste porţi controlabile pot fi folosite pentru crearea căilor spre, sau dinspre un subcircuit arbitrar, pentru a controla intrările şi a observa ieşirile. Poarta controlabilă poate funcţiona ca şi un multiplexor *low-cost*.

De notat că dezavantajul numărului mare de pini de intrare/ieşire implicaţi de metodă, poate fi redus prin utilizarea unui registru de deplasare prin care să se controleze liniile de control ale circuitului. Conţinutul acestui registru de deplasare se poate încărca serial din exterior, printr-un pin adiţional; fiecare celulă a registrului de deplasare având control asupra a unei, sau a mai multor porţi.

Astfel, conform celor anterior prezentate, pentru reducerea dimensiunilor conurilor asociate ieşirilor sumatoarelor, este necesară realizarea unei segmentări fizice la nivelul acestora. Segmentarea fizică se poate realiza în blocuri de dimensiune $m = 2, 4, 8, 16, 32$. Împărţirea fizică la nivelul sumatoarelor RCA, respectiv CCA se poate realiza în blocuri de dimensiune arbitară, în schimb în cazul sumatoarelor CSkA, respectiv a celor realizate prin aplicarea tehnicilor de *lookahead* segmentarea fizică va urmări segmentarea structurală a acestora.

3.2.4.4 Testarea sumatoarelor cu ajutorul pattern-urilor de pondere constantă

Considerăm circuitele sumatoarelor ca fiind partiţionate fizic cu ajutorul unor porţi comandate, astfel ca în regim de test să avem pentru fiecare bloc $CY-in = CY-out$, ambele egale cu C_{-1}. Ne propunem să determinăm numărul testelor generate prin *pattern*-uri de pondere constantă ce sunt necesare pentru o testare completă, prin tehnica de *verification testing* anterior prezentată, a acestor sumatoare în raport cu defectele singulare de blocare la 0 sau la 1.

BUPT

În vederea ajungerii la rezultatele dorite, vom presupune că lungimea operanzilor este n, iar dimansiunea blocurilor ce se constituie ca şi segmente fizice obţinute prin partiţionare este m. Pentru fiecare ieşire S_i, a sumatoarelor, se asociază conul format din logica asociată acesteia. Conurile de dimensiune maximă vor fi cele asociate celui mai semnificativ bit sumă la nivelul fiecărui segment fizic obţinut; întrucât semnalul CY-in al fiecărui bloc (ce a fost asignat ca fiind C_{-1}), este tratat îm mod distinct, dimensiunea lor de intrare este 2^{2m}.

O *verification testing* produce toate cele 2^m *pattern*-uri de intrare pe fiecare din cele C_n^m subseturi de n intrări ale circuitului C. Sumatorul (indiferent de tip) ale cărei ieşiri sunt testate concurent pe baza acestei tehnici, se constituie ca şi un circuit de tipul $(2n,2m)$-CUT. unde $2m < 2n$.

De notat că în situaţia testării sumatoarelor de tipul CSkA cu blocuri de dimensiune variabilă, m va reprezenta dimensiunea celui mai mare bloc al sumatorului. Întrucât în toate cazurile de analiză pe care le-am considerat la nivelul sumatoarelor, dimensiunea blocurilor sale este mai mică decât 1/2 din dimensiunea sumatorului, pentru calculul ponderilor constante ale *pattern*-urilor ce se generează se va ţine cont de observaţia că dimensiunea setului minim căutat apare când $w = \lfloor 2m/2 \rfloor = m$ şi $w = \lfloor 2m/2 \rfloor + (2n - 2m + 1)$.
Rezultatele analizei sunt cuprinse sintetic în tabelul de mai jos, unde $2m$ corespunde lui k utilizat în secţiunea 3.2.4.1:

n	m=$\lfloor k/2 \rfloor$	2n-2m+1	$C_{2n}^{\lfloor k/2 \rfloor}$	$C_{2n}^{k-1-\lfloor k/2 \rfloor}$	Număr total teste la testarea exhaustivă a ambilor operanzi	Număr total teste la testarea exhaustivă a unui singur operand
16	2	29	$C_{32}^{2}=496$	$C_{32}^{1}=32$	C_{33}^{2}	$C_{17}^{2}=136$
	4	25	$C_{32}^{4}=35960$	$C_{32}^{3}=4960$	C_{33}^{4}	$C_{17}^{4}=9520$
	8	17	$C_{32}^{8}=10518300$	$C_{32}^{7}=23560992$	C_{33}^{8}	$C_{17}^{8}=680680$
32	2	61	C_{64}^{2}	$C_{64}^{1}=64$	C_{65}^{2}	$C_{33}^{2}=528$
	4	57	C_{64}^{4}	C_{64}^{3}	C_{65}^{4}	$C_{33}^{4}=153120$
	8	49	C_{64}^{8}	C_{64}^{7}	C_{65}^{8}	$C_{33}^{8}=10518300$
	16	33	C_{64}^{16}	C_{64}^{15}	C_{65}^{16}	C_{33}^{16}
64	2	125	C_{128}^{2}	$C_{128}^{1}=128$	C_{129}^{2}	$C_{65}^{2}=2080$
	4	121	C_{128}^{4}	C_{128}^{3}	C_{129}^{4}	$C_{65}^{4}=677040$
	8	113	C_{128}^{8}	C_{128}^{7}	C_{129}^{8}	C_{65}^{8}
	16	97	C_{128}^{16}	C_{128}^{15}	C_{129}^{16}	C_{65}^{16}
	32	65	C_{128}^{32}	C_{128}^{31}	C_{129}^{32}	C_{65}^{32}

Tabel 3. 17

Numărul testelor obţinut pe această cale este inacceptabil, cu atât mai mult cu cât ele trebuiesc aplicate odată pentru $C_{-1}=0$, şi încă o dată pentru $C_{-1}=1$.

S-ar putea obţine o reducere a acestui număr în ideea testării exhaustive a unuia dintre operanzi, şi pentru fiecare testare exhaustivă a acestuia, celălalt operand ia pe rând valorile:

BUPT

0000...0000
1111...1111
0101...0101
1010...1010

Pattern-urile de mai sus sunt aplicate atât pentru $C_{-1}=0$ cât şi pentru $C_{-1}=1$.
Aplicând tehnicii descrisă mai sus această simplificare, se obţin rezultatele din ultima coloană a tabelului de mai sus, cu observaţia că numărul total de teste se obţine prin înmulţirea cu 8 a datelor din această coloană.
Întrucât şi pentru aceasă situaţie numărul testelor obţinute este mult prea mare, se procedează la identificarea liniilor de intrare ce pot fi stimulate cu aceleaşi semnale de test.

3.2.4.5 Identificarea intrărilor semnal de test

Se consideră un circuit cu n intrări. Pe parcursul testării există situaţii când prin aplicarea valorilor de intrare pe p linii semnal de test $(p < n)$ să se realizeze comanda prin aceste p linii a tuturor celor n intrări ale circuitului. Pentru aceste situaţii interesează identificarea celor p linii semnal de test precum şi a testelor asociate acestor linii.

Un circuit se spune că este *maximal-test-concurrency circuit* (MTC) dacă numărul minim de semnale de test necesare pentru testarea acestui circuit este egal cu numărul maxim de intrări de care depinde fiecare ieşire a circuitului.

În continuare se prezintă o procedură de partiţionare a intrărilor unui circuit pentru determinarea:

- numărului minim de semnale de test necesare pentru testarea circuitului
- identificarea intrărilor circuitului testat ce pot avea aceleaşi semnale de test

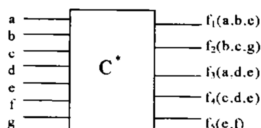


Fig.3. 40

Astfel de circuite pot fi testate cu *pattern*-uri cu pondere constantă.
Paşii procedurii vor fi ilustraţi pentru circuitul C* din Fig.3. 40.
Pe baza celor prezentate se va putea face identificarea circuitelor MTC şi construirea testelor atât pentru circuitele MTC cât şi pentru cele care nu sunt MTC.

<u>Procedură de identificare a setului minimal de semnale de test</u>

<u>Pasul 1:</u> Partiţionarea circuitului în subcircuite disjuncte
Circuitul C^* constă dintr-o singură partiţie.

<u>Pasul 2:</u> Pentru fiecare subcircuit disjunt se realizează următorii paşi:

a) Se generează o matrice de dependenţă.

Pentru un circuit cu m ieşiri şi n intrări, matricea de dependenţă $D = |d_{ij}|$ este o matrice cu m linii şi n coloane, în care $d_{ij} = 1$ dacă ieşirea i este dependentă de intrarea j, în caz contrar $d_{ij} = 0$. Pentru circuitul C^* avem:

$$D = \begin{array}{|ccccccc|c} a & b & c & d & e & f & g & \\ \hline 1 & 1 & 0 & 0 & 1 & 0 & 0 & f_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & f_2 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & f_3 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & f_4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & f_5 \end{array}$$

BUPT

b) Partiționarea matricii în grupe de ieşiri astfel încât două sau mai multe intrări dintr-un grup să nu afecteze aceleaşi ieşiri. Pentru aceasta fiecare rând al unui grup trebuie să aibă mai puțin de două unități, numărul grupurilor trebuind să fie minimal. Din păcate determinarea unor astfel de partiții este o problemă *NP-complete*.

Prin reordonare şi regrupare se obține matricea D_g.

Grupări

	I		II		III	IV		
	a	c	b	d	e	f	g	
$D_g =$	1	0	1	0	1	0	0	f_1
	0	1	1	0	0	0	1	f_2
	1	0	0	1	1	0	0	f_3
	0	1	0	1	1	0	0	f_4
	0	0	0	0	1	1	0	f_5

c) Se comprimă fiecare grup pentru a se forma o intrare echivalentă, numită *intrare semnal de test*.

Această matrice comprimată (*collapsed*) echivalentă, D_c, se obține aplicând operația OR între toate coloanele fiecărui grup. Pentru circuitul C^* avem:

	I	II	III	IV	
$D_c =$	1	1	1	0	f_1
	1	1	0	1	f_2
	1	1	1	0	f_3
	1	1	1	0	f_4
	0	0	1	1	f_5

Pasul 3: Se caracterizează matricea D_c cu ajutorul a doi parametrii: p şi w.

- p reprezintă numărul partițiilor din D_c, fiind numit lățimea lui D_c, şi exprimă numărul maxim al semnalelor de intrare ce sunt necesare pentru testarea subcircuitelor disjuncte.
- w este dat de numărul maxim al unităților dintr-un rând, fiind numit ponderea lui D_c şi exprimă numărul maxim al semnalelor de care depinde fiecare ieşire.

Prin urmare, lungimea unui test pseudoexhaustiv va fi $\geq 2^w$ şi $\leq 2^p$.

Pentru matricea D parametrii determinați sunt: $p = 4$ şi $w = 3$.

Un set de test minimal universal pseudoexhaustiv ce are parametrii (p, w) este un set minimal de *pattern*-uri de test ce conține pentru toate cele C_n^w subseturi ce constau din w din cele p linii de semnal, toate cele 2^w *pattern*-uri de test. Proprietățile acestor seturi de test sunt determinate de valorile relative ale lui w şi p, unde prin definiție $p > w$. Pentru un anumit circuit, dacă toate ieşirile sunt funcții de w intrări, atunci acest set este minimal în lungime. Pentru celelalte situații s-ar putea să nu fie.

Pasul 4: Se construiesc *pattern*-urile de test pentru circuit pe baza următoarelor cazuri.

Cazul 1: $p = w$

Cazul 2: $p = w + 1$

cazul 3: $p > w + 1$

Cazul 1: $p = w$

Cazul corespunde circuitelor MTC şi testul constă din toate cele 2^p *pattern*-uri de test de p biți. Testele pot fi generate cu ajutorul unui numărător sau a unui

BUPT

LFSR. În mod evident acesta este un set de test pseudoexhaustiv minimal universal. Acest caz corespunde generării *pattern*-urilor de pondere constantă cu $k = n$, setul de test rezultat fiind T_0.

Cazul 2: $p = w + 1$

Cazul corespunde situației de *pattern*-uri cu pondere constantă având $n = k + 1$. Setul de test minimal constă din toate *pattern*-urile posibile de p biți ce au fie paritate pară, fie paritate impară. Există $2^{n-1} = 2^w$ asemenea *pattern*-uri. Mai jos sunt listate testele circuitului C^* pentru paritate impară.

A	B	C	D	
0	0	0	1	
0	0	1	0	paritate=1
0	1	0	0	
1	0	0	0	
0	1	1	1	
1	0	1	1	paritate=3
1	1	0	1	
1	1	1	0	

De observat că pentru oricare subset de 3 coloane apar toți cei 2^3 tripleți binari.

Acest set de test pseudoexhaustiv este format de 8 *pattern*-uri de test, în timp ce o testare exhaustivă a circuitului presupune utilizarea a 128 *pattern*-uri de test. Fiecare coloană reprezintă o intrare pentru fiecare linie a unui grup (de exemplu coloana A poate fi intrare pentru liniile a și c ale circuitului original).

p	w	T	Ponderi constante
p>3	2	p+1	(0,p-1) sau (1,p)
p>4	3	2p	(1,p-1)
p>5	4	$\frac{1}{2}p\cdot(p+1)$	(1, p-2) sau (2,p-1)
p>6	5	$p\cdot(p-1)$	(2,p-2)
p=8	6	$\frac{1}{2}p\cdot(p+1)$	(1,4,7)
p>8	6	C^4_{p+1}	(2,p-3) sau (3, p-2)
p=9	7	170	(0,3,6,9)
p>9	7	$2\cdot C^3_p$	(3,p-3)
p=10	8	341	(0,3,6,9) sau (1,4,7,10)
p=11	8	496	(0,4,8) sau (3,7,9)
p>11	8	C^4_{p+1}	(3,p-4) sau (4,p-3)
p=11	9	682	(1,4,7,10)
p=12	9	992	(0,4,8,12)
p>12	9	$2C^4_p$	(4,p-4)
p=12	10	1365	(1,4,7,10) sau (2,5,8,11)
p=13	10	2016	(0,4,8,12) sau (1,5,9,13)
p=14	10	3004	(0,5,10) sau (4,9,14)
p>14	10	C^5_{p+1}	(4,p-5) sau (5, p-4)

Tabel 3. 18

Cazul 3: $p > w + 1$

În acest caz setul de test constă din *2* sau mai multe subseturi de *pattern*-uri, fiecare din ele conținând toate *pattern*-urile posibile de p biți ce au o pondere constantă specifică.

Numărul total al *pattern*-urilor de test T este o funcție de p și w. În Tabel 3. 18 sunt date valorile ponderilor constante și T pentru diferite valori ale lui p și w. De notat că $(a_1, a_2, \ldots)$ reprezintă un vector de ponderi constante, unde a_i este ponderea constantă pentru un subset de pattern-uri.

Din păcate nu există ponderi constante pentru toate perechile p și w. În aceste situații se poate crește w astfel încât să se realizeze un test pseudoexhaustiv cu pondere constantă, dar care nu va fi minimal ca lungime.

BUPT

Din păcate, pentru $p > w+1$, nu totdeauna este simplu de construit un circuit ce să genereze teste pseudoexhaustive, iar pe de altă parte consumul de suprafaţă de integrare implicat de asemenea circuite este câteodată destul de ridicat.
În continuare vor fi prezentate pe scurt câteva tehnici de proiectare a circuitelor generatoare de teste pseudoexhaustive. Multe dintre acestea nu generează totdeauna teste minimale, dar ele duc la o proiectare hardware eficientă. Întrucât cele mai multe TPG-uri utilizează anumite forme de LFSR-uri, şi deoarece în unele cazuri sunt necesare mai multe secvenţe de test, pentru iniţializarea stării LFSR-ului, este necesară în unele situaţii mai mult de o valoare de pornire (*seed*).

3.2.4.6 Determinarea setului complet de teste pentru testarea pseudoexhaustivă a sumatoarelor

Se consideră sumatoarele partiţionate fizic, dimensiunea blocurilor fiind *m*. În cazul sumatoarelor CSkA cu dimensiune variabilă a blocurilor, *m* reprezintă dimensiunea celui mai mare bloc al sumatorului.
Pentru identificarea liniilor semnal de test ale acestora se consideră un sumator pe 8 biţi partiţionat în blocuri de 2 biţi. Rezultatele analizei următoare pot fi generalizate sumatoarelor de orice dimensiune partiţionate fizic în blocuri ce au şi ele de orice dimensiune.
Se începe cu construirea matricei de dependenţă a sumatorului propus pentru analiză.

D=

A_7	B_7	A_6	B_6	A_5	B_5	A_4	B_4	A_3	B_3	A_2	B_2	A_1	B_1	A_0	B_0	C_{-1}	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	S_0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	S_1
0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	S_2
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	S_3
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	S_4
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	S_5
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	S_6
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	S_7

Pe baza matricei de dependenţă se construielte matricea D_g:

D_g=

A_6	A_4	A_2	A_0	B_6	B_4	B_2	B_0	A_7	A_5	A_3	A_1	B_7	B_5	B_3	B_1	C_{-1}	
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	S_0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	S_1
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1	S_2
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1	S_3
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	S_4
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	S_5
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	S_6
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	S_7

Se obţine matricea D_c:

D_c=

I	II	III	IV	V	
1	1	1	1	1	S_0
1	1	1	1	1	S_1
1	1	1	1	1	S_2
1	1	1	1	1	S_3
1	1	1	1	1	S_4
1	1	1	1	1	S_5
1	1	1	1	1	S_6
1	1	1	1	1	S_7

BUPT

Se caracterizează matricea D_c cu ajutorul a doi parametrii: p şi w, a căror valoare este: $w = p = 2^2 + 1 = m^2 + 1$, unde:

- p numărul maxim al semnalelor de intrare ce sunt necesare pentru testarea subcircuitelor disjuncte.
- w exprimă numărul maxim al semnalelor de care depinde fiecare ieşire.

Cazul corespunde circuitelor MTC şi testul constă din toate cele $2^w = 2^{2m+1}$ *pattern*-uri de test de p biţi. Testele pot fi generate cu ajutorul unui numărător sau a unui LFSR. În mod evident acesta este un set de test pseudoexhaustiv minimal universal. Acest caz corespunde generării *pattern*-urilor de pondere constantă cu $k = n$, setul de test rezultat fiind T_0.

Valorile celor doi parametrii determinaţi au aceleaşi expresii indiferent de dimensiunea blocurilor, de dimensiunea sumatorului şi de structura lui.

În concluzie, prin aplicarea metodei de segmentare fizică (ce urmăreşte partiţionarea structurală în blocuri a sumatorului testat), în conjuncţie cu tehnica *verification testing* şi aplicând teste doar liniilor ce au fost identificate ca fiind de test, se obţin, relativ la numărul total de teste necesare unei testări complete pentru detecţia defectelor de blocare singulară la 0 sau 1, rezultatele din Tabel 3. 19.

Dimensiunea segmentului fizic (m)	$p = w$	Număr total teste necesar testării complete a sumatorului indiferent de dimensiunea lui
2	5	$2^p = 2^5 = 32$
4	9	$2^p = 2^9 = 512$
8	17	$2^p = 2^{17} = 131072$
16	33	$2^p = 2^{33}$
32	65	$2^p = 2^{65}$

Tabel 3. 19

Valorile obţinute pentru $m = 16$ şi $m = 32$ sunt mult prea mari pentru a fi acceptabile. Prin urmare, metoda propusă este aplicabilă pentru sumatoare la care dimensiunea segmentelor fizice este maxim 8.

3.3 Concluzii

Capitolul 3 abordează problematica specifică autocontrolului bazat pe hardware. În prima parte a acestui capitol se tratează problematica autocontrolului *on-line*, precum şi cea a proiectării circuitelor de verificare cu proprietăţi speciale cum ar fi testabilitate, autotestabilitate şi altele, precum şi a circuitelor cu autoverificare totală, restrângându-se astfel domeniul de investigaţii. Pentru autocontrolul structurilor de însumare se dau soluţii de reconfigurare a acestora, ce să permită autotestarea lor (reconfigurări prin dublarea semnalelor CY, precum şi reconfigurări ce forţează dependenţa funcţiilor de însumare de valorile semnalelor CY generate) prin controlul parităţii, a sumelor de control, respectiv prin utilizarea codurilor combinate.

În a doua parte a acestui capitol este abordată problematica autocontrolului *off-line*, insistându-se asupra identificării numărului de pattern-uri necesare aplicării acestor tehnici la nivelul structurilor de însumare. Concluziile analizelor efectuate sunt următoarele:

- Testarea exhaustivă a sumatoarelor nu este eficientă pentru dimensiuni de însumare mai mari decât 8 ranguri

BUPT

- Testarea pseudoexhaustivă bazată pe partiționare logică prin activarea căilor este eficientă doar pentru dimensiuni mici ale blocurilor logice (implicit ale blocurilor de însumare) și ea nu depinde de dimensiunea superblocurilor, întrucât nu este eficientă segmentarea logică la nivelul acestora.
- Testarea pseudoexhaustivă cu ajutorul pattern-urilor cu pondere constantă este aplicabilă în conjuncție cu partiționarea structurilor prin segmentare fizică și cu identificarea semnalelor comune de test, dovedindu-se eficientă doar pentru dimensiuni ale blocurilor fizice mai mici decât 8.
- Testarea pseudoexhaustivă prin generare aleatoare a testelor este ineficientă datorită numărului mare de teste implicat în testarea structurilor de însumare. Numărul acestor teste este funcție de probabilitatea detectării defectului cel mai dificil detectabil la nivelul structurilor de însumare (este vorba de defectele ce afectează lanțurile semnalelor CY) și de calitatea ce se dorește pentru procesul testării. În secțiunea 3.2.3.3 este prezentat în pseudocod un algoritm ce permite calculul măsurilor de testabilitate a circuitelor logice, inclusiv cel al probabilității de detecție a defectelor la nivelul acestora.

BUPT

4. Evaluarea rezultatului testării utilizând mediul LDL

Validarea experimentală a rezultatelor obţinute în cadrul tezei, a fost făcută prin simulare pe calculator, urmărindu-se următoarele aspecte:

- Validarea numărului minim de teste pentru fiecare structură de însumare analizată
- Validarea acoperirii complete, prin algoritmii optimali elaboraţi, a defectelor de blocare singulară la 1 sau 0 la nivelul structurilor de însumare analizate

Pentru realizarea simulării, a fost dezvoltat, ca şi instrument cu puternice facilităţi în această direcţie, mediul integrat LDL (*Logic Description Language* – LDL) (descris în Anexa B). Modulele sale program au fost implementate în limbaj Visual C++, favorabil realizării unei interfaţări plăcute şi consistente cu utilzatorul.

Mediul LDL permite descrierea prin limbaj LDL a schemei logice a circuitului analizat din punct de vedere al testării, pe baza căreia se obţine, prin compilare, modelul intern al circuitului sub forma unui cod tabelar. Specificaţii legate de descrierea modelării interne a circuitelor se găsesc în [ABRA96].
Odată circuitul modelat prin intermediul modulului editor de texte şi a modulului compilator, el poate fi supus în cadrul mediului LDL următoarelor operaţii şi evaluări:

- Simulare logică – modulul SLC (simulare logică prin compilare).
  Operaţia de simulare logică a circuitelor este realizată imediat după faza lor de proiectare, fiind necesară pentru validarea lipsei de erori de concepţie la nivelul proiectelor logice.
- Obţinere vectori stimuli de test prin metoda algebrică de elaborare a ecuaţiilor Poage – modulul GVST-Poage (generare vectori stimuli prin metoda Poage) [VLĂD89B]
- Simulare a defectelor – modulul SPD (simulare paralelă a defectelor)
  Modulul SPD realizează o simulare paralelă a defectelor bazată pe procesul de injecţie a defectelor pe liniile circuitului analizat. Pentru injecţie se face uz la nivelul tuturor liniilor (s) de circuit de cele două măşti ce li se asociază: mask(s), şi fvalue(s) (secţiunea 1.5.3.2.2). Permite evaluarea acoperirii defectelor
- Minimizare a seturilor de vectori stimuli prin metoda determinării acoperirii minime a vectorilor stimuli de test – modul MVST-McCluskey [VLĂD89B]
  Modulul realizează minimizarea setului de vectori stimuli de test, înscrişi într-un fişier ASCII, pentru o schemă logică al cărei cod este dat într-un fişier specificat, rezultatul minimizării este înscris într-un fişier de ieşire, şi se afişează numărul testelor iniţiale, precum şi numărul minimizat al acestora, obţinut în urma rulării.
- Evaluare a măsurilor de testabilitate pe baza algoritmului prezentat în secţiunea 3.2.1.3 – modulul H-Assign
  Calculul măsurilor de testabilitate realizat de modulul H-Assign este important pentru estimarea lungimii *pattern*-urilor de test necesare testării pseudoaleatoare a circuitului ce corespunde schemei logice pentru care se face rularea.

În vederea implementarea mediului LDL a fost necesară îmbinarea metodelor hardware cu cele software; astfel, aplicaţia conţine un mediu de editare al limbajului LDL, cu facilităţile unui editor de texte, generator de cod (care simplifică adăugarea de noi module în sursa programului), un compilator pentru translatarea programului într-o sursă internă, un analizor de tabele de adevăr pentru a verifica semnalele de intrare,

BUPT

intermediare şi cele de ieşire în cazul în care circuitul funcţionează fără defecte şi în cazul în care se injectează defecte de tip blocare la 1 sau 0 a liniilor de semnal, un simulator logic pentru analiza funcţionării circuitului în timp şi un altul pentru simularea defectelor.

Este important de notat aspectul referitor la faptul că transpunerea schemelor logice în limbaj oferă, în funcţie de nivelul de abstractizare al descrierii, diferite perspective asupra circuitelor.
Detalii suplimentare legate de acest program se găsesc în Anexa B.

Prin urmare, validarea numărului minimal de teste identificate în cadrul capitolului 2 la nivelul fiecărei structuri, a fost făcută rulând modulul GVST-Poage a mediului integrat LDL, pentru structurile de însumare analizate.
Pe de altă parte, validarea acoperirii complete a defectelor de blocare singulară la 1 sau 0 la nivelul structurilor de însumare, pentru testele obţinute pe baza algoritmilor originali (prezentaţi în Anexa A) de generare automată a vectorilor stimuli de test, a fost realizată cu ajutorul simulatorului de defecte - modulul SPD a mediului integrat LDL.

În felul acesta s-a realizat validarea completă a contribuţiilor originale în domeniul generării automate a seturilor minimale de teste necesare detecţiei SSF la nivelul structurilor de însumare.

4.1 Evaluări ale sumatoarelor cu propagarea serială a transportului

Evaluarea sumatorului RCA
Testele necesare unei testări complete a sumatorului RCA în raport cu defectele de blocare singulară la 1 sau 0 specificate în [FRIE73][TUNG87] şi [CHIH94] sunt date în Tabel 4. 1.
A fost necesară includerea acestor teste (în elaborarea cărora nu au fost aduse nici un fel de contribuţii) în cadrul acestui capitol, întrucât ele sunt necesare pentru testarea altor structuri de însumare.

	C_{-1}	A_0B_0	A_1B_1	$A_2\ B_2$	•••	$A_i\ B_i$	$A_{i+1}\ B_{i+1}$...	$S_0\ S_1\ S_2\ S_3$ •••	C_n
T_1	0	0 0	0 0	0 0	•••	0 0	0 0 •••	0 0 0 0 •••	0
T_2	0	0 1	0 1	0 1	•••	0 1	0 1•••	1 1 1 1 •••	0
T_3	0	1 0	1 0	1 0	•••	1 0	1 0 •••	1 1 1 1 •••	0
T_4	1	0 1	0 1	0 1	•••	0 1	0 1•••	0 0 0 0 •••	1
T_5	1	1 0	1 0	1 0	•••	1 0	1 0•••	0 0 0 0 •••	1
T_6	1	1 1	1 1	1 1	•••	1 1	1 1•••	1 1 1 1 •••	1
T_7	0	1 1	0 0	1 1	•••	0 0 sau 1 1	1 1••• sau 0 0 •••	0 1 0 1 •••	0 sau 1
T_8	1	0 0	1 1	0 0	•••	1 1 sau 0 0	0 0 ••• sau 1 1•••	1 0 1 0 •••	1 sau 0

Tabel 4. 1

Fiecare linie a Tabel 4. 1 prezintă pattern-urile de intrare ce corespund unui test. Ultimele două coloane ale tabelului cuprind valorile de ieşire ce corespund testelor aplicate la intrările circuitului.
Timpul de operare pentru sumatorul RCA este $(2 \cdot n - 1)\tau$, deci performabilitatea sa este $[8 \cdot (2 \cdot n - 1)]^{-1}$.

BUPT

Evaluarea sumatorului CCA

Testele necesare unei testări complete a sumatorului CCA în raport cu defectele de blocare singulară la 1 sau 0 sunt date în Tabel 4. 2:

	$C_{-1}^0\ C_{-1}^1$	A_0B_0	A_1B_1	$A_2\ B_2$	•••	$A_i\ B_i$	$A_{i+1}\ B_{i+1}$...	$S_0\ S_1\ S_2\ S_3$ •••
T_1	0 1	0 1	0 1	0 1	•••	0 1	0 1•••	0 0 0 0•••
T_2	0 1	1 1	1 1	1 1	•••	1 1	1 1•••	1 1 1 1•••
T_3	0 1	1 0	1 0	1 0	•••	1 0	1 0•••	0 0 0 0•••
T_4	1 0	0 0	0 0	0 0	•••	0 0	0 0•••	0 0 0 0•••
T_5	1 0	0 1	0 1	0 1	•••	0 1	0 1••	1 1 1 1•••
T_6	1 0	1 0	1 0	1 0	•••	1 0	1 0•••	1 1 1 1•••
T_7	0 1	0 0	1 1	0 0	•••	1 1 sau 0 0	1 0 ••• sau 1 1•••	1 0 1 0•••
T_8	1 0	1 1	0 0	1 1	•••	0 0 sau 1 1	1 1••• sau 0 0 •••	0 1 0 1•••

Tabel 4. 2

Timpul mediu de operare pentru sumatorul RCA este $(2\cdot\log_2 n+4)\tau$, deci performabilitatea sa este $[8\cdot(2\cdot\log_2 n+4)]^{1}$.

4.2 Evaluări ale sumatoarelor cu propagarea anticipată a transportului

4.2.1 Evaluarea sumatorului CLA

Timpul de operare al structurilor de însumare de tip CLA este constant, fiind egal cu 4τ.

Din punctul de vedere al testării este de notat faptul că celulele sumatoare de tip CLA în formă pură stau la baza formării celulelor repetabile ale structurilor hibride de însumare ce se constituie ca şi structuri ILA. Ca urmare a acestui fapt, este importantă identificarea setului minimal de teste necesare testării lor complete în raport cu defectele de blocare singulară la 1 sau 0. Acest set minimal a fost determinat cu ajutorul algoritmului original dat în Fig.A.1 a Anexei A; testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL

A fost demonstrat, prin teorema 2.3, că numărul acestor teste este $\left(\frac{(n+1)\cdot(n+2)}{2}+1\right)$, prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este: $\left[4\cdot\left(\frac{(n+1)\cdot(n+2)}{2}+1\right)\right]^{-1}$

CLA în formă pură pe 2 biţi

Setul complet şi minimal de teste pentru detecţia defectelor de blocare singulară la 0 sau 1 a sumatoarelor CLA pe 2 biţi este dat în Tabel 4. 3:

BUPT

	A_{2k+1}	B_{2k+1}	A_{2k}	B_{2k}	C_{2k-1}	C_{2k+1}
T1	0	1	0	1	0	0
T2	1	0	1	0	1	1
T3	0	1	0	0	1	0
T4	0	0	0	1	1	0
T5	0	1	1	1	0	1
T6	0	0	1	1	0	0
T7	1	1	0	1	0	1

Tabel 4. 3

CLA în formă pură pe 4 biţi

Setul complet şi minimal de teste pentru detecţia defectelor de blocare singulară la 0 sau 1 a sumatoarelor CLA pe 4 biţi este dat în Tabel 4. 4:

	A_{4k+3}	B_{4k+3}	A_{4k+2}	B_{42k+2}	A_{4k+1}	B_{4k+1}	A_{4k}	B_{4k}	C_{4k-1}	C_{4k+3}
T1	0	1	1	0	0	1	0	1	0	0
T2	1	0	0	1	1	0	1	0	1	1
T3	0	1	0	1	0	1	0	0	1	0
T4	0	1	0	1	0	0	0	1	1	0
T5	0	1	0	0	0	1	0	1	1	0
T6	0	0	0	1	0	1	0	1	1	0
T7	0	1	0	1	0	1	1	1	0	1
T8	0	1	0	1	0	0	1	1	0	0
T9	0	1	0	0	0	1	1	1	0	0
T10	0	0	0	1	0	1	1	1	0	0
T11	0	1	0	1	1	1	0	1	0	1
T12	0	1	0	0	1	1	0	1	0	0
T13	0	0	0	1	1	1	0	1	0	0
T14	0	1	1	1	0	1	0	1	0	1
T15	0	0	1	1	0	1	0	1	0	0
T16	1	1	0	1	0	1	0	1	0	1

Tabel 4. 4

CLA în formă pură pe 8 biţi

Setul complet şi minimal de teste pentru detecţia defectelor de blocare singulară la 0 sau 1 a sumatoarelor CLA pe 16 biţi este dat în Tabel 4. 5:

	A_{8k+7}	B_{8k+7}	...	...	...	...	...	...	A_{4k}	B_{4k}	C_{4k-1}	C_{4k+3}
T1	0	1	0 1	0 1	0 1	0 1	0 1	0 1	0	1	0	0
T2	1	0	1 0	1 0	1 0	1 0	1 0	1 0	1	0	1	1
T3	0	1	0 1	0 1	0 1	0 1	0 1	0 1	0	0	1	0
T4	0	1	0 1	0 1	0 1	0 1	0 1	0 0	0	1	1	0
T5	0	1	0 1	0 1	0 1	0 1	0 0	0 1	0	1	1	0
T6	0	1	0 1	0 1	0 1	0 0	0 1	0 1	0	1	1	0
T7	0	1	0 1	0 1	0 0	0 1	0 1	0 1	0	1	1	0
T8	0	1	0 1	0 0	0 1	0 1	0 1	0 1	0	1	1	0
T9	0	1	0 0	0 1	0 1	0 1	0 1	0 1	0	1	1	0
T10	0	0	0 1	0 1	0 1	0 1	0 1	0 1	0	1	1	0
T11	0	1	0 1	0 1	0 1	0 1	0 1	0 1	1	1	0	1
T12	0	1	0 1	0 1	0 1	0 1	0 1	0 0	1	1	0	0
T13	0	1	0 0	0 1	0 1	0 1	0 0	0 1	1	1	0	0
T14	0	1	0 1	0 1	0 1	0 0	0 1	0 1	1	1	0	0
T15	0	1	1 1	0 1	0 0	0 1	0 1	0 1	1	1	0	0

BUPT

	A_{8k+7} B_{8k+7}	...	...	...	...	...	...	A_{4k} B_{4k}	C_{4k-1}	C_{4k+3}
T16	0 1	1 1	*0 0*	0 1	0 1	0 1	0 1	*1 1*	0	0
T17	0 1	*0 0*	0 1	0 1	0 1	0 1	0 1	*1 1*	0	0
T18	*0 0*	0 1	0 1	0 1	0 1	0 1	0 1	*1 1*	0	0
T19	0 1	0 1	0 1	0 1	0 1	0 1	*1 1*	0 1	0	1
T20	0 1	0 1	0 1	0 1	0 1	*0 0*	*1 1*	0 1	0	0
T21	0 1	0 1	0 1	0 1	*0 0*	0 1	*1 1*	0 1	0	0
T22	0 1	0 1	0 1	*0 0*	0 1	0 1	*1 1*	0 1	0	0
T23	0 1	0 1	*0 0*	0 1	0 1	0 1	*1 1*	0 1	0	0
T24	0 1	*0 0*	0 1	0 1	0 1	0 1	*1 1*	0 1	0	0
T25	*0 0*	0 1	0 1	0 1	0 1	0 1	*1 1*	0 1	0	0
T26	0 1	0 1	0 1	0 1	0 1	*1 1*	0 1	0 1	0	1
T27	0 1	0 1	0 1	0 1	*0 0*	*1 1*	0 1	0 1	0	0
T28	0 1	0 1	0 1	*0 0*	0 1	*1 1*	0 1	0 1	0	0
T29	0 1	0 1	*0 0*	0 1	0 1	*1 1*	0 1	0 1	0	0
T30	0 1	*0 0*	0 1	0 1	0 1	*1 1*	0 1	0 1	0	0
T31	*0 0*	0 1	0 1	0 1	0 1	*1 1*	0 1	0 1	0	0
T32	0 1	0 1	0 1	0 1	*1 1*	0 1	0 1	0 1	0	1
T33	0 1	0 1	0 1	*0 0*	*1 1*	0 1	0 1	0 1	0	0
T34	0 1	0 1	*0 0*	0 1	*1 1*	0 1	0 1	0 1	0	0
T35	0 1	*0 0*	0 1	0 1	*1 1*	0 1	0 1	0 1	0	0
T36	*0 0*	0 1	0 1	0 1	*1 1*	0 1	0 1	0 1	0	0
T37	0 1	0 1	0 1	*1 1*	01	0 1	0 1	0 1	0	1
T38	0 1	0 1	*0 0*	*1 1*	01	0 1	0 1	0 1	0	0
T39	0 1	*0 0*	0 1	*1 1*	01	0 1	0 1	0 1	0	0
T40	*0 0*	0 1	0 1	*1 1*	01	0 1	0 1	0 1	0	0
T41	0 1	0 1	*1 1*	0 1	01	0 1	0 1	0 1	0	1
T42	0 1	*0 0*	*1 1*	0 1	01	0 1	0 1	0 1	0	0
T43	*0 0*	0 1	*1 1*	0 1	01	0 1	0 1	0 1	0	0
T44	0 1	*1 1*	0 1	0 1	0 1	0 1	0 1	0 1	0	1
T45	*0 0*	*1 1*	0 1	0 1	0 1	0 1	0 1	0 1	0	0
T46	*1 1*	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0	1

Tabel 4. 5

CLA în formă pură pe 16 biţi

Setul complet de teste pentru detecţia defectelor de blocare singulară la 0 sau 1 a sumatoarelor CLA pe 16 biţi este dat în Tabel 4. 6:

Test	Ultimul bit	...	...	...	...	...	...	...	...	...	...	...	...	...	...	Primul bit	CY-in	CY-out
T1	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	0	0
T2	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	1	1
T3	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	*00*	1	0
T4	01	01	01	01	01	01	01	01	01	01	01	01	01	01	*00*	01	1	0
T5	01	01	01	01	01	01	01	01	01	01	01	01	01	*00*	01	01	1	0
T6	01	01	01	01	01	01	01	01	01	01	01	01	*00*	01	01	01	1	0
T7	01	01	01	01	01	01	01	01	01	01	01	*00*	01	01	01	01	1	0
T8	01	01	01	01	01	01	01	01	01	01	*00*	01	01	01	01	01	1	0
T9	01	01	01	01	01	01	01	01	01	*00*	01	01	01	01	01	01	1	0
T10	01	01	01	01	01	01	01	01	*00*	01	01	01	01	01	01	01	1	0
T11	01	01	01	01	01	01	01	*00*	01	01	01	01	01	01	01	01	1	0
T12	01	01	01	01	01	01	*00*	01	01	01	01	01	01	01	01	01	1	0
T13	01	01	01	01	01	*00*	01	01	01	01	01	01	01	01	01	01	1	0

BUPT

Test	Ultimul bit	...	...	...	...	...	...	...	...	...	...	...	...	...	...	Primul bit	CY-in	CY-out
T14	01	01	01	01	*00*	01	01	01	01	01	01	01	01	01	01	01	1	0
T15	01	01	01	*00*	01	01	01	01	01	01	01	01	01	01	01	01	1	0
T16	01	01	*00*	01	01	01	01	01	01	01	01	01	01	01	01	01	1	0
T17	01	*00*	01	01	01	01	01	01	01	01	01	01	01	01	01	01	1	0
T18	*00*	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	1	0
T19	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	*11*	1	0
T20	01	01	01	01	01	01	01	01	01	01	01	01	01	01	*00*	*11*	0	0
T21	01	01	01	01	01	01	01	01	01	01	01	01	01	*00*	01	*11*	0	0
T22	01	01	01	01	01	01	01	01	01	01	01	01	*00*	01	01	*11*	0	0
T23	01	01	01	01	01	01	01	01	01	01	01	*00*	01	01	01	*11*	0	0
T24	01	01	01	01	01	01	01	01	01	01	*00*	01	01	01	01	*11*	0	0
T25	01	01	01	01	01	01	01	01	01	*00*	01	01	01	01	01	*11*	0	0
T26	01	01	01	01	01	01	01	01	*00*	01	01	01	01	01	01	*11*	0	0
T27	01	01	01	01	01	01	01	*00*	01	01	01	01	01	01	01	*11*	0	0
T28	01	01	01	01	01	01	*00*	01	01	01	01	01	01	01	01	*11*	0	0
T29	01	01	01	01	01	*00*	01	01	01	01	01	01	01	01	01	*11*	0	0
T30	01	01	01	01	*00*	01	01	01	01	01	01	01	01	01	01	*11*	0	0
T31	01	01	01	*00*	01	01	01	01	01	01	01	01	01	01	01	*11*	0	0
T32	01	01	*00*	01	01	01	01	01	01	01	01	01	01	01	01	*11*	0	0
T33	01	*00*	01	01	01	01	01	01	01	01	01	01	01	01	01	*11*	0	0
T34	*00*	01	01	01	01	01	01	01	01	01	01	01	01	01	01	*11*	0	0
T35	01	01	01	01	01	01	01	01	01	01	01	01	01	01	*11*	01	1	0
T36	01	01	01	01	01	01	01	01	01	01	01	01	01	*00*	*11*	01	0	0
T37	01	01	01	01	01	01	01	01	01	01	01	01	*00*	01	*11*	01	0	0
T38	01	01	01	01	01	01	01	01	01	01	01	*00*	01	01	*11*	01	0	0
T39	01	01	01	01	01	01	01	01	01	01	*00*	01	01	01	*11*	01	0	0
T40	01	01	01	01	01	01	01	01	01	*00*	01	01	01	01	*11*	01	0	0
T41	01	01	01	01	01	01	01	01	*00*	01	01	01	01	01	*11*	01	0	0
T42	01	01	01	01	01	01	01	*00*	01	01	01	01	01	01	*11*	01	0	0
T43	01	01	01	01	01	01	*00*	01	01	01	01	01	01	01	*11*	01	0	0
T44	01	01	01	01	01	*00*	01	01	01	01	01	01	01	01	*11*	01	0	0
T45	01	01	01	01	*00*	01	01	01	01	01	01	01	01	01	*11*	01	0	0
T46	01	01	01	*00*	01	01	01	01	01	01	01	01	01	01	*11*	01	0	0
T47	01	01	*00*	01	01	01	01	01	01	01	01	01	01	01	*11*	01	0	0
T48	01	*00*	01	01	01	01	01	01	01	01	01	01	01	01	*11*	01	0	0
T49	*00*	01	01	01	01	01	01	01	01	01	01	01	01	01	*11*	01	0	0
T50	01	01	01	01	01	01	01	01	01	01	01	01	01	*11*	01	01	0	1
T51	01	01	01	01	01	01	01	01	01	01	01	01	*00*	*11*	01	01	0	0
T52	01	01	01	01	01	01	01	01	01	01	01	*00*	01	*11*	01	01	0	0
T53	01	01	01	01	01	01	01	01	01	01	*00*	01	01	*11*	01	01	0	0
T54	01	01	01	01	01	01	01	01	01	*00*	01	01	01	*11*	01	01	0	0
T55	01	01	01	01	01	01	01	01	*00*	01	01	01	01	*11*	01	01	0	0
T56	01	01	01	01	01	01	01	*00*	01	01	01	01	01	*11*	01	01	0	0
T57	01	01	01	01	01	01	*00*	01	01	01	01	01	01	*11*	01	01	0	0
T58	01	01	01	01	01	*00*	01	01	01	01	01	01	01	*11*	01	01	0	0
T59	01	01	01	01	*00*	01	01	01	01	01	01	01	01	*11*	01	01	0	0
T60	01	01	01	*00*	01	01	01	01	01	01	01	01	01	*11*	01	01	0	0
T61	01	01	*00*	01	01	01	01	01	01	01	01	01	01	*11*	01	01	0	0
T62	01	*00*	01	01	01	01	01	01	01	01	01	01	01	*11*	01	01	0	0
T63	*00*	01	01	01	01	01	01	01	01	01	01	01	01	*11*	01	01	0	0
T64	01	01	01	01	01	01	01	01	01	01	01	01	*11*	01	01	01	0	1
T65	01	01	01	01	01	01	01	01	01	01	01	*00*	*11*	01	01	01	0	0

BUPT

Test	Ultimul bit	...	...	...	...	...	...	...	...	...	...	...	...	...	...	Primul bit	CY-in	CY-out
T66	01	01	01	01	01	01	01	01	01	01	*00*	01	*11*	01	01	01	0	0
T67	01	01	01	01	01	01	01	01	01	*00*	01	01	*11*	01	01	01	0	0
T68	01	01	01	01	01	01	01	01	*00*	01	01	01	*11*	01	01	01	0	0
T69	01	01	01	01	01	01	01	*00*	01	01	01	01	*11*	01	01	01	0	0
T70	01	01	01	01	01	01	*00*	01	01	01	01	01	*11*	01	01	01	0	0
T71	01	01	01	01	01	*00*	01	01	01	01	01	01	*11*	01	01	01	0	0
T72	01	01	01	01	*00*	01	01	01	01	01	01	01	*11*	01	01	01	0	0
T73	01	01	01	*00*	01	01	01	01	01	01	01	01	*11*	01	01	01	0	0
T74	01	01	*00*	01	01	01	01	01	01	01	01	01	*11*	01	01	01	0	0
T75	01	*00*	01	01	01	01	01	01	01	01	01	01	*11*	01	01	01	0	0
T76	*00*	01	01	01	01	01	01	01	01	01	01	01	*11*	01	01	01	0	0
T77	00	01	01	01	01	01	01	01	01	01	01	*11*	01	01	01	01	0	1
T78	01	01	01	01	01	01	01	01	01	01	*00*	*11*	01	01	01	01	0	0
T79	01	01	01	01	01	01	01	01	01	*00*	01	*11*	01	01	01	01	0	0
T80	01	01	01	01	01	01	01	01	*00*	01	01	*11*	01	01	01	01	0	0
T81	01	01	01	01	01	01	01	*00*	01	01	01	*11*	01	01	01	01	0	0
T82	01	01	01	01	01	01	*00*	01	01	01	01	*11*	01	01	01	01	0	0
T83	01	01	01	01	01	*00*	01	01	01	01	01	*11*	01	01	01	01	0	0
T84	01	01	01	01	*00*	01	01	01	01	01	01	*11*	01	01	01	01	0	0
T85	01	01	01	*00*	01	01	01	01	01	01	01	*11*	01	01	01	01	0	0
T86	01	01	*00*	01	01	01	01	01	01	01	01	*11*	01	01	01	01	0	0
T87	01	*00*	01	01	01	01	01	01	01	01	01	*11*	01	01	01	01	0	0
T88	*00*	01	01	01	01	01	01	01	01	01	01	*11*	01	01	01	01	0	0
T89	01	01	00	01	01	01	01	01	01	01	*11*	01	01	01	01	01	0	1
T90	01	01	01	01	01	01	01	01	01	*00*	*11*	01	01	01	01	01	0	0
T91	01	01	01	01	01	01	01	01	*00*	01	*11*	01	01	01	01	01	0	0
T92	01	01	01	01	01	01	01	*00*	01	01	*11*	01	01	01	01	01	0	0
T93	01	01	01	01	01	01	*00*	01	01	01	*11*	01	01	01	01	01	0	0
T94	01	01	01	01	01	*00*	01	01	01	01	*11*	01	01	01	01	01	0	0
T95	01	01	01	01	*00*	01	01	01	01	01	*11*	01	01	01	01	01	0	0
T96	01	01	01	*00*	01	01	01	01	01	01	*11*	01	01	01	01	01	0	0
T97	01	01	*00*	01	01	01	01	01	01	01	*11*	01	01	01	01	01	0	0
T98	01	*00*	01	01	01	01	01	01	01	01	*11*	01	01	01	01	01	0	0
T99	*00*	01	01	01	01	01	01	01	01	01	*11*	01	01	01	01	01	0	0
T100	01	01	00	01	01	01	01	01	01	*11*	01	01	01	01	01	01	0	1
T101	01	01	01	01	01	01	01	01	*00*	*11*	01	01	01	01	01	01	0	0
T102	01	01	01	01	01	01	01	*00*	01	*11*	01	01	01	01	01	01	0	0
T103	01	01	01	01	01	01	*00*	01	01	*11*	01	01	01	01	01	01	0	0
T104	01	01	01	01	01	*00*	01	01	01	*11*	01	01	01	01	01	01	0	0
T105	01	01	01	01	*00*	01	01	01	01	*11*	01	01	01	01	01	01	0	0
T106	01	01	01	*00*	01	01	01	01	01	*11*	01	01	01	01	01	01	0	0
T107	01	01	*00*	01	01	01	01	01	01	*11*	01	01	01	01	01	01	0	0
T108	01	*00*	01	01	01	01	01	01	01	*11*	01	01	01	01	01	01	0	0
T109	*00*	01	01	01	01	01	01	01	01	*11*	01	01	01	01	01	01	0	0
T110	01	01	00	01	01	01	01	01	*11*	01	01	01	01	01	01	01	0	1
T111	01	01	01	01	01	01	01	*00*	*11*	01	01	01	01	01	01	01	0	0
T112	01	01	01	01	01	01	*00*	01	*11*	01	01	01	01	01	01	01	0	0
T113	01	01	01	01	01	*00*	01	01	*11*	01	01	01	01	01	01	01	0	0
T114	01	01	01	01	*00*	01	01	01	*11*	01	01	01	01	01	01	01	0	0
T115	01	01	01	*00*	01	01	01	01	*11*	01	01	01	01	01	01	01	0	0
T116	01	01	*00*	01	01	01	01	01	*11*	01	01	01	01	01	01	01	0	0
T117	01	*00*	01	01	01	01	01	01	*11*	01	01	01	01	01	01	01	0	0

BUPT

Test	Ultimul bit	...	...	...	...	...	...	...	...	...	...	...	...	...	...	Primul bit	CY-in	CY-out
T118	*00*	01	01	01	01	01	01	01	*11*	01	01	01	01	01	01	01	0	0
T119	01	01	00	01	01	01	01	*11*	01	01	01	01	01	01	01	01	0	1
T120	01	01	01	01	01	01	*00*	*11*	01	01	01	01	01	01	01	01	0	0
T121	01	01	01	01	01	*00*	01	*11*	01	01	01	01	01	01	01	01	0	0
T122	01	01	01	01	*00*	01	01	*11*	01	01	01	01	01	01	01	01	0	0
T123	01	01	01	*00*	01	01	01	*11*	01	01	01	01	01	01	01	01	0	0
T124	01	01	*00*	01	01	01	01	*11*	01	01	01	01	01	01	01	01	0	0
T125	01	*00*	01	01	01	01	01	*11*	01	01	01	01	01	01	01	01	0	0
T126	*00*	01	01	01	01	01	01	*11*	01	01	01	01	01	01	01	01	0	0
T127	01	01	00	01	01	01	*11*	01	01	01	01	01	01	01	01	01	0	1
T128	01	01	01	01	01	*00*	*11*	01	01	01	01	01	01	01	01	01	0	0
T129	01	01	01	01	*00*	01	*11*	01	01	01	01	01	01	01	01	01	0	0
T130	01	01	01	*00*	01	01	*11*	01	01	01	01	01	01	01	01	01	0	0
T131	01	01	*00*	01	01	01	*11*	01	01	01	01	01	01	01	01	01	0	0
T132	01	*00*	01	01	01	01	*11*	01	01	01	01	01	01	01	01	01	0	0
T133	*00*	01	01	01	01	01	*11*	01	01	01	01	01	01	01	01	01	0	0
T134	01	01	00	01	01	*11*	01	01	01	01	01	01	01	01	01	01	0	1
T135	01	01	01	01	*00*	*11*	01	01	01	01	01	01	01	01	01	01	0	0
T136	01	01	01	*00*	01	*11*	01	01	01	01	01	01	01	01	01	01	0	0
T137	01	01	*00*	01	01	*11*	01	01	01	01	01	01	01	01	01	01	0	0
T138	01	*00*	01	01	01	*11*	01	01	01	01	01	01	01	01	01	01	0	0
T139	*00*	01	01	01	01	*11*	01	01	01	01	01	01	01	01	01	01	0	0
T140	01	01	00	01	*11*	01	01	01	01	01	01	01	01	01	01	01	0	1
T141	01	01	01	*00*	*11*	01	01	01	01	01	01	01	01	01	01	01	0	0
T142	01	01	*00*	01	*11*	01	01	01	01	01	01	01	01	01	01	01	0	0
T143	01	*00*	01	01	*11*	01	01	01	01	01	01	01	01	01	01	01	0	0
T144	*00*	01	01	01	*11*	01	01	01	01	01	01	01	01	01	01	01	0	0
T145	01	01	00	*11*	01	01	01	01	01	01	01	01	01	01	01	01	0	1
T146	01	01	*00*	*11*	01	01	01	01	01	01	01	01	01	01	01	01	0	0
T147	01	*00*	01	*11*	01	01	01	01	01	01	01	01	01	01	01	01	0	0
T148	*00*	01	01	*11*	01	01	01	01	01	01	01	01	01	01	01	01	0	0
T149	01	01	*11*	01	01	01	01	01	01	01	01	01	01	01	01	01	0	1
T150	01	*00*	*11*	01	01	01	01	01	01	01	01	01	01	01	01	01	0	0
T151	*00*	01	*11*	01	01	01	01	01	01	01	01	01	01	01	01	01	0	0
T152	01	*11*	01	01	01	01	01	01	01	01	01	01	01	01	01	01	0	1
T153	00	*11*	01	01	01	01	01	01	01	01	01	01	01	01	01	01	0	0
T154	*11*	01	11	01	01	01	01	01	01	01	01	01	01	01	01	01	0	1

Tabel 4. 6

4.2.2 Evaluarea sumatorului RCLA

Timpul de operare al structurilor de însumare pe n biți de tip RCLA având dimensiunea blocurilor m, este dat de valoarea expresiei: $2 \cdot \left(\frac{n}{m} + 1\right)\tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip RCLA a fost determinat cu ajutorul algoritmului original dat în Fig.A.3 a Anexei A pe baza seturilor minimale de test ale celulor CLA (din Tabel 4. 3, Tabel 4. 4, Tabel 4. 5, Tabel 4. 6); testele obținute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.

BUPT

Lățimea (n)	Dimensiune bloc (m)	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
16	2	18	7	79.36
	4	10	16	62.50
	8	6	46	36.23
32	2	34	7	42.01
	4	18	16	34.72
	8	10	46	21.73
	16	6	154	10.82
64	2	66	7	21.64
	4	34	16	18.38
	8	18	46	12.07
	16	10	154	6.49
	32	6	562	2.96

Tabel 4. 7

A fost demonstrat, prin teorema 2.4, că pentru un sumator RCLA de n biți, având dimensiunea blocurilor m, numărul acestor teste este $\left(\frac{(m+1)\cdot(m+2)}{2}+1\right)$ (independent de n), prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este: $\left[2\cdot\left(\frac{n}{m}+1\right)\cdot\left(\frac{(m+1)\cdot(m+2)}{2}+1\right)\right]^{-1}$. Pe baza relațiilor de mai sus au fost obținute datele de performanță de operare / performabilitate pentru structurile de însumare de tip RCLA (Tabel 4. 7):

Analizând datele din Tabel 4. 7, se observă că pe măsură ce crește dimensiunea blocurilor, sumatorul RCLA seamănă tot mai mult cu un sumator CLA, și deci cu cât sunt mai mari dimesiunile blocurilor, cu atât mai mult se îmbunătățesc performanțele de viteză ale acestor sumatoare și crește numărul testelor necesare testării lor complete în raport cu defectele de blocare singulară la 1 sau 0.
Pe de altă parte, odată cu creșterea dimensiunii blocurilor, fan-in-ul, fan-out-ul cresc și ele. Astfel, cele mai practice dimensiuni de blocuri sunt cele din mijlocul domeniului, adică: 4, sau 8.

În continuare se specifică în mod explicit testele ce trebuiesc aplicate structurilor de însumare de tip RCLA pentru diferite valori ale dimensiunilor blocurilor.

RCLA ce au blocuri de dimensiune m=2

Indiferent de dimensiunea operanzilor ce se însumează, testarea acestor structuri de însumare se face cu un set minimal de teste ce numără 7 teste date în Tabel 4. 8:

	Testul din Tabel 4. 3 ce se aplică blocurilor impare	Testul din Tabel 4. 3 ce se aplică blocurilor pare	C_{-1}
1.	T1	T1	0
2.	T2	T2	1
3.	T6	T6	0
4.	T5	T3	1
5.	T3	T5	0
6.	T7	T4	1
7.	T4	T7	0

Tabel 4. 8.

BUPT

RCLA ce au blocuri de dimensiune m=4

Indiferent de dimensiunea operanzilor ce se însumează, testarea acestor structuri de însumare se face cu un set minimal de teste ce numără 16 teste date în Tabel 4. 9:

	Testul din Tabel 4. 4 ce se aplică blocurilor impare	Testul din Tabel 4. 4 ce se aplică blocurilor pare	C_{-1}
1.	T1	T1	0
2.	T2	T2	1
3.	T8	T8	0
4.	T9	T9	0
5.	T10	T10	0
6.	T12	T12	0
7.	T13	T13	0
8.	T15	T15	0
9.	T3	T7	0
10.	T7	T3	1
11.	T4	T11	0
12.	T11	T4	1
13.	T5	T14	0
14.	T14	T5	1
15.	T6	T16	0
16.	T16	T6	1

Tabel 4. 9

RCLA ce au blocuri de dimensiune m=8

Indiferent de dimensiunea operanzilor ce se însumează, testarea acestor structuri de însumare se face cu un set minimal de teste ce numără 46 teste date în Tabel 4. 10:

Număr teste	Testul din Tabel 4. 5 ce se aplică blocurilor impare	Testul din Tabel 4. 5 ce se aplică blocurilor pare	C_{-1}
1	T1	T1	0
1	T2	T2	1
7	T12 ... T18	T12 ... T18	0
6	T20 ... T25	T20 ... T25	0
5	T27 ... T31	T27 ... T31	0
4	T33 ... T36	T33 ... T36	0
3	T38 ... T40	T38 ... T40	0
2	T42, T43	T42, T43	0
1	T45	T45	0
1	T11	T3	1
1	T3	T11	0
1	T19	T4	1
1	T4	T19	0
1	T26	T5	1
1	T5	T26	0
1	T32	T6	1
1	T6	T32	0
1	T37	T7	1
1	T7	T37	0
1	T41	T8	1
1	T8	T41	0
1	T44	T9	1
1	T9	T44	0
1	T46	T10	1
1	T10	T46	0

Tabel 4. 10

BUPT

RCLA ce au blocuri de dimensiune m=16

Indiferent de dimensiunea operanzilor ce se însumează, testarea acestor structuri de însumare se face cu un set minimal de teste ce numără 154 teste date în Tabel 4. 11:

Număr teste	Testul din Tabel 4. 6 ce se aplică blocurilor impare	Testul din Tabel 4. 6 ce se aplică blocurilor pare	C_{-1}
1	T1	T1	0
1	T2	T2	1
15	T20 ... T34	T20 ... T34	0
14	T36 ... T49	T36 ... T49	0
13	T51 ... T63	T51 ... T63	0
12	T65 ... T76	T65 ... T76	0
11	T78 ... T88	T78 ... T88	0
10	T90 ... T99	T90 ... T99	0
9	T101 ... T109	T101 ... T109	0
8	T111 ... T118	T111 ... T118	0
7	T120 ... T126	T120 ... T126	0
6	T128 ... T133	T128 ... T133	0
5	T135 ... T139	T135 ... T139	0
4	T141 ... T144	T141 ... T144	0
3	T146 ... T148	T146 ... T148	0
2	T150 ... T151	T150 ... T151	0
1	T153	T153	0
1	T19	T3	1
1	T3	T19	0
1	T35	T4	1
1	T4	T35	0
1	T50	T5	1
1	T5	T50	0
1	T64	T6	1
1	T6	T64	0
1	T77	T7	1
1	T7	T77	0
1	T89	T8	1
1	T8	T89	0
1	T100	T9	1
1	T9	T100	0
1	T110	T10	1
1	T10	T110	0
1	T119	T11	1
1	T11	T119	0
1	T127	T12	1
1	T12	T127	0
1	T134	T13	1
1	T13	T134	0
1	T140	T14	1
1	T14	T140	0
1	T145	T15	1
1	T15	T145	0
1	T149	T16	1
1	T16	T149	0
1	T152	T17	1
1	T17	T152	0
1	T154	T18	1
1	T18	T154	0

Tabel 4. 11

BUPT

4.2.3 Evaluarea sumatorului BCLA

Timpul de operare al structurilor de însumare pe n biţi de tip BCLA având dimensiunea blocurilor m, este dat de valoarea expresiei: $2\cdot\left(m+\frac{n}{m}+1\right)\tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip BCLA a fost determinat cu ajutorul algoritmului original dat în Fig.A.4 a Anexei A pe baza seturilor minimale de test ale celulor CLA (din Tabel 4. 3, Tabel 4. 4, Tabel 4. 5, Tabel 4. 6) la care se adaugă testele suplimentare necesare testării complete a sumatoarelor RCA; testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.
A fost demonstrat, prin teorema 2.5, că pentru un sumator BCLA de n biţi, având dimensiunea blocurilor m, numărul acestor teste este $\left(\frac{(m+1)\cdot(m+2)}{2}+7\right)$ (independent de n), prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este: $\left[2\cdot\left(m+\frac{n}{m}+1\right)\cdot\left(\frac{(m+1)\cdot(m+2)}{2}+7\right)\right]^{-1}$. Pe baza relaţiilor de mai sus au fost obţinute datele de performanţă de operare / performabilitate pentru structurile de însumare de tip BCLA (Tabel 4. 12) cele mai uzuale:

Lăţimea (n)	Dimensiune bloc (m)	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
16	2	18	13	42.73
	4	14	22	32.46
	8	18	52	10.68
32	2	34	13	22.62
	4	22	22	20.66
	8	22	52	8.74
	16	34	160	1.83
64	2	66	13	11.65
	4	38	22	11.96
	8	30	52	6.41
	16	38	160	1.64
	32	55	570	0.31

Tabel 4. 12

În Tabel 4. 12 au fost haşurate în cadrul fiecărei grupe valorile maxime ale indicatorului de performabilitate.

4.2.4 Evaluarea sumatoarelor organizate pe superblocuri

4.2.4.1 Evaluarea sumatorului SBCLA

Timpul de operare al structurilor de însumare pe n biţi de tip SBCLA având dimensiunea blocurilor m, iar a superblocurilor M, este dat de valoarea expresiei: $2\cdot\left(\frac{n}{m\cdot M}+M+m-1\right)\tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip SBCLA a fost determinat cu ajutorul algoritmului original dat în Fig.A5 a Anexei A pe baza seturilor minimale de

BUPT

test ale celulor CLA (din Tabel 4. 3, Tabel 4. 4, Tabel 4. 5, Tabel 4. 6) (organizate la nivel de bloc şi superbloc) la care se adaugă testele suplimentare necesare testării complete a sumatoarelor RCA; testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.
A fost demonstrat, prin teorema 2.6, că pentru un sumator SBCLA de n biţi, având dimensiunea blocurilor m, şi cea a superblocurilor M, numărul acestor teste este $\left(\frac{m\cdot(m+3)}{2}+\frac{M\cdot(M+3)}{2}+8\right)$ (independent de n); prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este: $\left[2\cdot\left(\frac{n}{m\cdot M}+M+m-1\right)\cdot\left(\frac{m\cdot(m+3)}{2}+\frac{M\cdot(M+3)}{2}+8\right)\right]^{-1}$. Pe baza relaţiilor de mai sus au fost obţinute datele de performanţă de operare / performabilitate pentru structurile de însumare de tip SBCLA (Tabel 4. 13) mai larg utilizate în practică:

Lăţime (n)	Dimens. grup m	M	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
16	2	2	14	16	44.64
		4	14	25	28.57
	4	2	14	25	28.57
32	2	2	22	16	28.40
		4	18	25	22.22
		8	22	55	8.26
	4	2	18	25	22.22
		4	18	34	16.33
	8	2	22	55	8.26
64	2	2	38	16	16.44
		4	26	25	15.84
		8	26	55	6.99
		16	38	163	1.61
	4	2	26	25	15.84
		4	22	34	13.36
		8	26	64	6.00
	8	2	26	55	6.99
		4	26	64	6.00
	16	2	38	163	1.61

Tabel 4. 13

În Tabel 4. 13 au fost haşurate în cadrul fiecărei grupe valorile maxime ale indicatorului de performabilitate.

4.2.4.2 Evaluarea sumatorului ISBCLA

Timpul de operare al structurilor de însumare pe n biţi de tip ISBCLA având dimensiunea blocurilor m, iar a superblocurilor M, este dat de valoarea expresiei: $2\cdot\left(\frac{n}{m\cdot M}+m+1\right)\tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip ISBCLA a fost determinat cu ajutorul algoritmului original dat în Fig.A5 a Anexei A pe baza seturilor minimale de

BUPT

test ale celulor CLA (din Tabel 4. 3, Tabel 4. 4, Tabel 4. 5, Tabel 4. 6) (organizate la nivel de bloc şi superbloc) la care se adaugă testele suplimentare necesare testării complete a sumatoarelor RCA; testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.
Pentru un sumator ISBCLA de n biţi, având dimensiunea blocurilor m, şi cea a superblocurilor M, numărul acestor teste necesare testării lui complete este $\left(\frac{m\cdot(m+3)}{2}+\frac{M\cdot(M+3)}{2}+8\right)$ (independent de n), prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este: $\left[2\cdot\left(\frac{n}{m\cdot M}+m+1\right)\cdot\left(\frac{m\cdot(m+3)}{2}+\frac{M\cdot(M+3)}{2}+8\right)\right]^{-1}$. Pe baza relaţiilor de mai sus au fost obţinute datele de performanţă de operare / performabilitate pentru structurile de însumare de tip ISBCLA (Tabel 4. 14) cele mai uzuale:

Lăţime (n)	Dimens. m	grup M	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
16	2	2	14	16	44.64
		4	10	25	40.00
	4	2	14	25	28.57
32	2	2	22	16	28.40
		4	14	25	28.57
		8	10	55	18.18
	4	2	18	25	22.22
		4	14	34	21.00
	8	2	22	55	8.26
64	2	2	38	16	16.44
		4	22	25	18.18
		8	14	55	12.98
		16	10	163	6.13
	4	2	26	25	15.38
		4	18	34	16.33
		8	14	64	11.16
	8	2	26	55	6.99
		4	22	64	7.10
	16	2	38	163	1.61

Tabel 4. 14

În Tabel 4. 14 au fost haşurate în cadrul fiecărei grupe valorile maxime ale indicatorului de performabilitate.

4.2.4.3 Evaluarea sumatorului SRCLA

Timpul de operare al structurilor de însumare pe n biţi de tip SRCLA având dimensiunea blocurilor m, iar a superblocurilor M, este dat de valoarea expresiei: $2\cdot\left(\frac{n}{m\cdot M}+3\right)\tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip SRCLA a fost determinat cu ajutorul algoritmului original dat în Fig.A5 a Anexei A pe baza seturilor minimale de test ale celulor CLA (din Tabel 4. 3, Tabel 4. 4, Tabel 4. 5, Tabel 4. 6) (organizate la nivel de

BUPT

bloc şi superbloc); testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.
Pentru un sumator SRCLA de n biţi, având dimensiunea blocurilor m, şi cea a superblocurilor M, numărul acestor teste necesare testării lui complete este $\left(\frac{m\cdot(m+3)}{2}+\frac{M\cdot(M+3)}{2}\right)$ (independent de n), prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este: $\left[2\cdot\left(\frac{n}{m\cdot M}+3\right)\cdot\left(\frac{m\cdot(m+3)}{2}+\frac{M\cdot(M+3)}{2}\right)\right]^{-1}$. Pe baza relaţiilor de mai sus au fost obţinute datele de performanţă de operare / performabilitate pentru structurile de însumare de tip SRCLA (Tabel 4. 15) cele mai uzuale:

	Dimens. grup				
Lăţime (n)	m	M	Timp (τ)	Număr de teste	Performa-bilitate $\cdot 10^{-4}$
16	2	2	14	10	71.42
		4	10	19	52.63
	4	2	10	19	52.53
32	2	2	22	10	45.45
		4	14	19	37.59
		8	10	19	52.63
	4	2	14	19	37.59
		4	10	28	35.71
	8	2	10	49	20.40
64	2	2	38	10	26.31
		4	22	19	23.92
		8	14	19	37.59
		16	10	157	6.36
	4	2	22	19	23.92
		4	14	28	25.51
		8	10	58	17.24
	8	2	14	49	14.57
		4	10	58	17.24
	16	2	10	157	6.36

Tabel 4. 15

În Tabel 4. 15 au fost haşurate în cadrul fiecărei grupe valorile maxime ale indicatorului de performabilitate.

4.2.5 Evaluarea sumatorului PyCLA

Timpul de operare al structurilor de însumare pe n biţi de tip PyCLA are valoarea: $(2\cdot\log_2 n)\ \tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip PyCLA a fost determinat cu ajutorul algoritmului original dat în Fig.A6 a Anexei A; testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.
A fost demonstrat, prin teorema 2.7, că pentru un sumator PyCLA de n biţi, numărul testelor necesare testării lui complete este $\left(8+14\cdot\log_2\frac{n}{2}\right)$ (dependent de n), prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de

BUPT

însumare este: $\left[(2\cdot\log_2 n)\left(8+14\cdot\log_2\frac{n}{2}\right)\right]^{-1}$. Pe baza relațiilor de mai sus au fost obținute datele de performanță de operare/performabilitate pentru structurile de însumare de tip PyCLA având diferite dimensiuni.

Testarea celulelor PyCLA pe 4 ranguri

Tetsele T1÷T14 din Tabel 4. 16 realizează testarea completă a celulelor PyCLA de două ranguri, şi ele aplică pattern-urile înscrise între parantezele rotunde (asociate unui test) tuturor grupelor de două ranguri ce se pot forma la nivelul structurii de însumare care se testează (în cazul structurii PyCLA pe 4 ranguri avem numai două astfel de grupe). Pe de altă parte, pentru testarea completă a celulelor PyCLA de 4 ranguri trebuiesc adăugate testelor deja determinate testele T15-T22 ce aplică pattern-urile înscrise între parantezele rotunde (asociate unui test) tuturor grupelor de patru ranguri ce se pot forma la nivelul structurii de însumare care se testează (în cazul structurii PyCLA pe 4 ranguri avem numai o singură astfel de grupă).

	A_1B_1	A_0B_0	C_{-1}
T1	(0 0	0 0)	0
T2	(0 1	0 1)	0
T3	(1 0	1 0)	0
T4	(0 1	0 1)	1
T5	(1 0	1 0)	1
T6	(1 1	1 1)	1
T7	(1 1	0 0)	1
T8	(0 0	1 1)	1

	A_3B_3	A_2B_2	A_1B_1	A_0B_0	C_{-1}
T9			(0 0	0 1)	0
T10			(0 1	0 0)	0
T11			(1 1	0 1)	1
T12			(0 1	1 1)	1
T13			(0 0	1 1)	0
T14			(1 1	0 0)	1
T15	(0 0	0 0	1 1	1 1)	0
T16	(1 1	1 1	0 0	0 0)	1
T17	(0 0	1 1	1 1	0 0)	0
T18	(1 1	0 0	0 0	1 1)	1
T19	(0 0	0 1	1 1	0 1)	0
T20	(1 1	0 1	0 0	0 1)	1
T21	(0 1	0 0	0 1	1 1)	0
T22	(0 1	1 1	0 1	0 0)	1

Tabel 4. 16

Testarea celulelor PyCLA pe 8 ranguri

Testelor necesare testării complete a celulelor PyCLA pe 4 ranguri (specificate în Tabel 4. 16) li se adaugă testele T23÷T36 din Tabel 4. 17, cu observațiile:

- testele T23÷T28 aplică pattern-urile specificate în Tabel 4. 17 în paralel tuturor grupelor de 4 ranguri ce se pot forma din rangurile structurii de însumare ce se testează
- testele T29÷T36 aplică pattern-urile specificate în Tabel 4. 17 în paralel tuturor grupelor de 8 ranguri ce se pot forma din rangurile structurii de însumare ce se testează.

	A_7B_7	A_6B_6	A_5B_5	A_4B_4	A_3B_3	A_2B_2	A_1B_1	A_0B_0	C_{-1}
T23					(0 0	0 0	0 1	0 1)	0
T24					(1 1	1 1	0 1	0 1)	1
T25					(0 1	0 1	0 0	0 0)	0
T26					(0 1	0 1	1 1	1 1)	1
T27					(0 0	0 0	1 1	1 1)	0
T28					(1 1	1 1	0 0	0 0)	1

BUPT

	A_7B_7	A_6B_6	A_5B_5	A_4B_4	A_3B_3	A_2B_2	A_1B_1	A_0B_0	C_{-1}
T29	(0 0	0 0	0 0	0 0	1 1	1 1	1 1	1 1)	0
T30	(1 1	1 1	1 1	1 1	0 0	0 0	0 0	0 0)	1
T31	(0 0	0 0	1 1	1 1	1 1	1 1	0 0	0 0)	0
T32	(1 1	1 1	0 0	0 0	0 0	0 0	1 1	1 1)	1
T33	(0 0	0 0	0 1	0 1	1 1	1 1	0 1	0 1)	0
T34	(1 1	1 1	0 1	0 1	0 0	0 0	0 1	0 1)	1
T35	(0 1	0 1	0 0	0 0	0 1	0 1	1 1	1 1)	0
T36	(0 1	0 1	1 1	1 1	0 1	0 1	0 0	0 0)	1

Tabel 4. 17

Testarea celulelor PyCLA pe 16 ranguri

Testelor necesare testării complete a celulelor PyCLA pe 8 ranguri (specificate în Tabel 4. 16 şi Tabel 4. 17) li se adaugă testele T37÷T50 din Tabel 4. 18, cu observaţiile:

- testele T37÷T42 aplică pattern-urile specificate în Tabel 4. 18 în paralel tuturor grupelor de 8 ranguri ce se pot forma din rangurile structurii de însumare ce se testează
- testele T43÷T50 aplică pattern-urile specificate în Tabel 4. 18 în paralel tuturor grupelor de 16 ranguri ce se pot forma din rangurile structurii de însumare ce se testează.

	$A_{15}B_{15}A_{14}B_{14}$	...	...	...	$A_7B_7A_6B_6$	...	$A_3B_3A_2B_2$	$A_1B_1A_0B_0$	C_{-1}
T37					(0 0 0 0	00 00	0 1 0 1	0 1 0 1)	0
T38					(1 1 1 1	11 11	0 1 0 1	0 1 0 1)	1
T39					(0 1 0 1	01 01	0 0 0 0	0 0 0 0)	0
T40					(0 0 0 0	00 00	0 1 0 1	0 1 0 1)	1
T41					(0 0 0 0	00 00	1 1 1 1	1 1 1 1)	0
T42					(1 1 1 1	11 11	0 0 0 0	0 0 0 0)	1
T43	(0 0 0 0	00 00	00 00	00 00	1 1 1 1	11 11	1 1 1 1	1 1 1 1)	0
T44	(1 1 1 1	11 11	11 11	11 11	0 0 0 0	00 00	0 0 0 0	0 0 0 0)	1
T45	(0 0 0 0	00 00	11 11	11 11	1 1 1 1	11 11	0 0 0 0	0 0 0 0)	0
T46	(1 1 1 1	11 11	00 00	00 00	0 0 0 0	00 00	1 1 1 1	1 1 1 1)	1
T47	(0 0 0 0	00 00	01 01	01 01	1 1 1 1	11 11	0 1 0 1	0 1 0 1)	0
T48	(1 1 1 1	11 11	01 01	01 01	0 0 0 0	00 00	0 1 0 1	0 1 0 1)	1
T49	(0 1 0 1	01 01	00 00	00 00	0 1 0 1	01 01	1 1 1 1	1 1 1 1)	0
T50	(0 1 0 1	01 01	11 11	11 11	0 1 0 1	01 01	0 0 0 0	0 0 0 0)	1

Tabel 4. 18

Testarea celulelor PyCLA pe 32 ranguri

Testelor necesare testării complete a celulelor PyCLA pe 16 ranguri (specificate în Tabel 4. 16 , Tabel 4. 17 şi Tabel 4. 18) li se adaugă testele T51÷T64 din Tabel 4. 19, cu observaţiile:

- testele T51÷T56 aplică pattern-urile specificate în Tabel 4. 19 în paralel tuturor grupelor de 16 ranguri ce se pot forma din rangurile structurii de însumare ce se testează
- testele T57÷T64 aplică pattern-urile specificate în Tabel 4. 19 în paralel tuturor grupelor de 32 ranguri ce se pot forma din rangurile structurii de însumare ce se testează.

	$A_{31}B_{31} \ldots A_{24}B_{24}$	$A_{23}B_{23} \ldots A_{16}B_{16}$	$A_{15}B_{15} \ldots A_8B_8$	$A_7B_7 \ldots A_0B_0$	C_{-1}
T51			(0 0 ... 0 0	0 1 ... 0 1)	0
T52			(0 1 ... 0 1	0 0 ... 0 0)	0
T53			(1 1 ... 1 1	0 1 ... 0 1)	1

BUPT

	$A_{31}B_{31} \ldots A_{24}B_{24}$	$A_{23}B_{23} \ldots A_{16}B_{16}$	$A_{15}B_{15} \ldots A_8B_8$	$A_7B_7 \ldots A_0B_0$	C_{-1}
T54			(0 1 ... 0 1	1 1 ... 1 1)	1
T55			(1 1 ... 1 1	0 0 ... 0 0)	0
T56			(0 0 ... 0 0	1 1 ... 1 1)	1
T57	(0 0 ... 0 0	0 0 ... 0 0	1 1 ... 1 1	1 1 ... 1 1)	0
T58	(1 1 ... 1 1	1 1 ... 1 1	0 0 ... 0 0	0 0 ... 0 0)	1
T59	(0 0 ... 0 0	1 1 ... 1 1	1 1 ... 1 1	0 0 ... 0 0)	0
T60	(1 1 ... 1 1	0 0 ... 0 0	0 0 ... 0 0	1 1 ... 1 1)	1
T61	(0 0 ... 0 0	0 1 ... 0 1	1 1 ... 1 1	0 1 ... 0 1)	0
T62	(1 1 ... 1 1	0 1 ... 0 1	0 0 ... 0 0	0 1 ... 0 1)	1
T63	(0 1 ... 0 1	0 0 ... 0 0	0 1 ... 0 1	1 1 ... 1 1)	0
T64	0 1 ... 0 1	1 1 ... 1 1	0 1 ... 0 1	0 0 ... 0 0)	1

Tabel 4. 19

Testarea celulelor PyCLA pe 64 ranguri

Testelor necesare testării complete a celulelor PyCLA pe 32 ranguri (specificate în Tabel 4. 16 , Tabel 4. 17, Tabel 4. 18 şi Tabel 4. 19) li se adaugă testele T65÷T78 din Tabel 4. 20, cu observaţiile:

- testele T65÷T70 aplică pattern-urile specificate în Tabel 4. 20 în paralel tuturor grupelor de 32 ranguri ce se pot forma din rangurile structurii de însumare ce se testează
- testele T71÷T78 aplică pattern-urile specificate în Tabel 4. 20 în paralel tuturor grupelor de 64 ranguri ce se pot forma din rangurile structurii de însumare ce se testează.

	$A_{63}B_{63} \ldots A_{48}B_{48}$	$A_{47}B_{47} \ldots A_{32}B_{32}$	$A_{31}B_{31} \ldots A_{16}B_{16}$	$A_{15}B_{15} \ldots A_0B_0$	C_{-1}
T65			(0 0 ... 0 0	0 1 ... 0 1)	0
T66			(0 1 ... 0 1	0 0 ... 0 0)	0
T67			(1 1 ... 1 1	0 1 ... 0 1)	1
T68			(0 1 ... 0 1	1 1 ... 1 1)	1
T69			(1 1 ... 1 1	0 0 ... 0 0)	0
T70			(0 0 ... 0 0	1 1 ... 1 1)	1
T71	(0 0 ... 0 0	0 0 ... 0 0	1 1 ... 1 1	1 1 ... 1 1)	0
T72	(1 1 ... 1 1	1 1 ... 1 1	0 0 ... 0 0	0 0 ... 0 0)	1
T73	(0 0 ... 0 0	1 1 ... 1 1	1 1 ... 1 1	0 0 ... 0 0)	0
T74	(1 1 ... 1 1	0 0 ... 0 0	0 0 ... 0 0	1 1 ... 1 1)	1
T75	(0 0 ... 0 0	0 1 ... 0 1	1 1 ... 1 1	0 1 ... 0 1)	0
T76	(1 1 ... 1 1	0 1 ... 0 1	0 0 ... 0 0	0 1 ... 0 1)	1
T77	(0 1 ... 0 1	0 0 ... 0 0	0 1 ... 0 1	1 1 ... 1 1)	0
T78	0 1 ... 0 1	1 1 ... 1 1	0 1 ... 0 1	0 0 ... 0 0)	1

Tabel 4. 20

4.2.6 Evaluări comparative ale sumatoarelor cu anticiparea transportului

Pentru a realiza o comparare concretă a acestor sumatoare, în Tabel 4. 21, Tabel 4. 22, şi Tabel 4. 23 sunt colectate datele de performanţă ale unor sumatoare cu 3 dimesiuni: 16 cifre binare, 32 cifre binare şi respectiv 64 cifre binare.

La majoritatea cazurilor s-au dat datele celor mai bune sumatoare în termeni de performanţă. Singura excepţie este RCLA-ul pentru care proiectul cu cea mai mare performanţă este CLA-ul în formă pură, şi proiectul cu cel mai mic număr de teste necesar pentru testarea lui completă în raport cu defectele de blocare singulară la 1 sau 0

BUPT

este sumatorul RCA; prin urmare, dimensiunea blocurilor acestui sumator a fost aleasă să corespundă la ceea ce este rezonabil în termenii cerințelor de fan-out şi de fan-in.

Sumator	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
CLA pur	4	154	16.23
RCLA (m=2)	18	7	79.36
(m=4)	10	16	62.50
BCLA (m=2)	18	13	42.73
(m=4)	14	22	32.46
SRCLA (m=2, M=2)	14	10	71,42
SBCLA (m=2, M=2)	14	16	44.64
ISBCLA (m=2, M=2)	14	16	44.64

Tabel 4. 21

Sumator	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
CLA pur	4	562	4.44
RCLA (m=8)	10	46	21.73
(m=4)	18	16	34.72
BCLA (m=4)	22	22	20.66
SRCLA (m=2, M=8)	10	19	52.63
SBCLA (m=2, M=2)	22	16	28.40
ISBCLA (m=2, M=4)	14	25	28.57
(m=4, M=2)	18	25	22.22

Tabel 4. 22

Sumator	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
CLA pur	4	2146	1.16
RCLA (m=2)	66	7	21.64
(m=4)	34	16	18.38
(m=8)	18	46	12.07
BCLA (m=2)	66	13	11.65
(m=4)	38	22	11.96
SRCLA (m=2, M=8)	14	19	37.59
(m=4, M=4)	14	28	25.51
(m=8, M=4)	10	58	17.24
SBCLA (m=2, M=4)	26	25	15.84
(m=4, M=2)	26	25	15.84
(m=2, M=2)	38	16	16.44
ISBCLA (m=2, M=4)	22	25	18.18
(m=4, M=4)	18	34	16.33
(m=8, M=4)	22	64	7.10

Tabel 4. 23

De notat că aceste date tabelate, precum şi formulele folosite nu sunt decât orientative. Cu toate acestea, este de notat faptul că cele mai multe din dimensiunile de bloc şi de superbloc, precum şi datele de performanță relative la o implementare sau alta, corespund de fapt celor care au fost folosite în implementările practice, precum şi la ceea ce este disponibil în mod curent la nivelul multor circuite integrate de la producătorii comerciali.

BUPT

Concluziile ce se desprind din aceste tabele sunt următoarele:

a) Pentru sumatoare mici, întârzierea şi numărul de teste implicate de utilizarea a mai mult de un nivel *lookahead* estompează orice câştig obţinut prin *lookahead*; prin urmare, se vor utiliza proiecte cu un singur nivel de întârziere. Dacă se doreşte optimizarea timpului de operare, şi a numărului de teste se va opta pentru utilizarea sumatorului RCLA

b) Pentru sumatoare mari este recomandată utilizarea fie a structurii SRCLA, fie a structurii ISBCLA. Comparând datele sumatorului BCLA de 64 biţi cu cele ale sumatorului ISBCLA de 64 biţi, rezultă clar care sunt avantajele celui de-al doilea nivel.

4.3 Evaluări ale sumatoarelor cu propagarea cu omitere a transportului

4.3.1 Evaluarea sumatorului CskA cu un nivel de omitere

Timpul de operare al structurilor de însumare pe n biţi de tip CSkA având dimensiunea blocurilor m, este dat de valoarea expresiei: $\left(2 \cdot \frac{n}{m} + 4 \cdot m - 5\right)\tau$, valoarea lui optimală fiind: $\approx 4 \cdot \sqrt{n}$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip CSkA a fost determinat cu ajutorul algoritmului original dat în Fig.A.7 a Anexei A; testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.

Am demonstrat, prin teorema 2.8, că pentru un sumator CSkA de n biţi, având dimensiunea blocurilor m, numărul acestor teste este $(10 + 2 \cdot m)$ (independent de n); prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare are valoarea: $\left[\left(2 \cdot \frac{n}{m} + 4 \cdot m - 5\right) \cdot (10 + 2 \cdot m)\right]^{-1}$

Pe baza relaţiilor de mai sus au fost obţinute datele de performanţă de operare / performabilitate pentru structurile de însumare de tip CSkA (Tabel 4. 24) utilizate în implementările practice:

Lăţimea (n)	Dimensiune bloc (m)	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
16	2	19	14	43.85
	4	19	18	32.89
	8	31	26	13.44
32	2	35	14	23.80
	4	27	18	23.14
	8	35	26	11.90
	16	64	50	3.25
64	2	67	14	12.43
	4	43	18	14.53
	8	43	26	9.68
	16	67	50	3.10
	32	127	74	1.09

Tabel 4. 24

BUPT

În Tabel 4. 24 au fost haşurate în cadrul fiecărei grupe valorile maxime ale indicatorului de performabilitate.

Cele $(10 + 2 \cdot m)$ teste necesare testării complete pentru detecţia defectelor de blocare singulară la 1 sau 0 a acestor structuri se obţin pe baza:

- celor 8 teste descrise în Tabel 4. 1 şi aplicate cu semnalul Test=1, la care se adaugă
- testele T_4 şi T_5 ale aceluiaşi tabel, aplicate împreună cu semnalul semnalul Test=0 şi
- cele $2 \cdot m$ teste suplimentare necesare testării logicii CY-skip ce sunt prezentate în Tabel 4. 25

	C_{-1}	Blocuri impare A_i B_i ...	Blocuri pare A_i B_I ...
T_{10}	1	0 0 0 1 0 1... 0 1	0 0 0 1 0 1... 0 1
T_{11}	1	0 0 0 1 0 1... 1 1	0 0 0 1 0 1... 0 1
T_{12}	1	0 1 0 0 0 1... 0 1	0 1 0 0 0 1... 1 1
T_{13}	1	0 1 0 0 0 1... 1 1	0 1 0 0 0 1... 0 1
		...	
$T_{10+2m-1}$	1	0 1 0 1 0 1... 0 0	0 1 0 1 0 1... 1 1
T_{10+2m}	1	0 1 0 1 0 1... 1 1	0 1 0 1 0 1... 0 0

Tabel 4. 25

4.3.2 Evaluarea sumatorului CSkA multinivel

Timpul de operare al structurilor de însumare pe n biţi de tip CSkA multinivel organizat pe 2 niveluri (având dimensiunea blocurilor m, şi cea a superblocurilor M) este dat de valoarea expresiei: $\left(4 \cdot m + 4 \cdot M + \frac{2 \cdot n}{m \cdot M} - 12\right)\tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip CSkA cu două niveluri de omitere a fost determinat cu ajutorul algoritmului original dat în Fig.A.8 a Anexei A; testele obţinute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.

Am demonstrat, prin teorema 2.9, că pentru un sumator CSkA multinivel de n biţi, având dimensiunea blocurilor m, şi cea a superblocurilor M, numărul acestor teste este $(11 + 2 \cdot m + 2 \cdot M)$ (independent de n); prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este:

$$\left[\left(4 \cdot m + 4 \cdot M + \frac{2 \cdot n}{m \cdot M} - 12\right) \cdot (11 + 2 \cdot m + 2 \cdot M)\right]^{-1}$$

Pe baza relaţiilor de mai sus au fost obţinute datele de performanţă de operare / performabilitate pentru structurile de însumare de tip CSkA multinivel (Tabel 4. 26) utilizate în implementările practice.

În Tabel 4. 26 au fost haşurate în cadrul fiecărei grupe valorile maxime ale indicatorului de performabilitate.

BUPT

	Dimens.grup				
Lățime (n)	m	M	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
16	2	2	12	19	43.85
		4	16	23	27.17
	4	2	16	23	27.17
32	2	2	20	19	26.31
		4	20	23	21.73
		8	32	31	10.08
	4	2	20	23	21.73
		4	24	27	15.43
	8	2	32	31	10.08
64	2	2	36	19	14.61
		4	28	23	15.52
		8	36	31	8.96
		16	64	47	3.32
	4	2	28	23	15.52
		4	28	27	13.22
		8	40	35	7.14
	8	2	36	31	8.96
		4	40	35	7.14
	16	2	64	47	3.32

Tabel 4. 26

În Tabel 4. 27 sunt date cele $(2 \cdot M + 1)$ teste suplimentare (celor specificate în secțiunea 5.3.1) necesare testării logicii CY-*skip* a superblocurilor acestor sumatoare. În prezentarea pattern-urilor de test au fost grupate între paranteze rotunde intrările aplicate unui superbloc, o intrare fiind formată dintr-o pereche A_i B_i, $i = \overline{0, m-1}$.

Test	C_{-1}	Intrări ale super-blocurilor impare	Intrări ale super-blocurilor pare	
$T_{10 \cdot 2m \cdot 1}$	1	(00 ... 00) (11 ... 11) ... (11 ... 11) (11 ... 11)	(00 ... 00) (11 ... 11) ... (11 ... 11) (11 ... 11)	
$T_{10 \cdot 2m \cdot 2}$	1	(00 ... 00) (01 ... 01) ... (01 ... 01) (01 ... 01)	(00 ... 00) (01 ... 01) ... (01 ... 01) (11 ... 11)	K_0 s-a-0
$T_{10 \cdot 2m \cdot 3}$	1	(00 ... 00) (01 ... 01) ... (01 ... 01) (11 ... 11)	(00 ... 00) (01 ... 01) ... (01 ... 01) (01 ... 01)	K_0 s-a-0
$T_{10 \cdot 2m \cdot 4}$	1	(01 ... 01) (00 ... 00) ... (01 ... 01) (01 ... 01)	(01 ... 01) (00 ... 00) ... (01 ... 01) (11 ... 11)	K_0 s-a-0
$T_{10 \cdot 2m \cdot 5}$	1	(01 ... 01) (00 ... 00) ... (01 ... 01) (11 ... 11)	(01 ... 01) (00 ... 00) ... (01 ... 01) (01 ... 01)	K_0 s-a-0
...				
$T_{10 \cdot 2m \cdot 2M}$	1	(01 ... 01) (01 ... 01) ... (01 ... 01) (00 ... 00)	(01 ... 01) (01 ... 01) ... (01 ... 01) (11 ... 11)	K_0 s-a-0
$T_{10 \cdot 2m \cdot 2M \cdot 1}$	1	(01 ... 01) (01 ... 01) ... (01 ... 01) (11 ... 11)	(01 ... 01) (01 ... 01) ... (01 ... 01) (00 ... 00)	K_0 s-a-0

Tabel 4. 27

4.4 Evaluări ale sumatoarelor cu propagarea specială a transportului

4.4.1 Evaluarea sumatorului LR-CSIA

Timpul de operare al structurilor de însumare pe *n* biți de tip LR-CSIA având dimensiunea blocurilor *m*, are valoarea dată de următoarea expresie $\left[6 + \left(2 \cdot \frac{n}{m} - 1\right)\right] \cdot \tau$.

Setul minimal de teste necesare testării complete în raport cu defectele de blocare singulară la 1 sau 0 a structurilor de însumare de tip LR-CSIA a fost determinat

BUPT

cu ajutorul algoritmului prezentat în Fig.A.3 a Anexei A; testele obținute au fost validate ca fiind acoperitoare şi minimale cu ajutorul mediului LDL.

Pentru un sumator LR-CSIA de n biți, având dimensiunea blocurilor m, numărul acestor teste este $\cdot\left(1+\frac{(m+1)\cdot(m+2)}{2}\right)$ (independent de n); prin urmare, valoarea indicatorului de performabilitate pentru acest tip de structuri de însumare este: $\left[\left(6+\left(2\cdot\frac{n}{m}-1\right)\right)\cdot\left(1+\frac{(m+1)\cdot(m+2)}{2}\right)\right]^{-1}$

Pe baza relațiilor de mai sus au fost obținute datele de performanță de operare / performabilitate pentru structurile de însumare de tip LR-CSIA (Tabel 4. 28) utilizate în implementările practice:

Lățimea (n)	Dimensiune bloc (m)	Timp (τ)	Număr de teste	Performabilitate $\cdot 10^{-4}$
16	2	21	7	68.02
	4	13	16	48.07
	8	9	46	24.15
32	2	37	7	38.61
	4	21	16	29.76
	8	13	46	16.72
	16	9	154	7.21
64	2	69	7	20.70
	4	37	16	16.89
	8	21	46	10.35
	16	13	154	4.99
	32	9	562	1.97

Tabel 4. 28

În Tabel 4. 28 au fost haşurate în cadrul fiecărei grupe valorile maxime ale indicatorului de performabilitate.

Pentru anumite cazuri, valorile teoretice optime ale anumitor parametrii (dimensiunea blocurilor de exemplu) se poate întâmpla să nu poată fi atinse întrucât ele nu rezultă ca şi valori întregi. Din acest motiv se impune ca şi criteriu de comparație valorile lui n folosite în analizele tabelare ale sumatoarelor analizate.

4.5 Concluzii

Prin rularea pentru structurile de însumare analizate a modulului GVST - Poage a mediului integrat LDL s-a realizat validarea numărului minimal de teste identificate în cadrul capitolului 2 la nivelul fiecărei structuri.

Pe de altă parte, validarea acoperirii complete a defectelor de blocare singulară la 1 sau 0 la nivelul structurilor de însumare, pentru testele obținute pe baza algoritmilor originali (prezentați în Anexa A) de generare automată a vectorilor stimuli de test, a fost realizată cu ajutorul simulatorului de defecte - modulul SPD a mediului integrat LDL.
În felul acesta s-a realizat validarea completă a contribuțiilor originale în domeniul generării automate a seturilor minimale de teste necesare detecției SSF la nivelul structurilor de însumare.

BUPT

Întrucât algoritmii obținuți de autoare sunt aplicabili în cadrul unei structuri de însumare pentru orice dimensiune a operanzilor ce se adună, rezultatul acestei validări se traduce printr-un câștig cert la nivelul producătorului a cărui consum de timp pentru generarea automată a testelor (descris în Fig.2.17.) se reduce la utilizarea directă a algoritmilor optimali elaborați. Sunt eliminate astfel toate backtracking-urile implicate de un modul de generare algoritmică a testelor, precum și timpii consumați pentru simularea defectelor și evaluarea acoperirilor.
Pe de altă parte, testarea structurilor de însumare cu seturile minimale de stimuli determinate, comparativ cu testarea lor aleatoare sau pseudoexhaustivă bazată pe partiționare (analizate în Capitolul 3) duce la un câștig net și sub aspectul timpilor implicați în procesul de testare propriu-zis, datorită minimalității numărului de teste a seturilor determinate.
Un alt mare avantaj conferit de utilizarea algoritmilor optimali obținuți de autoare este legat de posibilitatea utilizării lor pentru testarea sistemelor cu facilități BIST *off-line* ce prezintă astfel de structuri de însumare în componența lor; testarea acestora se poate face prin rularea algoritmilor specifici structurilor respective.

Sumator	Timp (τ)	Număr de teste	Performabilitate
CCA (medie)	12	8	104.16
SRCLA (m=2, M=2)	14	10	71.42
CSIA (m=2)	21	7	68.02
RCLA (m=4)	10	16	62.50
ISBCLA (m=2, M=2)	14	16	44.64
SBCLA (m=2, M=2)	14	16	44.64
CSkA pe 2 nivele (m=2, M=4)	12	19	43.85
CSkA pe un nivel (m=2)	19	14	43.85
RCA	31	8	40.32
RCLA (m=8)	6	46	36.23
BCLA (m=4)	14	22	32.46
PyCLA	9	50	22.22
CLA în formă pură	4	154	16.23

Tabel 4. 29

Sumator	Timp (τ)	Număr de teste	Performabilitate
CCA (medie)	14	8	89.28
SRCLA (m=2, M=8)	10	19	52.63
CSIA (m=2)	37	7	38.61
RCLA (m=4)	18	16	34.72
ISBCLA (m=2, M=4)	10	25	28.57
SBCLA (m=2, M=2)	18	16	28.40
CSkA pe 2 nivele (m=2, M=2)	20	19	26.31
CSkA pe un nivel (m=2)	35	14	23.80
RCLA (m=8)	10	46	21.73
BCLA (m=4)	22	22	20.66
RCA	63	8	19.84
CLA în formă pură	4	561	17.82
PyCLA	11	64	14.20

Tabel 4. 30

Pe baza numărului minim de teste determinat pentru fiecare structură de însumare în parte și pe baza timpilor de operare determinați prin analiza funcțională din Capitolul 2,

BUPT

s-a realizat o evaluare comparativă prin prisma indicatorului de performabilitate propus (Capitolul 2) a structurilor de însumare având dimensiunile date de implementările practice existente, rezultând pe baza acestei comparații datele cu referire la sumatoarele pe 16 biți prezentate în Tabel 4. 29, cele cu referire la sumatoarele pe 32 biți prezentate în Tabel 4. 30 şi cele cu referire la sumatoarele pe 32 cifre binare prezentate în Tabel 4. 31

Sumator	Timp (τ)	Număr de teste	Performabilitate
CCA (medie)	16	8	78.125
SRCLA (m=4, M=4)	14	28	25.51
RCLA (m=4)	34	16	18.38
SRCLA (m=8, M=4)	10	58	17.24
CSlA (m=4)	37	16	16.89
ISBCLA (m=4, M=4)	18	34	16.33
SBCLA (m=4, M=2)	22	25	15.84
CSkA pe 2 nivele (m=2, M=4)	28	23	15.52
CSkA pe un nivel (m=4)	29	18	14.53
RCLA (m=8)	18	46	12.07
BCLA (m=4)	38	22	11.96
PyCLA	13	78	9.86
RCA	127	8	9.84
ISBCLA (m=8, M=4)	14	64	7.10
BCLA (m=8)	30	52	6.41
CLA în formă pură	4	2080	4.80

Tabel 4. 31

Pe baza rezultatelor tabelate în Tabel 4. 29, Tabel 4. 30 şi Tabel 4. 31 se concluzionează prin a afirma că:

Cea mai performabilă structură de însumare pe:

- 16 cifre binare este: structura de tip CCA, urmată de cea de tip SRCLA cu m=2 şi M=2
- 32 cifre binare este: structura de tip CCA, urmată de cea de tip SRCLA cu m=2 şi M=8
- 64 cifre binare este: structura de tip CCA, urmată de cea de tip SRCLA cu m=4 şi M=4

Cea mai puțin performabilă structură de însumare pe:

- 16 cifre binare este: structura CLA în formă pură, urmată de cea de tip BCLA
- 32 cifre binare este: structura CLA în formă pură, urmată de cea de tip BCLA
- 64 cifre binare este: structura CLA în formă pură, urmată de cea de tip BCLA

BUPT

5. Concluzii

Lucrarea aparține domeniului de calcul, situându-se la joncțiunea domeniului de calcul cu cel al fiabilității; mai exact, ținta cercetărilor este o parte a echipamentului de calcul, şi anume, unitatea de însumare a acestuia.

Această joncțiune de domenii se poate urmări în întreaga construcție a lucrării, unde în mod original, s-a făcut legătura şi pe capitole între aceste chestiuni.

Propagarea pe două direcții a semnalelor activate pe parcursul unei adunări, plasează structurile de însumare într-o subclasă specială a circuitelor combinaționale, şi ținând cont de importanța la nivelul sistemelor de calcul a funcției implementate de ele, în contextul dinamicii tehnologice la care asistăm, consider pe deplin justificată cercetarea întreprinsă în direcția creşterii performanței operaționale în conjucție şi a celor de testabilitate ale acestor structuri.

Orientarea cercetărilor în direcția structurilor de însumare a avut drept argument decisiv frecvența mare a operațiilor de însumare la nivelul sistemelor de calcul. Astfel, se poate afirma că performanța sistemelor de calcul este influențată capital de cea a structurilor de însumare care stau la baza implementării Unității Aritmetico-Logice.
În cadrul tezei, problema creşterii performanțelor de testabilitate a structurilor de însumare a fost abordată în două mari direcții:

- Identificare a unor algoritmi optimali de generare automată a stimulilor pentru testarea structurilor de însumare, care ținând cont de analiza funcțională să permită obținerea unui set minim de teste pentru detecție;
- Elaborarea unor soluții eficiente de reconfigurare a structurilor de însumare care să permită autotestarea lor; soluționările de reconfigurare au vizat pe de o parte testarea *off-line* a structurilor de însumare, pe de altă parte testarea lor *on-line*.

Elementele esențiale care au stat la baza cercetărilor întreprinse de autoare au fost:

- Analiza funcțională intimă a structurilor de însumare
- Exploatarea proprietății de repetabilitate spațială a structurilor de însumare
- Aplicarea conceptului C-testabilității
- Reconfigurarea structurilor – în situații bine justificate – în vederea realizării dezideratului C-testabilității

Rezultatele obținute de autoare, confirmă pe deplin orientarea sugerată de conducătorul ştiințific în direcția structurilor de însumare, soluțiile elaborate demonstrând clar, oportunitățile ample de cercetare oferite în domeniul menționat, cu implicații directe asupra creşterii testabilității acestor structuri.

Contribuțiile originale ale autoarei la prezenta teză, pot fi sintetizate astfel:

1. Prezentarea unei analize funcționale la nivel intim a structurilor de însumare, elaborându-se relații de calcul pentru performanțele lor de operare în termenii numărului maxim de porți traversate de semnalele activate în cadrul procesului de însumare.
2. Clasificarea într-o manieră originală, a structurilor de însumare funcție de modul de propagare a transportului, permițând astfel o abordare sistematică a problematicilor specifice, aceasta deoarece un defect apărut la nivelul lanțului de propagare şi/sau generare a semnalului CY va afecta întotdeauna bitul sumă ce include semnalul respectiv, dar poate afecta şi valorile biților sumă mai semnificative în raport cu valoarea bitului respectiv şi eventual valoarea transportului generat prin însumarea celor doi operanzi.

BUPT

3. Obținerea seturilor minimale pentru testarea completă a structurilor de însumare, în raport cu defectele de blocare singulară la 1 sau 0, prin exploatarea proprietății de repetabilitate spațială a acestor structuri, aplicând conceptul C-testabilității, realizând acolo unde a fost cazul (structuri CSkA, respectiv CCA) reconfigurări ale structurilor în vederea realizării dezideratului C-testabilității.
4. Enunțarea și demonstrarea unui număr de 8 teoreme și 2 corolare originale, care permit determinarea imediată a numărului minim de teste necesar pentru următoarele tipuri de structuri de însumare:
   - 4.1 Sumator de tip CCA (Teorema 2.2)
   - 4.2 Sumator de tip RCLA (Teorema 2.3)
   - 4.3 Sumator de tip BCLA (Teorema 2.4)
   - 4.4 Sumator de tip CLA (Teorema 2.5)
   - 4.5 Sumator de tip SBCLA (Teorema 2.6)
   - 4.6 Sumator de tip PyCLA(Teorema 2.7)
   - 4.7 Sumator de tip CSkA cu un nivel de omitere (Teorema 2.8 și Corolar 2.8.1)
   - 4.8 Sumator de tip CSkA cu două nivele de omitere (Teorema 2.9 și Corolar 2.9.1)

   Cele două corolare se referă la identificarea imediată a numărului minim de teste necesar pentru testarea structurilor de însumare cu omiterea transporturilor la care, pentru optimizarea performanțelor de operare, dimensiunea blocurilor este variabilă.
5. Elaborarea a 9 algoritmi optimali originali de generare automată a seturilor minimale de teste pentru testarea completă în raport cu defectele de blocare singulară a structurilor de însumare analizate. Astfel, autoarea apreciază că prin algoritmii originali obținuți ea a reușit să elimine:
   - *Backtracking*-urile implicate de metodele clasice de generare algoritmică a testelor
   - Timpii consumați pentru simularea defectelor
   - Timpii consumați pentru evaluarea acoperirilor defectelor structurilor testate

   Având în vedere aceste aspecte, precum și faptul că algoritmii prezentați conduc întotdeauna la numărul minim de teste, autoarea a calificat acești algoritmi ca și "*algoritmi optimali*" de generare automată a testelor.
6. Reducerea consumului de timp pentru generarea automată a testelor la producătorii de circuite, prin utilizarea directă a algoritmilor optimali elaborați, aceasta deoarece algoritmii obținuți de autoare sunt aplicabili în cadrul unei structuri de însumare pentru orice dimensiune a operanzilor ce se adună.
7. Posibilitatea utilizării algoritmilor optimali pentru testarea sistemelor cu facilități BIST *off-line*
8. Evaluarea comparativă a structurilor de însumare prin prisma indicatorului de performabilitate, pentru dimensiuni date de implementările practice existente; bazat pe această evaluare, realizată la nivelul dimensiunilor de 16, 32 și 64 cifre binare, cea mai performabilă structură de însumare a rezultat ca fiind cea de tip CCA
9. Validarea experimentală a rezultatelor obținute în cadrul tezei, constând în următoarele:

BUPT

9.1 Elaborarea mediului LDL (descris în Anexa 2) ca instrument cu puternice facilităţi în testarea circuitelor logice
9.2 Validarea acoperirii complete, prin algoritmii optimali elaboraţi, a defectelor de blocare singulară la 1 sau 0, la nivelul structurilor de însumare analizate cu ajutorul simulatorului SPD a mediului LDL.
9.3 Validarea numărului minimal de teste identificate la nivelul fiecărei structuri, rulând modulul GVST-Poage a mediului integrat LDL pentru structurile de însumare analizate.

In final, autoarea apreciază că ansamblul contribuţiilor originale dezvoltate în cadrul tezei, deschid o perspectivă nouă producătorilor de circuite integrate ce înglobează structuri de însumare complexe, cu implicaţii certe în creşterea testabilităţii acestora şi, respectiv, în scăderea preţului de cost al circuitelor la producător, bazat pe cele anterior menţionate.

De asemenea, autoarea poate afirma că problematica abordată în cadrul tezei, deschide teren pentru extinderea cercetărilor în acceaşi manieră şi asupra altor tipuri de structuri, propunându-şi în viitor, bazat pe rezultatele prezentei teze, dezvoltarea unor metode de proiectare pentru testabilitate optimală.

BUPT

Anexa A

```
Algoritm CLA_1()
Begin
    For i = 1,n formează (A_i = 0, B_i = 1)
    aplică testul obținut împreună cu C_-1 = 0
    For i = 1,n formează (A_i = 1, B_i = 0)
    aplică testul obținut împreună cu C_-1 = 1
    For i = 1,n
        Begin
            A_j = 0, B_j = 0, C_-1 = 1 și
            For (i = 1,n), i ≠ j   A_i = 0, B_i = 1
            aplică testul obținut
        end
    For i = 1,n
        Begin
            A_j = 1, B_j = 1, C_-1 = 0 și
            For (i = 1,n), i ≠ j   A_i = 0, B_i = 1
            aplică testul obținut
        end

    For k = 1,n
        Begin
            A_k = 1, B_k = 1, C_-1 = 0
            For j = k + 1,n
                Begin
                    A_j = 0, B_j = 0
                    For (i = 1,n), i ≠ j, i ≠ k
                        A_i = 0, B_i = 1
                    aplică testul obținut
                end
        end
end
```

Fig.A 1

```
Algoritm RCLA_1()
Begin
    For i = 0,2^m - 1
        For j = 0,2^m - 1
            Begin
                C_-1 = 0
                If i + j + C_-1 ≤ 2^m - 1
                    Begin
                        asignează pentru fiecare bloc de 2^m ranguri
                                valoarea i_2 operandului A și
                                valoarea j_2 operandului B
                        Aplică testul obținut
                    end
                else
                    Begin
```


BUPT

```
                    asignează pentru fiecare bloc par de 2^m ranguri
                              valoarea ((2^m - 1) - i)_2 operandului A şi
                              valoarea ((2^m - 1) - j)_2 operandului B
                    asignează pentru fiecare bloc impar de 2^m ranguri
                              valoarea i_2 operandului A şi
                              valoarea j_2 operandului B
                    Aplică testul obţinut
               end
          C_{-1} = 1
          If i + j + C_{-1} ≤ 2^m - 1
               Begin
                    asignează pentru fiecare bloc impar de 2^m ranguri
                              valoarea ((2^m - 1) - i)_2 operandului A şi
                              valoarea ((2^m - 1) - j)_2 operandului B
                    asignează pentru fiecare bloc par de 2^m ranguri
                              valoarea i_2 operandului A şi
                              valoarea j_2 operandului B
                    Aplică testul obţinut
               end
          else
               Begin
                    asignează pentru fiecare bloc de 2^m ranguri
                              valoarea i_2 operandului A şi
                              valoarea j_2 operandului B
                    Aplică testul obţinut
               end
     end
end
```

Fig.A 2

```
Algoritm RCLA 2()
Begin
    For i = 1,n asignează (A_i = 0, B_i = 1)
    aplică testul obţinut împreună cu C_{-1} = 0
    For i = 1,n asignează (A_i = 1, B_i = 0)
    aplică testul obţinut împreună cu C_{-1} = 1
    For j = 1,m
        Begin
            C_{-1} = 1
            Asignează rangului j al blocurilor pare A_j = 0, B_j = 0 , şi al celor impare A_j = 1, B_j = 1
            Pentru toate blocurile sumatorului Do For (i = 1,m), i ≠ j asignează A_i = 0, B_i = 1
            aplică testul obţinut
            C_{-1} = 0
            Asignează rangului j al blocurilor impare A_j = 0, B_j = 0, şi al celor pare A_j = 1, B_j = 1
            aplică testul obţinut
        end
    For k = 1,m
        Begin
            C_{-1} = 0 şi asignează rangului k al tuturor blocurilor sumatorului RCLA: A_k = 1, B_k = 1
```


BUPT

For $j = k+1, n$

Begin

asignează rangului j *al tuturor blocurilor sumatorului* $A_j = 0$, $B_j = 0$

Pentru toate blocurile sumatorului **Do**:

For $(i = 1, n), i \neq j, i \neq k$

asignează $A_i = 0$, $B_i = 1$

aplică testul obținut

end

end

end

Fig.A 3

Algoritm BCLA()

Begin

For $i = 0, n-1$

asignează $A_i = 0$, $B_i = 0$ *și* $C_{-1} = 0$

aplică testul obținut

For $i = 0, n-1$

asignează $A_i = 0$, $B_j = 1$ *și* $C_{-1} = 0$

aplică testul obținut

la aceeași asignare pentru operanzii A *și* B, *asignează* $C_{-1} = 1$

aplică testul obținut

For $i = 0, n-1$

Begin

asignează $A_i = 1$, $B_j = 0$ *și* $C_{-1} = 0$

aplică testul obținut

la aceeași asignare pentru operanzii A *și* B, *asignează* $C_{-1} = 1$

aplică testul obținut

end

For $i = 0, n-1$

asignează $A_i = 1$, $B_i = 1$ *și* $C_{-1} = 1$

aplică testul obținut

$C_{-1} = 0$

Repeat

i=0

asignează $A_i = 1$, $B_i = 1$, $A_{i+1} = 0$, $B_{i+1} = 0$

i=i+2

until $i \geq n$

aplică testul obținut

$C_{-1} = 1$

Repeat

i=0

asignează $A_i = 0$, $B_i = 0$, $C_{-1} = 1$, $B_{i+1} = 1$

i=i+2

until $i \geq n$

aplică testul obținut

For $j = 1, m$

Begin

$C_{-1} = 1$

Asignează rangului j *al blocurilor pare* $A_j = 0$, $B_j = 0$, *și al celor impare* $A_j = 1$, $i = 1, n$

Pentru toate blocurile sumatorului: **Do For** $(i = 1, m), i \neq j$ *asignează* $A_i = 0$, $B_i = 1$

BUPT

aplică testul obținut
$C_{-1} = 0$
Asignează rangului j *al blocurilor impare* $A_j = 0$, $B_j = 0$ *,și al celor pare* $A_j = 1$, $B_j = 1$
aplică testul obținut
end
For $k = 1, m$
Begin
$C_{-1} = 0$ *și asignează rangului* k *al tuturor blocurilor sumatorului BCLA:* $A_k = 1$, $B_k = 1$
For $j = k + 1, m$
Begin
asignează rangului j *al tuturor blocurilor sumatorului* $A_j = 0$, $B_j = 0$
Pentru toate blocurile sumatorului ***Do For*** $(i = 1, m), i \neq j, i \neq k$
asignează $A_j = 0$, $B_i = 1$
aplică testul obținut
end
end
end

Fig.A 4

Algoritm SBCLA ()
Begin
For $i = 0, n - 1$
asignează $A_i = 0$, $B_i = 0$ *și* $C_{-1} = 0$
aplică testul obținut
For $i = 0, n - 1$
asignează $A_i = 0$, $B_i = 1$ *și* $C_{-1} = 0$
aplică testul obținut
la aceeași asignare pentru operanzii A *și* B, *asignează* $C_{-1} = 1$
aplică testul obținut
For $i = 0, n - 1$
Begin
asignează $A_i = 1$, $B_i = 0$ *și* $C_{-1} = 0$
aplică testul obținut
la aceeași asignare pentru operanzii A *și* B, *asignează* $C_{-1} = 1$
aplică testul obținut
end
For $i = 0, n - 1$
asignează $A_i = 1$, $B_i = 1$ *și* $C_{-1} = 1$
aplică testul obținut
$C_{-1} = 0$
Repeat
$i \leftarrow 0$
asignează $A_i = 1$, $B_i = 1$, $A_{i+1} = 0$, $B_{i+1} = 0$
$i \leftarrow i + 2$
until $i \geq n$
aplică testul obținut
$C_{-1} = 1$
Repeat
$i \leftarrow 0$
asignează $A_i = 0$, $B_i = 0$, $A_{i+1} = 1$, $B_{i+1} = 1$
$i \leftarrow i + 2$

BUPT

```
until i ≥ n
aplică testul obținut
For j = 1,m
   Begin
      C_-1 = 1
      Asignează rangului j al blocurilor pare A_j = 0 , B_j = 0 , și al celor impare A_j = 1 , B_j = 1
      Pentru toate blocurile sumatorului Do: For (i = 1,m),i ≠ j asignează A_i = 0, B_i = 1
      aplică testul obținut
      C_-1 = 0
      Asignează rangului j al blocurilor impare A_j = 0 , B_j = 0 , și al celor pare A_j = 1 , B_j = 1
      aplică testul obținut
   end
For k = 1,m
   Begin
      C_-1 = 0 și asignează rangului k al tuturor blocurilor sumatorului SBCLA: A_k = 1, B_k = 1
      For j = k + 1,m
         Begin
            asignează rangului j al tuturor blocurilor sumatorului A_j = 0, B_j = 0
            Pentru toate blocurile sumatorului
               For (i = 1,m),i ≠ j,i ≠ k asignează A_i = 0, B_i = 1
            aplică testul obținut
         end
   end
For j = 1,M
   Begin
      C_-1 = 1
      Asignează blocului j al superbloc. pare
         For (i = 1,m) A_i = 0, B_i = 0
      Asignează blocului j al superbloc. impare
         For (i = 1,m) A_i = 1, B_i = 1
      Pentru toate super-blocurile
            Pentru toate blocurile k , k ≠ j ale sumatorului:
               Do For (i = 1,m) asignează A_i = 0, B_i = 1
      aplică testul obținut
      C_-1 = 0
      Asignează blocului j al superbloc.impare
         For (i = 1,m) A_i = 0, B_i = 0
      Asignează blocului j al superbloc. pare
         For (i = 1,m) A_i = 1, B_i = 1
      aplică testul obținut
   end
For k = 1,M
   Begin
      C_-1 = 0
      Asignează blocului k al tuturor super-blocurilor sumatorului SBCLA:
         For (i = 1,m) A_k = 1, B_k = 1
      For j = k + 1,M
         Begin
            Asignează blocului j al tuturor super-blocurilor sumatorului
               For (i = 1,m) A_i = 0, B_i = 0
            pentru toate blocurile r (r ≠ j,r ≠ k) ale sumatorului:
```


BUPT

```
                    Do For (i = 1,m),
                        Asignează A_i = 0, B_i = 1
                  aplică testul obținut
                end
            end
end
```

Fig.A 5

```
Algoritm PyCLA ()
Begin
    For i = 0,n-1
            asignează A_i = 0, B_i = 0 şi C_-1 = 0
            aplică testul obținut
    For i = 0,n-1
            asignează A_i = 0, B_i = 1 şi C_-1 = 0
            aplică testul obținut
            la aceeaşi asignare pentru operanzii A şi B, asignează C_-1 = 1
            aplică testul obținut
    For i = 0,n-1
        Begin
            asignează A_i = 1, B_i = 0 şi C_-1 = 0
            aplică testul obținut
            la aceeaşi asignare pentru operanzii A şi B, asignează C_-1 = 1
            aplică testul obținut
        end
    For i = 0,n-1
            asignează A_i = 1, B_i = 1 şi C_-1 = 1
            aplică testul obținut
    C_-1 = 0
    Repeat
        i=0
        asignează A_i = 1, B_i = 0, A_i+1 = 0, B_i+1 = 0
        i=i+2
    until i ≥ n
    aplică testul obținut
    C_-1 = 1
    Repeat
        i=0
        asignează A_i = 0, B_i = 0, A_i+1 = 1, B_i+1 = 1
        i=i+2
    until i ≥ n
    aplică testul obținut
    k = 4
    Repeat
    Pentru fiecare bloc de k ranguri asignează tuturor rangurilor i ale semiblocului m.p.s.
              valorile A_i = 0, B_i = 1, iar pentru rangurile j ale semiblocul m.s. valorile A_j = 0,
              B_j = 0
                aplică testul obținut împreună cu C_-1 = 0
                Inversează valorile între semiblocuri
                aplică testul obținut împreună cu C_-1 = 0
    Pentru fiecare bloc de k ranguri asignează tuturor rangurilor i ale semiblocului m.s. valorile
              A_i = 1, B_i = 1, iar pentru rangurile j ale semiblocul m.p.s. valorile A_j = 0, B_j = 1
```


BUPT

aplică testul obținut împreună cu $C_{-1} = 1$
Inversează valorile între semiblocuri
aplică testul obținut împreună cu $C_{-1} = 0$

Pentru fiecare bloc de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.s. valorile* $A_i = 1$, $B_i = 1$, *iar pentru rangurile* j *ale semiblocul m.p.s. valorile* $A_j = 0$, $B_j = 0$
aplică testul obținut împreună cu $C_{-1} = 1$
Inversează valorile între semiblocuri
aplică testul obținut împreună cu $C_{-1} = 0$

Pentru fiecare bloc par de k *ranguri asignează tuturor rangurilor* i *valorile* $A_i = 0$, $B_i = 0$, *iar pentru rangurile* j *ale blocurilor impare de* k *ranguri valorile* $A_j = 1$, $B_j = 1$
aplică testul obținut împreună cu $C_{-1} = 0$
Inversează valorile blocurilor pare cu cele ale blocurilor impare
aplică testul obținut împreună cu $C_{-1} = 1$

Pentru fiecare bloc impar de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.s. valorile* $A_i = 1$, $B_i = 1$, *iar pentru rangurile* j *ale semiblocul m.p.s. valorile* $A_j = 0$, $B_j = 0$

Pentru fiecare bloc par de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.p.s. valorile* $A_i = 1$, $B_i = 1$, *iar pentru rangurile* j *ale semiblocul m.s. valorile* $A_j = 0$, $B_j = 0$
aplică testul obținut împreună cu $C_{-1} = 0$
Inversează valorile blocurilor pare cu cele ale blocurilor impare
aplică testul obținut împreună cu $C_{-1} = 1$

Pentru fiecare bloc impar de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.s. valorile* $A_i = 1$, $B_i = 1$, *iar pentru rangurile* j *ale semiblocul m.p.s. valorile* $A_j = 0$, $B_j = 1$

Pentru fiecare bloc par de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.p.s. valorile* $A_i = 0$, $B_i = 1$, *iar pentru rangurile* j *ale semiblocul m.s. valorile* $A_j = 0$, $B_j = 0$
aplică testul obținut împreună cu $C_{-1} = 0$
Inversează valorile blocurilor pare cu cele ale blocurilor impare
aplică testul obținut împreună cu $C_{-1} = 1$

Pentru fiecare bloc impar de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.s. valorile* $A_i = 1$, $B_i = 1$, *iar pentru rangurile* j *ale semiblocul m.p.s. valorile* $A_j = 1$, $B_j = 1$

Pentru fiecare bloc par de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.p.s. valorile* $A_i = 0$, $B_i = 0$, *iar pentru rangurile* j *ale semiblocul m.s. valorile* $A_j = 0$, $B_j = 1$
aplică testul obținut împreună cu $C_{-1} = 0$
Inversează valorile blocurilor pare cu cele ale blocurilor impare
aplică testul obținut împreună cu $C_{-1} = 1$

dublează k

until $k > \frac{n}{2}$

Pentru fiecare bloc de k *ranguri asignează tuturor rangurilor* i *ale semiblocului m.p.s. valorile* $A_i = 0$, $B_i = 1$, *iar pentru rangurile* j *ale semiblocul m.s. valorile* $A_j = 0$, $B_j = 0$
aplică testul obținut împreună cu $C_{-1} = 0$
aplică testul obținut împreună cu $C_{-1} = 1$
Inversează valorile între semiblocurile unui bloc
aplică testul obținut împreună cu $C_{-1} = 0$

BUPT

aplică testul obținut împreună cu $C_{-1}=1$
Asignează rangurilor j ale semiblocul m.s. valorile $A_j=1$, $B_j=1$
aplică testul obținut împreună cu $C_{-1}=0$
aplică testul obținut împreună cu $C_{-1}=1$
Inversează valorile între semiblocurile unui bloc
aplică testul obținut împreună cu $C_{-1}=1$
aplică testul obținut împreună cu $C_{-1}=0$
Asignează rangurilor j ale semiblocul m.s. valorile $A_j=0$, $B_j=1$
aplică testul obținut împreună cu $C_{-1}=1$
aplică testul obținut împreună cu $C_{-1}=0$
Inversează valorile între semiblocurile unui bloc
aplică testul obținut împreună cu $C_{-1}=1$
aplică testul obținut împreună cu $C_{-1}=0$
Asignează la toate rangurile blocurilor valorile $A_j=1$, $B_j=1$
aplică testul obținut împreună cu $C_{-1}=0$
Asignează la toate rangurile blocurilor valorile $A_j=0$, $B_j=0$
aplică testul obținut împreună cu $C_{-1}=1$

end

Fig.A 6

Algoritm CSkA()
Begin
For $i=0,n-1$
asignează $A_i=0$, $B_i=0$, $Test=1$ *și* $C_{-1}=0$
aplică testul obținut
For $i=0,n-1$
asignează $A_i=0$, $B_i=1$, $Test=1$ *și* $C_{-1}=0$
aplică testul obținut
la aceeași asignare pentru operanzii A și B, asignează $C_{-1}=1$
aplică testul obținut
la aceeași asignare pentru operanzii A și B, precum și pentru C_{-1}, *asignează* $Test=0$
aplică testul obținut
end
For $i=0,n-1$
Begin
asignează $A_i=1$, $B_i=0$, $C_{-1}=0$ *și* $Test=1$
aplică testul obținut
la aceeași asignare pentru operanzii A și B, asignează $C_{-1}=1$
aplică testul obținut
la aceeași asignare pentru operanzii A și B, precum și pentru C_{-1}, *asignează* $Test=0$
aplică testul obținut
end
$Test=1$
For $i=0,n-1$
asignează $A_i=1$, $B_i=1$ *și* $C_{-1}=0$
aplică testul obținut
$C_{-1}=0$
Repeat
i=0

BUPT

asignează $A_i = 1$, $B_i = 1$, $A_{i+1} = 0, C_{-1} = 0$
i=i+2
until $i \geq n$
aplică testul obținut
$C_{-1} = 1$
Repeat
i=0
asignează $A_i = 0$, $B_i = 0$, $A_{i+1} = 1$, $B_{i+1} = 1$
i=i+2
until $i \geq n$
aplică testul obținut
$C_{-1} = 1$
/ pentru blocuri de dimensiune constantă m */*
For $j = 1, m$
Begin
Asignează ca intrări pentru rangul j al tuturor blocurilor, valorile: $A_j = 0$, $B_j = 0$
Pentru toate blocurile sumatorului ***Do***
For $(i = 1, m), i \neq j$ *asignează pentru rangul i valorile* $A_i = 0$, $B_i = 1$
Asignează pentru ultimul rang al blocurilor pare valorile: $A_m = 1$, $B_m = 1$
aplică testul obținut
Inversează valorile de intrare ale ultimului rang al blocurilor pare cu al celor impare
aplică testul obținut
end
end

Fig.A 7

Algoritm CSkA_pe_2_nivele()
Begin
For $i = 0, n-1$
asignează $A_i = 0$, $B_i = 0$, $Test = 1$ *și* $C_{-1} = 0$
aplică testul obținut
For $i = 0, n-1$
asignează $A_i = 0$, $B_i = 1$, $Test = 1$ *și* $C_{-1} = 0$
aplică testul obținut
la aceeași asignare pentru operanzii A și B , asignează $C_{-1} = 1$
aplică testul obținut
la aceeași asignare pentru operanzii A și B, precum și pentru C_{-1}, *asignează* $Test = 0$
aplică testul obținut
end
For $i = 0, n-1$
Begin
asignează $A_i = 1$, $B_i = 0$, $C_{-1} = 0$ *și* $Test = 1$
aplică testul obținut
la aceeași asignare pentru operanzii A și B , asignează $C_{-1} = 1$
aplică testul obținut
la aceeași asignare pentru operanzii A și B, precum și pentru C_{-1}, *asignează* $Test = 0$
aplică testul obținut
end
$Test = 1$
For $i = 0, n-1$
asignează $A_i = 1$, $B_i = 1$ *și* $C_{-1} = 0$
aplică testul obținut

BUPT

```
C_-1 = 0
Repeat
    i=0
    asignează A_i = 1, B_i = 1 , A_i+1 = 0, C_-1 = 0
    i=i+2
until i ≥ n
aplică testul obținut
C_-1 = 1
Repeat
    i=0
    asignează A_i = 0, B_i = 0, A_i+1 = 1, B_i+1 = 1
    i=i+2
until i ≥ n
aplică testul obținut
    C_-1 = 1
 Test = 0
Asignează pentru primul bloc al fiecărui super-bloc
    For i = 1,m   valorle A_i = 0 și B_i = 0
Pentru toate superblocurile
    Pentru toate blocurile j, j = 2,M
        For i = 1,m asignează valorile A_i = 1 și B_i = 1
aplică testul obținut
 Test = 1
/*pentru blocuri de dimensiune constantă m *
For j = 1,m
    Begin
        Asignează ca intrări pentru rangul j al tuturor blocurilor valorile A_j = 0, B_j = 0
        Pentru toate blocurile sumatorului Do
                For (i = 1,m),i ≠ j  asignează pentru rangul i valorile A_i = 0, B_i = 1
        Asignează pentru ultimul rang al blocurilor pare valorile: A_m = 1, B_m = 1
        aplică testul obținut
        Inversează valorile de intrare ale ultimului rang al blocurilor pare cu al celor impare
        aplică testul obținut
    end
/*pentru super-blocuri ce au M blocuri (M constant) *
For j = 1,M
    Begin
        C_-1 = 1
        Pentru blocul j al tuturor superblocurilor
            For i = (1,m) asignează valorile A_i = 0, B_i = 0
        Pentru toate superblocurile
            Pentru blocurilor k, k = 1,M  (k ≠ j)
                For i = (1,m) asignează valorile A_i = 0, B_i = 1
        Pentru ultimul bloc al superblocurilor pare
                For i = (1,m) asignează A_i = 1, B_i = 1
        aplică testul obținut
        Inversează valorile de intrare ale ultimului bloc al super-blocurilor pare cu al celor impare
        aplică testul obținut
    end
end
```

Fig.A 8

BUPT

Anexa 2

A2.1 Structura programului

Aplicaţia este structurată în următoarele părţi, care interacţionează între ele după cum este prezentat în figura de mai jos:

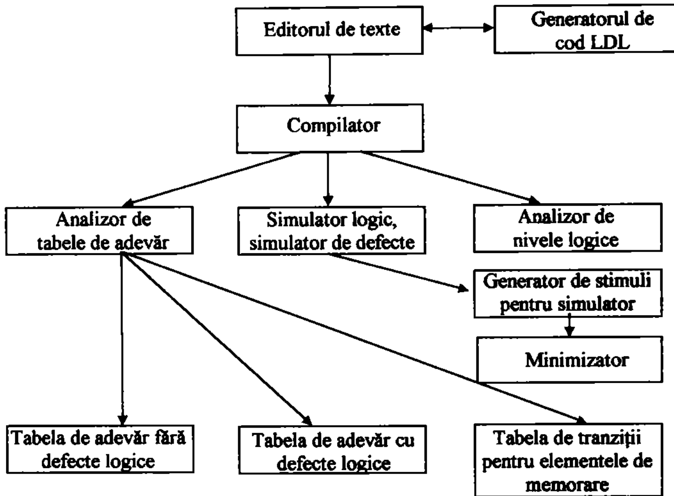


Fig.A2. 1

Funcţionalitatea părţilor componente ale programului este prezentată în subcapitolele următoare.

Aplicaţia a fost scrisă în limbajul *Visual C++* versiunea 4.2, cu ierarhia de clase MFC (*Microsoft Foundation Class*) şi rulează sub *Windows 95/NT*; pentru implementarea procedurilor s-au declarat următoarele clase:

Clase declarate	Explicaţii
CMainFrame	scheletul aplicaţiei
CLDLApp	aplicaţia propriu zisă
CLDLDoc	documentul aplicaţiei
CTxtView	editorul de text derivat din CEditView
CAbouLDLg	dialogul de informaţii despre aplicaţie
CBytes	tablou cu elemente de octet
CCalculate	evaluarea expresiilor
CChildFrame	scheletul pentru editorul de text
CCompiler	compilatorul LDL
CData	structura datelor din formatul intern care rezultă la compilare
CDrawView	fereastra de analizor de nivele
CEntryDlg	dialog la editarea de stimuli pentru simulator
CGotoLine	dialog pentru poziţionarea cursorului
CInfoDialog	dialog pentur informaţii generale despre descrierea schemei
CInputView	fereastră pentru listarea de stimuli la simulator
CImputWnd	fereastră schelet pentru CInputView

BUPT

CLevel	structura internă a datelor în analizorul de nivele
CLevelData	structura unei date în analizorul de nivele
CLevelDlg	dialog pentru afişarea dependenţelor dintre nivele de porţi
CLevelHead	structura care conţine lista de date pentru nivele
CLevelPage	pagina de opţiuni pentru analizatorul de nivele
CListCtrlEx	control pentru afişare într-o listă
CListCtrlExInfo	structură cu infomaţii pentru controlul de mai sus
CListViewEx	fereastră pentru afişare într-o listă
CListViewExInfo	structură cu informaţii pentru fereastra de mai sus
CMarkerDlg	dialog pentru poziţionarea timpului la simulator
CMDIDrawWnd	fereastra schelet pentru analizorul de nivele
CMDIOutWnd	fereastra schelet pentru afişarea mesajelor de compilator
CMDISimulatorWnd	fereastra schelet pentru simulator
CMDITabelWnd	fereastra schelet pentru analizorul de tabele de adevăr
CModule	structura internă a unui modul
CModulePage	fereastra pentru generatorul de sursă LDL
CModuleView	fereastra pentru informaţii cu semnalele şi modulele
CPictureButton	buton cu imagine
CPoage	structură internă a rezultatelor generală prin metoda Poage
CPrinterPage	pagină de opţiuni pentru setarea imprimantei
CResults	structura internă a rezultatelor după compilare
CSettingsPage	pagină de opţiuni pentru setări generale
CSignalDlg	dialog pentru introducerea informaţilor la generatorul de
CSimulatorPage	pagină de opţiuni pentru simulator
CSimulatorView	fereastra pentru simulator
CSortInfo	informaţii folosită de CListCtrlEx pentru sortarea în listă
CSoundPage	pagină de opţiuni pentru setarea sunetului
CSplashWnd	fereastra de intrare al aplicaţiei
CStimulData	structura datei la simulator
CStimulHead	structura listei cu date la simulator
CStimulusDlg	dialog pentru generarea de stimuli la simulator
CTabDlg	dialog pentru setarea tabulaţiei la editorul de text
CTabe	structura de informaţii despre un tabel de adevăr
CTabeView	fereastra pentru tabele de adevăr
Token	structura unui element lexical folosit de analizorul lexical

În continuare este dat programul aplicaţiei.

```
//TableView.h : header file
//
class CBytes : public CByteArray
{
public:
        CBytes() {};
        CBytes(CByteArray& byte)
        {
                SetSize(byte.GetSize());
                for(int i = 0; i < byte.GetSize(); i++)
                        SetAt(i, byte.GetAt(i));
        }
        CBytes &operator=(CBytes &arByte){
                SetSize(arByte.GetSize());
                for(int i = 0; i < arByte.GetSize();i++)
                        SetAt(i, arByte.GetAt(i));
                return (*this);
        };
};

class CTable : public CObject
{
public:
```


BUPT

```
            int GetHead(CString sId);
            BYTE GetTable(int x, int y);
            CString m_sName;
            int        m_iIn, m_iOut, m_iSig;
            int        m_iBytes;
            CStringList       *m_pHead;
            CArray<CBytes, CBytes&>                    m_arTable;
            void SetTable(int x, int y, BYTE value);
};

class CPoage : public CObject
{
public:
            CString GetType(CString sName);
            void SetExpr(CString sId, Cstring sExpr);
            CString GetExpr(CString sId);
            CString GetActive(CString sId);
            CString GetSentence(CString sId);
            CPoage() {};
            ~CPoage() {};
            CStringList               m_listType;
            CStringList               m_listName;
            CStringList               m_listActive;
            CStringList m_listExpr;
            CTable                    m_tablePoage;
};
/////////////////////////////////////////////////////////////////////////////
//CTableView view

class CTableView :  public CScrollView
{
protected:
            void CreatePoageModule(CTable* table, CModule* module);
            void CalculatePoageTable(CTable* table, CModule* module);
            void CalculateTable(CTable* table, Cmodule* module);
            CTableView();          // protected constructor used by dynamic creation
            DECLARE_DYNCREATE(CTableView)
// Attributes
public:
            CSize m_sizePoage;
            CSize m_sizeTable;
            int m_xexpr;
            int m_xtype;
            int m_xname;
            int m_iDisplayType;
            void TablePrint();
            void InitTable();
            CObList *m_pTable;
            Cpoage  *m_pPoage;
            int m_cy;
// Operations
public:
// Overrides
            //ClassWizard generated virtual function overrides
            //{{AFX_VIRTUAL(CTableView)
            protected:
            virtual void OnDraw(CDC* pDc);              // overriden to draw this view
            virtual void OnInitialUpdate();   // first time after construct
            virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
            virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
            virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
            //}}AFX_VIRTUAL
            afx_msg LRESULT  OnInitTable(UINT wParam, long lParam);
// Implementation
protected:
            int CalculateModule(int y, CData* data, CTable*, BOOL bPoage);
            virtual ~CTableView();
#ifdef_DEBUG
            virtual void AssertValid() const;
            virtual void Dump(CDumpContext& dc) const;
#endif
```


BUPT

```
		// Generated message map functions
		//{{AFX_MSG(CTableView)
			// NOTE - the ClassWizard will add and remove member functions here.
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
private:
		void DisplayFlow(CDC* pDC);
		int m_f;
		void DrawTable(CDC* pDC, CTable* table);
		void RemovePoage();
		void DisplayPoage(CDC* pDC);
		void DisplayTable(CDC* pDC);
		int m_dx;
		int m_dy;
		int m_py;
		int m_y;
		int m_x;
		void RemoveTable();
};
/////////////////////////////////////////////////////////////////////////////
// Calculate.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CCalculate class

class CCalculate : public CObject
{
//Construction
public:
		voidCompute();
		int GetResult();
		void SetExpression(CString expr);
		void SetModule(CModule* module);
		int GetReservedIndex(CString sId);
		CCalculate();
// Attributes
public:
// Operations
public:
//Overrides
		// ClassWizard generated virtual function overrides
		//{{AFX_VIRTUAL(CCalculate)
		//}}AFX_VIRTUAL

// Implementation
public:
		virtual ~CCalculate();

		// Generated message map functions
protected:
		//{{AFX_MSG(CCalculate)
			// NOTE - the ClassWizard will add and remove member functions here.
		//}}AFX_MSG
		//DECLARE_MESSAGE_MAP()
private:
		int GetTokenValue(CString sId);
		int CalcFact();
		int CalcTerm();
		int CalcExpr();
		CString GetToken();
		CString m_sToken;
		int m_iResult;
		CString m_sExpr;
		CModule* m_obModule;
		CStringList m_listReserved;
};
/////////////////////////////////////////////////////////////////////////////

// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////
class CChildFrame : public CMDIChildWnd
```


BUPT

```
{
        DECLARE_DYNCREATE(CChildFrame)
public:
        CChildFrame();
// Attributes
public:
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CChildFrame)
        public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~CChildFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
// Generated message map functions
protected:
        //{{AFX_MSG(CChildFrame)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
// Compiler.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CCompiler window

//#include "Results.h"
enum EClass
{
        TDL_Nothing = 0,
        TDL_Error,
        TDL_End,                //end of file
        TDL_Keyword,    //module, inputs, testing
        TDL_Id,                 //a, b, x1
        TDL_Number,                     //1, 2, 3, 4...
        TDL_Par1,               //(
        TDL_Par2,               //)
        TDL_Sep1,                       //,
        TDL_Sep2,                       //;
        TDL_Operator,   //=,=>
        TDL_Comment             //--comment until the end  of line
};
strukt Token
{
        EClass  Type;
        int             Value;
        CString Pid;
        Token &operator = (Token& a)
        {
                Type = a.Type;
                Value = a.Value;
                Pid = a.Pid;
                return (*this);
        }
};

class CResults;
class CCompiler : public CObject
{
// Construction
public:
        CResults  *m_pResults;
        int m_iLine;
        int m_iWarnings;
        int m_iErrors;
```


BUPT

```
            BOOL CanDraw();
            virtual void Analyze();
            void SetBoxList(CListBox* listBox);
            void SetStringList(CStringList* listString);
            CCompiler();
// Attributes
public:
// Operations
public:
// Overrides
            // ClassWizard generated virtual function overrides
            //{{AFX_VIRTUAL(CCompiler)
            //}}AFX_VIRTUAL
// Implementation
public:
            CString m_sFileName;
            virtual ~CCompiler();
            // Generated message map functions
protected:
            Token& NextModule();
            BOOL m_bMain;
            CString m_sExpr;
            Cmodule* m_testModule;
            BOOL m_bSem;
            Token& NextLine();
            Token m_cToken;
            BOOL m_bDraw
            CString m_token;
            int m_ipLine;
            char NextChar();
            Token& NextToken();
            char m_c;
            //{{AFX_MSG(CCompiler)
                  // NOTE - the ClassWizard will add and  remove member functions here.
            //}}AFX_MSG
            //DECLARE_MESSAGE_MAP()
private:
            void TDLFaultsData(CModule* module);
            int TDLFindFault();
            int TDLFindModule();
            int TDLFindName();
            void TDLModuleCheck(CModule *newModule, CModule *testModule);
            void TDLModuleExpression(CModule *newModule);
            void TDLModuleData(CModule *newModule, int iIndex);
            CString GetReservedKeyword(int nIndex);
            void TDLWarning(int iWarning, CString sMessage);
            void TDLFaults();
            void SemanticError();
            void SemanticFact();
            void SemanticTerm();
            void SemanticExpr();
            void TDLExpression();
            void TDLConections();
            void TDLModule();
            void InitResults();
            void DestroyResults();
            void TDLError(int iError, CString sMessage);
            void TDLName();
            BOOL IsLexical(EClass Type);
            void DoTDL();
            int GetReservedIndex(CString sAtom);
            CStringList *m_listReserved;
            CStringList* m_listString;
            CListBox* m_listBox;
};
/////////////////////////////////////////////////////////////////////////////
// Results.h : header file
//
/////////////////////////////////////////////////////////////////////////////
//CResults object

class CData : public CObject
```


BUPT

```
{
public:
        CData () {};
        ~CData () {};
        CString m_sId;
        int             m_iValue;
        CString m_sExpr;
        int m_iType;    //0=expr, 1=module
};

class CModule : public CObject
{
public:
        int GetOutputs();
        int GetSignals();
        int GetInputs();
        BOOL IsOutput(CString sId);
        BOOL IsSignal(CString sId);
        BOOL IsInput(CString sId);
        CData* GetData(CString sId);
        void AddSignal(CString sId, int iValue, CString sExpr);
        void AddOutput(CString sId, int iValue, CString sExpr);
        void AddInput(CString sId, int iValue, CString sExpr);
        void DestroyTDLModule();
        void InitTDLModule();
        CModule() {};
        ~CModule() {};
        CString m_sName;
        CObList *m_obInputs;
        CObList *m_obOutputs;
        CObList *m_obSignals;
};

class CResults : public CObject
{
// Construction
public:
        CModule* GetModule(CString sModule);
        int IsFault(CString sId);
        void AddFault(CString sId, int iValue);
        void DestroyTDLResults();
        void InitTDLResults();
        CString GetTDLName();
        void SetTDLName(CString name);
        CString m_TDLName;
        CObList *m_obModule;
        CObList *m_obFaults;
        CResults();
// Attributes
public:
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CResults)
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~CResults();
        // Generated message map functions
protected:
        //{{AFX_MSG(CResults)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        // DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////
//CLevelData object
class CLevelData : public CData
{
```


BUPT

```
public:
        void AddDependencies(CStringList *listDep);
        CLevelData();
        ~CLevelData();
        CStringList              *m_listDep;
};

class CLevelHead : public CObject
{
public:
        CObList *m_listData;
        int m_iLevel;
        CString m_sName;
        CLevelHead();
        ~CLevelHead();
};
////////////////////////////////////////////////
//CLevel object
class CLevel : public CObject
{
public:
        CLevelData* GetLevelData(int x, int y);
        int GetLevel(CString sId, int & iDeep, CLevelData* getData);
        int m_cy;
        int m_cx;
        BOOL IsEmpty();
        void DestroyLevel();
        void InitLevel();
        CLevel();  //public constructor
        ~Clevel(); //public destrcutor
        CObList *m_listLevel;
protected:
        void PutInLevel(CData* data);
};

// Draw View.h : header file
//
////////////////////////////////////////////////////
// CDrawView view

class CDrawView : public CScrollView
{
protected:
        CDrawView();          // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CDrawView)
// Attributes
public:
        void SchemePrint();
// Operations
public:
        double m_dZoomFactor;
        void SetArrowColor(COLORREF colorDep);
        COLORREF GetArrowColor();
        BOOL m_bShowDep;
        BOOL m_bGrid;
        void InitLevels();
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDraw View)
        protected:
        virtual void OnDraw(CDC* pDC);            // overridden to draw this view
        virtual void OnInitialUpdate();   // first time after construct
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL
// Implementation
protected
        void PrintLevels(CDC* pDC);
        void DrawArrow(CDC *pDC, int x, int y, int xd, int yd);
        void DrawDependencies(CDC* pDC);
        void DrawModule(CDC* pDC, int x, int y, CString sName, CString sExpr);
```


BUPT

```
        void DrawExpression(CDC* pDC, int x, int y, CString sName, CString Expr);  CPen m_penDep;
        CPen m_penDot;
        CPen m_penPrimary
        void DrawOutput(CDC* pDC, int x, int y, CString sName, CString sExpr);
        void DrawInput(CDC* pDC, int x, int y, CString sName);
        void DrawLevels(CDC *pDC, CObList *level);
        void DrawHeader(CDC *pDC);
        virtual ~CDrawView();
#ifdef_DEBUG
        virtual void Assert Valid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
        // Generated message map functions
        //{{AFX_MSG(CDraw View)
        afx_msg void OnLButtonDown(UINT nFLags, CPoint point);
        afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        BOOL m_bShowEnable;
        int m_iCursorType;
        COLORREF m_colorDep;
        CBitmap m_bmpDC;
        CDC m_mDC;
        CLevelData* m_dataLevel;
        HCURSOR m_cursorOld;
        HCURSOR m_cursorZoom;
        int m_grid;
        int m_dy;
        int m_hy;
        int m_dx;
        CLevel *m_obLevel;
};
//////////////////////////////////////////////////////
#if !defined(AFX_ENTRYDLG_H__26637EC2_BFCD_11D0_B539_000021847524
__INCLUDED_)
#define AFX_ENTRYDLG_H__26637EC2_BFCD_11D0_B539_000021847524__
INCLUDED_

#if_MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// EntryDlg.h : header fle
//
//////////////////////////////////////////////////
// CEntryDlg dialog

class CEntryDlg : public CDialog
{
// Construction
public:
        CEntryDLG(CWnd* pParent = NULL);  // standard constructor
// Dialog Data
        //{{AFX_DATA(CEntryDlg)
        enum { IDD = IDD_ENTRY };
        double m_dTime1;
        double m_dTime2;
        int                     m_iType;
        //}}AFX_DATA
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CEntryDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CEntryDlg)
                // NOTE: the ClassWizard will add member functions here
```


BUPT

```
	//}}AFX_MSG
	DECLARE_MESSAGE-MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.
#endif //
!defined(AFX_ENTRYDLG_H__26637EC2_BFCD_11D0_B539_000021847524__
INCLUDED_)

//GotoLine.h : header file
//
//////////////////////////////////////////////////////
// CGotoLine dialog

class CGotoLine : public CDialog
{
// Construction
public:
	CGotoLine(CWnd* pParent = NULL);  // standard constructor
// Dialog Data
	//{{AFX_DATA(CGotoLine)
	enum { IDD = IDD_GOTO };
	CString m_sLine
	//}}AFX_DATA
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CGotoLine)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CGotoLine)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

// InfoDlg.h : header file
//
//////////////////////////////////////////////////////
// CInfoDlg dialog

class CInfoDlg : public CDialog
{
// Construction
public:
	CInfoDlg(CWnd* pParent = NULL);  // standard constructor
// Dialog Data
	//{{AFX_DATA(CInfoDlg)
	enum { IDD = IDD_INFO };
	CString m_sErrors;
	CString m_sLines;
	CString m_sWarnings;
	CString m_s Modules;
	CString m_sName;
	CString m_sFileName;
	//}}AFX_DATA
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CInfoDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation
protected:
	// Generated message map functions
	//{{AFX_MSG(CInfoDlg)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
```


BUPT

```
	DECLARE_MESSAGE_MAP()
};

#if !defined(AFX_IMPUTVIEW_H__99B1F223_B588_11D0_B539_000021847524
__INCLUDED_)
#define AFX_INPUTVIEW_H__99B1F223_B588_11D0_B539_000021847524__
INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif// _MSC_VER >= 1000
// InputView.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CInputView view
class CListViewEx;

class CInputView : public CListViewEx
{
protected:
	CInputView();	// protected constructor used by dynamic creation
	DECLARE_DYNCREATE(CInputView)
// Attributes
public:
// Operations
public:
	CObList* m_listStimul;
// Overrides
	// ClassWizard generated virtual function overrides
	//{{FX_VIRTUAL(CInputView)
	public:
	virtual void OnInitialUpdate();
	protected:
	virtual void OnDraw(CDC* pDC);	// overriden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	//}}AFX_VIRTUAL
// Implementation
protected:
	virtual ~CInputView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	// Generated message map functions
protected:
	void EditInput(int nItem);
	//{{AFX_MSG(CInputView)
	afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
	afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
	afx_msg void OnInputList();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	CInputWnd* m_wndInput;
};
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
//Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_INPUTVIEW_H__99B1F223_B588_11D0_B539_000021847524__INCLUDED_)

#if !defined(AFX_INPUTWND_H__E5122EA2_BB56_11D0_B539_000021847524
__INCLUDED_)
#define AFX_INPUTWND_H__E5122EA2_BB56_11D0_B539_000021847524__
INCLUDED_

#include "ListCtrlEx.h"	// Added by ClassView
#if _MSC_VER >= 1000
#pragma once
```


BUPT

```
#endif // _MSC_VER >= 1000
// InputWnd.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CInputWnd window
class CStimulHead;

class CInputWnd : public CWnd
{
// Construction
public:
	CInputWnd();
// Attributes
public:
// Operations
public:
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CInputWnd)
	//}}AFX_VIRTUAL
// Implementation
public:
	void SetStimulHead(CStimulHead* headStimul);
	CListCtrlEx m_listInput;
	void SetDropButton(CButton* wndButton);
	virtual ~CInputWnd();
	// Generated message map functions
protected:
	//{{AFX_MSG(CInputWnd)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnPaint();
	//}}AFX_MSG
	afx_msg void OnListKillFocus(NMHDR * pNotifyStruct, LRESULT * result);
	afx_msg void OnListKeyDown(NMHDR * pNotifyStruct, LRESULT * result);
	afx_msg LRESULT OnListQuit(WPARAM wParam, LPARAM lParam);
	DECLARE_MESSAGE_MAP()
private:
	void FillInputList();
	CFont* m_fontList;
	CStimulHead* m_headStimul;
	CButton* m_btnDrop;
};
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the pevious line.

#endif //
!defined(AFX_INPUTWND_H__E5122EA2_BB56_11D0_B539_000021847524__
INCLUDED_)

// LevelDlg.h : header file
//
/////////////////////////////////////////////////////////////////////////////
//CLevelDlg dialog
class CLevel;
class CLevelData;

class CLevelDlg : public CDialog
{
// Construction
public:
	void SetLevel(CLevel* obLevel);
	void SetLevelData(CLevelData* dataLevel);
	CLevelDlg(CWnd* pParent = NULL);  // standard constructor
// Dialog Data
	//{{AFX_DATA(CLevelDlg)
	enum { IDD = IDD_LEVEL };
	CListCtrlEx	m_listDep;
	CString m_sExpr;
	CString m_sId;
	//}}AFX_DATA
```


BUPT

```
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CLevelDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        // Generated message map functions
        //{{AFX_MSG(CLevelDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        void FillLevelList();
        CLevel* m_obLevel;
        CLevelData* m_dataLevel;
};

#if
!defined(AFX_LEVELPAGE_H__456DD6A2_B503_11D0_B539_000021847524__
INCLUDED_)
#define AFX_LEVELPAGE_H__456DD6A2_B503_11D0_B539_000021847524__
INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// LevelPage.h : header file
//
/////////////////////////////////////////////////////
// CLevelPage dialog

class CLevelPage : public CPropertyPage
{
        DECLARE_DYNCREATE(CLevelPage)
// Construction
public:
        COLORREF m_colorDep;
        CLevelPage();
        ~CLevelPage();
// Dialog Data
        //{{AFX_DATA(CLevelPage)
        enum { IDD = IDD_LEVELPAGE };
        CStatic m_staticDep;
        BOOL m_bShowDep;
        //}}AFX_DATA
// Overrides
        // ClassWizard generate virtual function overrides
        //{{AFX_VIRTUAL(CLevelPage)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
        //AFX_VIRTUAL
// Implementation
protected:
        // Generated message map functions
        //{{AFX_MSG(CLevelPage)
        afx_msg void OnDepColor();
        virtual BOOL OnInitDialog();
        afx_msg void OnPaint();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        CRect m_rcDep;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_LEVELPAGE_H__456DD6A2_B503_11D0_B539_000021847524__
INCLUDED_)
```


BUPT

```
#ifndef__LISTCTRLEX_H__
#define__LISTCTRLEX_H__
// ListCtrlEx.h : header file
//

#define LN_NONE             0
#define LN_VERT             1
#define LN_HORZ             2
#define LN_BOTH             3

#include <afxdisp.h>

class CListCtrlExInfo : public CObject
{
public:
        CListCtrlExInfo() {};
        ~CListCtrlExInfo() {};
        int                 m_iColumn;
        UNIT     m_nID;
        CButton m_btnDrop;
};
//////////////////////////////////////////////////////////////
// CListCtrlEx control

class CListCtrlEx : public CListCtrl
{
// Construction
public:
        void SortColumn(int iColumn = 0, BOOL bDescending = FALSE, BOOL bNumber = FALSE);
        BOOL     GetFullSelect();
        void SetFullSelect(BOOL bFullSel);
        int AddItem(int nItem, int nSubItem, LPCTSTR strItem,
                int nMask=LVIF_TEXT, int nImageIndex= -1);
        int AddColumn(LPCTSTR strItem, int nItem,
                int nFmt=LVCFMT_LEFT,
        int nSubItem= -1,
        int nMask=LVCF_FMT|LVCF_WIDTH|LVCF_TEXT|LVCF_SUBITEM);
        CListCtrlEx();
// Attibutes
public:
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CListCtrlEx)
        protected:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void PreSubclassWindow();
        virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
        //}}AFX_VIRTUAL
// Implementation
public:
        static CString VariantToString(const COleVariant& var);
        BOOL GetAutoResize();
        void SetAutoResize(BOOL bAutoResize);
        COLORREF GetLineColor();
        void SetLineColor(COLORREF clrLine);
        CButton* GetDropDownButton(int iColumn);
        BOOL SetDropDownButton(int iColumn, UINT nID);
        int GetDropDownButtonWidth();
        void SetDropDownButtonWidth(int cxDrop);
        int GetLines();
        void SetLines(int iLineType);
        COLORREF GetRowBkColor();
        void SetRowBkColor(COLORREF clrBkgndSec, BOOL bEven = FALSE);
        virtual ~CListCtrlEx();
        BOOL m_bClientWidthSel;
        /// Generated message map functions
protected:
        void ShowDropDown();
        void HideDropDown();
```


BUPT

```
	COLORREF		m_clrLine;
	static int CALLBACK CompareFunc(LPARAM lParam1, LPARAM lParam2, LPARAM lParamSort)
	void RepaintSelectedItems();
	int m_cxStateImageOffset;
	static LPCTSTR MakeShortString(CDC* pDC, LPCTSTR lpszLong, int nColumnLen, int nOffset);
	int m_cxClient;
	COLORREF m_clrBkgnd;
	COLORREF m_clrTextBk;
	COLORREF m_clrText;
	COLORREF m_clrBkgndSec;
	BOOL  m_bEven;
	virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
	//{{AFX_MSG(CListCtrlEx)
	afx_msg void OnSize(UINT nType, int cx, int cy);
	afx_msg void OnKillFocus(CWnd* pNewWnd);
	afx_msg void OnSetFocus(CWnd* pOldWnd);
	afx_msg void OnEndtrack(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnBegintrack(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnTrack(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void On Paint();
	afx_msg void OnItemchanged(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnHScroll(UINT nSBCod, UINT nPos, CScrollBar* pScrollBar);
	afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
	//}}AFX_MSG
	afx_msg LRESULT OnAutoResize(WPARAM wParam, LPARAM lParam);
	DECLARE_MESSAGE_MAP()
private:
	void ResizeColumns(int cx);
	BOOL m_bAutoResize:
	CDC m_dcMem;
	CPen m_penLine;
	int m_xTrack;
	CBitmap m_bmpArrow;
	int m_cxDrop;
	int m_cy Track;
	int m_cxTrack;
	int m_iLineType;
	BOOL m_bFullSel;
	CObList m_listBtn;
};
/////////////////////////////////////////////////////////////////////////////

#endif

// ListVwEx.h : interface of the CListViewEx class
//
// This class provedes a full row selection mode for the report
// mode list view control.
//
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992-1997 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and related
// electronic documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

class CListViewExInfo : public CObject
{
public:
	CListViewExInfo() {};
	~CListViewExInfo() {};
	int				m_iColumn;
	UINT   m_nID;
	CButton m_btnDrop;
};

class CListViewEx : public CListView
{
	DECLARE_DYNCREATE(CListViewEx)
```


BUPT

```
// Construction
public:
        CListViewEx();
// Attributes
protected:
        BOOL m_bFullRowSel;
public:
        BOOL SetFullRowSel(BOOL bFillRowSel);
        BOOL GetFullRowSel();

        BOOL m_bClientWidthSel;
// Overrides
protected:
        virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
        // ClassWizard generated virtual function ovverides
        //{{AFX_VIRTUAL(CListViewEx)
        public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL
// Implementation
public:
        void HideDropDown();
        BOOL GetVertLines();
        void SetVertLines(BOOL bVertLines);
        CButton* GetDropDownButton(int iColumn);
        void SetDropDownButton(int iColumn, UINT nID);
        BOOL GetAutoResize();
        void SetAutoResize(BOOL bAutoResize);
        int AddItem (int nItem, int nSubItem,
                LPCTSTR strItem,
                int nMask=LVIF_TEXT, int nImageIndex= -1);
        int AddColumn(LPCTSTR strItem, int nItem,
                int nFmt=LVCFMT_LEFT, int nSubItem= -1,
                int nMask=LVCF_FMT|LVCF_WIDTH|LVCF_TEXT|LVCF
        _SUBITEM);
        virtual ~CListViewEx();
#ifdef _DEBUG
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:
        static LPCTSTR MakeShortString(CDC* pDC, LPCTSTR lpszLong, int nColumnLen, int nOffset);
        void RepaintSelectedItems();
// Implementation - client area width
        int m_cxClient;
// Implementation - state icon width
        int m_cxStateImageOffset;
        afx_msg LRESULT OnSetImageList(WPARAM wParam, LPARAM lParam);
// Implementation - list view colors
        COLORREF m_clrText;
        COLORREF m_clrTextBk;
        COLORREF m_clrBkgnd;
        afx_msg LRESULT OnSetTextColor(WPARAM wParam, LPARAM lParam);
        afx_msg LRESULT OnSetTextBkColor(WPARAM wParam, LPARAM lParam);
        afx_msg LRESULT OnSEtBkColor(WPARAM wParam, LPARAM lParam);
// Generated message map functions
protected:
        void RepaintEmptyArea();
        int m_cxTrack;
        void ResizeColumns(int cx);
        BOOL m_bAutoResize;
        //{{AFX_MSG(CListViewEx)
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg void OnPaint();
        afx_msg void On SetFocus(CWnd* pOldWnd);
        afx_msg void OnKillFocus(CWnd* pNewWnd);
        afx_msg void On Endtrack (NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnBegintrack(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnTrack(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnItemchanged(NMHDR* pNMHDR, LRESULT pResult);
        afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
        //}}AFX_MSG
```


BUPT

```
		afx_msg LRESULT    OnAutoResize(WPARAM wParam, LPARAM lParam);
		afx_msg LRESULT    OnEmptyArea(WPARAM wParam, LPARAM lParam);
		DECLARE_MESSAGE_MAP()
private:
		BOOL m_bVertLines;
		void ShowDropDown();
		CBitmap m_bmpArrow;
		CObList m_listBtn;
};
/////////////////////////////////////////////////

// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////

class CMDIDrawWnd;
class CMDITableWnd;
class CMDIOutWnd;
class CMDISimulatorWnd;

class CMainFrame : public CMDIFrameWnd
{
		DECLARE_DYNAMIC(CMainFrame)
public:
		void SetStatusBarMessage(LPCTSTR sMsg, LPCTSTR sParam);
		void SetProgressPost(int pos);
		void SetProgressRange(int min, int max);
		void ShowProgressCtrl(int code);
		CProgressCtrl m_progressCtrl;
		virtual void OnDisplayOut();
		CMainFrame();
// Atributes
public:
		CMDISimulatorWnd *m_wndSimulator;
		CMDIDrawWnd* m_wndDraw;
		CMDIOutWnd    *m_wndOut;
		CMDITableWnd* m_wndTable;
		CToolBar    m_wndToolBar;
		CToolBar            m_wndTdlTool;
// Operations
public:
// Overrides
		// ClassWizard generated virtual function overrides
		//{{AFX_VIRTUAL(CMainFrame)
		public:
		virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
		//}}AFX_VIRTUAL
// Implementation
public:
		void SetLineText(CString sText);
		virtual ~CMainFrame();
#ifdef_DEBUG
		virtual void AssertValid() const;
		virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
		afx_msg void OnInitMenu(CMenu* pMenu);
		CStatusBar m_wndStatusBar;
// Generated message map functions
protected:
		//{{AFX_MSG(CMainFrame)
		afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
		afx_msg void OnViewTools();
		afx_msg void OnUpdateToolsMenu(CCmdUI* pCmdUI);
		afx_msg void OnTdlDraw();
		afx_msg void OnViewOutput();
		afx_msg void OnUpdateOutputMenu(CCmdUI* pCmdUI);
		afx_msg void OnTdlInfo();
		afx_msg void OnTdlTable();
		afx_msg void OnToolsOptions();
		afx_msg void OnViewTable();
```


BUPT

```
		afx_msg void OnUpdateViewTableMenu(CCmdUI* pCmdUI);
		afx_msg void OnTdlSimulator();
		afx_msg void OnViewDraw();
		afx_msg void OnUpdateViewDraw(CCmdUI* pCmdUI);
		afx_msg void OnShowOutput();
		afx_msg void OnViewSimulator();
		afx_msg void OnUpdateViewSimulator(CCmdUI* pCmdUI);
		afx_msg void OnSize(UINT nType, int cx, int cy);
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////

#if
!defined(AFX_MARKERDLG_H__CF10D1A2_C1E2_11D0_B539_000021847524
__INCLUDED_)
#define AFX_MARKERDLG_H__CF10D1A2_C1E2_11D0_B539_000021847524__
INCLUDED_

#if_MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MarkerDlg.h : header file
//
/////////////////////////////////////////////////////
// CMakerDlg dialog

class CMarkerDlg : public CDialog
{
// Construction
public:
		CMarkerDlg(CWnd* pParent = NULL); // standard constructor
// Dialog Data
		//{{AFX_DATA(CMarkerDlg)
		enum { IDD = IDD_MARKER };
		double m_dMarker;
		//}}AFX_DATA
// Overrides
		// ClassWizard generated virtual function overrides
		//{{AFX_VIRTUAL(CMarkerDlg)
		protected:
		virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
		//}}AFX_VIRTUAL

// Implementation
protected:

		// Generated message map functions
		//{{AFX_MSG(CMarkerDlg)
			// NOTE: the ClassWizard will add member functions here
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_MARKERDLG_H__CF10D1A2_C1E2_11D0_B539_000021847524__INCLUDED_)

// MDIDraw Wnd.h : header file
//
/////////////////////////////////////////////////////
// CMDIDraw Wnd frame
class CDrawView;

class CMDIDrawWnd : public CMDIChildWnd
{
		DECLARE_DYNCREATE(CMDIDrawWnd)
protected:

// Attributes
```


BUPT

```
public:
        CDrawView* m_viewDraw;
        CMDIDrawWnd();    // public constructor used by dynamic creation
        virtual ~CMDIDrawWnd();
// Operations
public:
        void InitDraw();
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMDIDrawWnd)
        public:
        virtual BOOL PreTranslateMessage(MSG* pMsg);
        //}}AFX_VIRTUAL
// Implementation
protected:
        // Generated message map functions
        //AFX_MSG(CMDIDrawWnd)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnClose();
        //}}AFX_MSG
        afx_msg void OnBack();
        afx_msg void OnSchemePrint();
        afx_msg void OnDrawOptions();
        afx_msg void OnChangeZoom();
        DECLARE_MESSAGE_MAP()
private:
        CDialogBar m_wndSelect;
};
/////////////////////////////////////////////////////////////

// MDIOutWnd.h : header file
//
/////////////////////////////////////////////////////////////
// CMDIOutWnd frame

//#include "Compiler.h"
class CCompiler;
class CTxtView;
class CMDIOutWnd : public CMDIChildWnd
{
        DECLARE_DYNCREATE(CMDIOutWnd)
public:
        void GetActiveMDI();
        CCompiler* m_Compiler;
        CStringList* m_sList;
        CListBox* m_listBox;
        virtual void Analyse();
        CMDIOutWnd();    // public constructor used by dinamic creation
        virtual ~CMDIOutWnd(); // public destructor
// Attributes
public:
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMDIOutWnd)
        public:
        virtual BOOL PreTranslateMessage(MSG* pMSG);
        //}}AFX_VIRTUAL
// Implementation
protected:
        void ShowOutputMenu();
        CChildFrame* m_mdiActive;
        // Generated message functions
        //{{AFX_MSG(CMDIOutWnd)
        afx_msg int On Create(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnDestroy();
        afx_msg void OnClose();
        afx_msg void OnListClick();
        afx_msg void OnOutputHide();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
```


BUPT

```
};
/////////////////////////////////////////////////////////

//CStimulHead object

class CStimulHead : public CObject
{
public:
        virtual void Serialize(CArhive& ar);
        int GetValue(double dTime);
        void DestroyData();
        Coblist* m_listData:
        BOOL m_bDefined;
        CString m_sId;
        CStimulHead();
        ~CStimulHead();
};
//CStimulData object
class CStimulData : public CObject
{
public:
        virtual void Serialize(CArhive & ar);
        int m_iValue;
        double m_dTime;
        CStimulData();
        ~CStimulData();
};
// MDISimulatorWnd.h : header file
//
/////////////////////////////////////////////////////////
// CMDISimulatorWnd frame
class CMDISimulator Wnd : public CMDIChildWnd
{
        DECLARE_DYNCREATE(CMDISimulatorWnd)
public:
        void AddEntry(CStimulHead* entry, CStimulHead* head);
        CImageList m_imageList;
        void InitSimulator();
        CSplitterWnd m_Splitter;
        CMDISimulatorWnd();            // public constructor used by dynamic creation
        virtual ~CMDISimulatorWnd();
// Attributes
public:
// Operations
public:
        Coblist* m_listFault;
        double m_dTime;
        Coblist* m_listStimul;
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMDISimulatorWnd)
        public:
        virtual BOOL PreTranslateMessage(MSG* pMsg);
        protected:
        virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
        //}}AFX_VIRTUAL
// Implementation
protected:
        // Generated message  map functions
        //{{AFX_MSG(CMDISimulator Wnd)
        afx_msg void OnClose();
        afx-msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        //}}AFX_MSG
        afx_msg void OnBack();
        afx_msg void OnPrintSimulator();
        afx_msg void OnRunSimulator();
        afx_msg void OnSimulatorOptions();
        afx_msg void OnSimulatorLoad();
        afx_msg void OnSimulatorSave();
        DECLARE_MESSAGE_MAP()
private:
        void CalculateFaults(CStimulHead* entry, CStimulHead* head, CTable* table, int iln);
```


BUPT

```
            void Destroy Fault();
            void CalculateOutputs(CStimulHead *entry, CStimulHead *head, CTable *table, int iIn);
            CBitmap m_bmpSave;
            CBitmap m_bmpLoad;
            void Destroy Stimul();
            void InitStimul();
            void FillInputList();
            CDialogBar m_wndSelect;
};
/////////////////////////////////////////////////////////////////////////////

// MDITableWnd.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CMDITableWnd frame

class CData;
class CResults;
class CMDITableWnd : public CMDIChildWnd
{
            DECLARE_DYNCREATE(CMDITableWnd)
public:
            CImageList m_imageList;
            void InitModule();
            CsplitterWnd m_Splitter,
            CMDITableWnd();        // public constructor used by dynamic creation
// Attributes
public:
// Operations
public:
// Overrides
            // ClassVizard generated virtual function overrides
            //{{AFX_VIRTUAL(CMDITableWnd)
            public:
            virtual BOOL PreTranslateMessage(MSG* pMsg);
            protected:
            virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
            //}}AFX_VIRTUAL
// Implementation
protected:
            virtual ~CMDITableWnd();
            // Generated message map functions
            //{{AFX_MSG(CMDITableWnd)
            afx_msg void OnClose();
            afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
            afx_msg void OnTableExit();
            //}}AFX_MSG
            afx_msg void OnTablePrint();
            afx_msg void OnTablePoage();
            afx_msg void OnTableShow();
            afx_msg void OnTableFlow();
            afx_msg LRESULT OnInitModule(UINT wParam, long lParam);
            DECLARE_MESSAGE_MAP()
private:
            CString FormatDeclaration(CData *data);
            CResults* m_pResults;
            CDialogBar m_wndSelect;
};
/////////////////////////////////////////////////////////////////////////////

// ModulePage.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CModulePage dialog

class CModulePage : public CPropertyPage
{
            DECLARE_DYNCREATE(CModulePage)
// Construction
public:
            CModulePage();
            ~CModulePage();
```


BUPT

```
// Dialog Data
	//{{AFX_DATA(CModulePage)
	enum {IDD = IDD_WIZARD };
	CButton m_buttonUpper;
	CListCtrlEx	m_listSig;
	CEdit	m_editName;
	//}}AFX_DATA
// Overrides
	// ClassWizard generate virtual function overrides
	//{{AFX_VIRTUAL(CModulePage)
	public:
	virtual BOOL OnSetActive();
	virtual BOOL PreTranslateMessage(MSG* pMsg);
	virtual BOOL OnVizardFinish();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation
protected:
	void InsertTDLSource();
	void OnDelete();
	CImageList m_imageSig;
	// Generated message map functions
	//{{AFX_MSG(CModulePage)
	virtual BOOL OnInitDialog();
	afx_msg void OnInsert();
	afx_msg void OnColumnclickListSig(NMHDR* pNMHDR, LRESULT* pResult);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
// ModuleView.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CModuleView view

class CModuleView : public CTreeView
{
protected:
	CModuleView();		// protected constructor used by dynamic creation
	DECLARE_DYNCREATE(CModuleView)
// Attributes
public:
// Operations
public:
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CModuleView)
	protected:
	virtual void OnDraw(CDC* pDC);		// overriden to draw this view
	//}}AFX_VIRTUAL
// Implementation
protected:
	virtual ~CModuleView();
#ifdef_DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
	// Generated message map functions
protected:
	//{{AFX_MSG(CModuleView)
	afx_msg void OnSelchanged(NMHDR* pNMHDR, LRESULT* pResult);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////

// PictureButton.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CPictureButton window
#ifndef PICTB
#define PICTB
```


BUPT

```
class CPictureButton : public CButton
{
// Construction
public:
		void SetPicture(UINT iResource);
		CPictureButton();
// Attributes
public:
// Operations
public:
// Overrides
		// ClassWizard generated virtual function overrides
		//{{AFX_VIRTUAL(CPictureButton)
		protected:
		virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
		//}}AFX_VIRTUAL
// Implementation
public:
		virtual ~CPictureButton();
		// Generated message map functions
protected:
		CImageList m_imageList;
		COLORREF m_colorButton;
		CBbitmap m_bPicture;
		CBitmap m_bMask;
		//{{AFX_MSG(CPictureButton)
		afx_msg void OnSysColorChange();
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
};
//////////////////////////////////////////////////
#endif
// TableView.h : header file
//
class CBytes : public CByteArray
{
public:
		CBytes() {};
		CBytes(CByteArray& byte)
{
		SetSize(byte.GetSize());
		for(int i = 0; i < byte.GetSize(); i++
				SetAt(i, byte.GetAt(i));
}
CBytes &operator=(CBytes &arByte){
		SetSize(arByte.GetSize());
		for(int i = 0; i < arByte.GetSize(); i++)
						SetAt(i, arByte.GetAt(i));
				return (*this);
		};
};

class Ctable : public CObject
{
public:
		int GetHead(CString sId);
		BYTE GetTable(int x, int y);
		Cstring m_sName;
		int	m_iIn, m_iOut, m_iSig;
		int	m_iBytes;
		CStringList	*m_pHead;
		CArray<CBytes, Cbytes&>	m_arTable;
		void SetTable(int x, int y, BYTE value);
};
class CPoage : public CObject
{
public:
		Cstring GetType(CString sName);
		void SetExpr(CString sId, Cstring sExpr);
		Cstring GetExpr(CString sId);
		Cstring GetActive(CString sId);
		Cstring GetSentence(CString sId);
```


BUPT

```
        CPoage() {};
        ~Cpoage() {};
        CStringList     m_listType;
        CStringList             m_listName;
        CStringList             m_listActive;
        CStringList m_listExpr;
        CTable                  m_tablePoage;
};
////////////////////////////////////////////////////
// CTableView view

class CTableView : public CScrollView
{
protected:
        void CreatePoageModule(CTable* table, CModule* module);
        void CalculatePoageTable(CTable* table, CModule* module);
        void CalculateTable(CTable* table, CModule* module);
        CTableView();       // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CTableView)
// Attributes
public:
        CSize m_sizePoage;
        CSize m_sizeTable;
        int m_xexpr;
        int m_xtype;
        int m_xname;
        int m_iDisplayType;
        void TablePrint();
        void InitTable();
        CObList* m_pTable;
        CPoage *m_pPoage;
        int m_cy;
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTableView)
        protected:
        virtual void OnDraw(CDC* pDC);      // overriden to draw this view
        virtual void OnInitialUpdate();     // first time after construct
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL
        afx_msg LRESULT OnInitTable(UINT wParam, long lParam);
// Implementation
protected:
        int CalculateModule(int y, CData* data, CTable* table, BOOL bPoage);
        virtual ~CTableView();
#ifdef_DEBUG
        virtual void Assert Valid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
        // Generated message map functions
        //{{AFX_MSG(CTableView)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE-MESSAGE-MAP()
private:
        void DisplayFlow(CDC* pDC);
        int m_f;
        void DrawTable(CDC* pDC, CTtable* table);
        void RemovePoage();
        void DisplayPoage(CDC* pDC);
        void DisplayTable(CDC* pDC);
        int m_dx;
        int m_dy;
        int m_py;
        int m_y;
        int m_x;
        void RemoveTable();
};
```


BUPT

```
///////////////////////////////////////////////////

// PrinterPage.h: header file
//
///////////////////////////////////////////////////
// CPrinterPage dialog

class CPrinterPage : public CPropertyPage
{
	DECLARE_DYNCREATE(CPrinterPage)
// Construction
public:
	int m_iAlign;
	CPrinterPage();
	~CPrinterPage();
// Dialog Data
	//{{AFX_DATA(CPrinterPage)
	enum { IDD = IDD_PRINTERPAGE };
	CSpinButtonCtrl m_spinSus;
	CSpinButtonCtrl m_spinStanga;
	CSpinButtonCtrl m_spinJos;
	CSpinButtonCtrl m_spinDreapta;
	CStatic m_staticRect;
	CButton  m_checkTabel;
	CButton m_checkPageNr;
	CComboBox   m_comboAlign;
	BOOL   m_bPageNr;
	BOOL   m_bGrid;
	UINT         m_uiDreapta;
	UINT    m_uiJos;
	UINT    m_uiStanga;
	UINT    m_ uiSus;
	int                    m_iOrientare;
	//}}AFX_DATA
// Overrides
	// ClassWizard generate virtual function overrides
	//{{AFX_VIRTUAL(CPrinterPage)
	public:
	virtual BOOL OnApply();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation
protected:
	// Generated message map functions
	//{{AFX_MSG(CPrinterPage)
	afx_msg void OnPaint();
	virtual BOOL OnInitDialog();
	afx_msg void OnCheckPaginare();
	afx_msg void OnCheckTabel();
	afx_msg void OnSelchangeComboAlign();
	afx_msg void OnDeltaposSpinDreapta(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnDeltaposSpinJos(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnDeltaposSpinStanga(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnDeltaposSpinSus(NMHDR* pNMHDR, LRESULT* pResult);
	afx_msg void OnRadioHorz();
	afx_msg void OnRadioVert();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	CRect m_rc;
	CPen m_penText;
};

// SignalDlg.h : header file
//
///////////////////////////////////////////////
// CSignalDlg dialog

class CSignalDlg : public CDialog
{
// Construction
```


BUPT

```
public:
        int m_iType;
        CSignalDlg(CWnd* pParent = NULL);  // standard constructor
// Dialog Data
        //{{AFX_DATA(CSignalDlg)
        enum { IDD = IDD_ADDSIG };
        CEdit   m_editDeclaration;
        CStatic m_staticDeclaration;
        CTabCtrl  m_tabType;
        CString m_sName;
        CString m_sDeclaration;
        //}}AFX_DATA
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSignalDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        CImageList m_imageSig;
        // Generated message map functions
        //{{AFX_MSG(CSignalDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSelchangeTabType(NMHDR* pNMHDR, LRESULT* pResult);
        //}}AFX_MSG DECLARE_MESSAGE_MAP()
};

#if
!defined(AFX_SIMULATORPAGE_H__9661E722_C199_11D0_B539_000021847524__INCLUDED_)
#define
AFX_SIMULATORPAGE_H__9661E722_C199_11D0_B539_000021847524__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// SimulatorPage.h : header file
//
////////////////////////////////////////////////////
// CSimulatorPage dialog

class CSimulatorPage : public CPropertyPage
{
        DECLARE_DYNCREATE(CSimulatorPage)
// Construction
public:
        COLORREF m_colorFault;
        COLORREF m_colorSignal;
        CSimulatorPage();
        ~CSimulatorPage();
// Dialog Data
        //{{AFX_DATA(CSimulatorPage)
        enum {IDD = IDD_SIMPAGE };
        CStatic m_staticSignal;
        CStatic m_staticFault;
        BOOL   m_bFault;
        BOOL   m_bEntry;
        //}}AFX_DATA
// Overrides
        // ClassWizard          generate virtual function overrides
        //{{AFX_VIRTUAL(CSimulatorPage)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        // Generated message map functions
        //{{AFX_MSG(CSimulatorPage)
        virtual BOOL OnInitDialog();
        afx_msg void On Paint();
        afx_msg void OnButtonFault();
        afx_msg void OnButtonSignal();
        //}}AFX_MSG
```


BUPT

```
	DECLARE_MESSAGE_MAP()
private:
	CRect m_rcFault;
	CRect m_rcSignal;
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_SIMULATORPAGE_H__9661E722_C199_11D0_D539_000021847524__INCLUDED_)

// SimulatorView.h : header file
//
/////////////////////////////////////////////////////
// CSimulatorView view

class CSimulatorView : public CScrollView
{
protected:
	CSimulatorView();	// protected constructor used by dynamic creation
	DECLARE_DYNCREATE(CSimulatorView)
// Attributes
public:
	void SimulatorPrint();
	CObList *m_listStimul;
	CPen m_penSignal;
// Operation
public:
	void SetMarker();
	BOOL m_bShowTime;
	CObList* m_listFault;
	COLORREF GetFaultColor();
	COLORREF GetSignalColor();
	void SetFaultColor(COLORREF colorFault);
	void SetSignalColor(COLORREF ColorSignal);
	CPen m_penMarker;
	CPen m_penFault;
	BOOL m_bShowFault;
	double m_dTime;
	void ScaleScrollSize();
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSimulatorView)
	protected:
	virtual void OnDraw(CDC* pDC);  // overriden to draw this view
	virtual void OnInitialUpdate();  // first time after construct
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	//}}AFX_VIRTUAL
// Implementation
protected:
	virtual ~CSimulatorView();
#ifdef_DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
	// Generated message map functions
	//{{AFX_MSG(CSimulatorView)
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
	afx_msg void OnMouseMove(UINT nFlags, CPoint point);
	afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
	afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
	afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	BOOL m_bDrag;
	int m_iCursorType;
	void DrawMarker(CDC* pDC);
	COLORREF m_colorMarker;
```


BUPT

```
		double m_dMarker;
		void DrawTime(CDC* pDC);
		void DrawFault(CDC* pDC);
		COLORREF m_colorFault;
		int m_sx;
		void DrawSignal(CDC *pDC);
		COLORREF m_colorSignal;
		int m_oy;
		int m_ox;
		int m_dy;
		void DrawSupport(CDC *pDC);
		int m_hx;
		int m_dx;
};
/////////////////////////////////////////////////////////////////////////////

// SoundPage.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CSoundPage dialog

class CPictureButton;
class CSoundPage : public CPropertyPage
{
		DECLARE_DYNCREATE(CSoundPage)
// Construction
public:
		CSoundPage();
		~CSoundPage();
// Dialog Data
		//{{AFX_DATA(CSoundPage)
		enum { IDD = IDD_SOUND };
		CListBox	m_listEvents;
		CPictureButton m_buttonStop;
		CPictureButton m_buttonPlay;
		//}}AFX_DATA
// Overrides
		// ClassWizard generate virtual function overrides
		//{{AFX_VIRTUAL(CSoundPage)
		protected:
		virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
		//}}AFX_VIRTUAL
// Implementation
protected:
		// Generated message map functions
		//{{AFX_MSG(CSoundPage)
		virtual BOOL OnInitDialog();
		afx_msg void OnButtonPlay();
		afx_msg void OnBrowse();
		afx_msg void OnButtonStop();
		afx_msg void OnDestroy();
		afx_msg void On SelchangeListEvents();
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
};
// CG: This file was added by the Splash Screen component.

#ifndef _SPLASH_SCRN_
#define _ SPLASH_SCRN_

// Splash.h : header file
//
/////////////////////////////////////////////////////////////////////////////
//  Splash Screen class

class CSplash Wnd : public CWnd
{
// Construction
protected:
		int m_iType;
		CSplashWnd();
// Attributes:
```


BUPT

```
public:
        void ChangeSplashScreen();
        CBitmap m_bitmap;
// Operation
public:
        static void EnableSplashScreen(BOOL bEnable = TRUE);
        static void ShowSplashScreen(CWnd* pParentWnd = NULL);
        static BOOL PreTranslateAppMessage(MSG* pMSG);
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSplashWnd)
        //}}AFX_VIRTUAL
// Implementation
public:
        ~CSplashWnd();
        virtual void PostNcDestroy();
protected:
        BOOL Create(CWnd* pParentWnd = NULL);
        void HideSplashScreen();
        static BOOL c_bShowSplashWnd;
        static CSplashWnd* c_pSplashWnd;
// Generated message map functions
protected:
        //{{AFX_MSG(CSplashWnd)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnPaint();
        afx_msg void OnTimer(UINT nIDEvent);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#endif

#if
!defined(AFX_STIMULUSDLG_H__C41093E2_B999_11D0_B539_000021847524__INCLUDED_)
#defineAFX_STIMULUSDLG_H__C41093E2_B999_11D0_B539_000021847524__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif// _MSC_VER >= 1000
// StimulusDlg.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CStimulusDlg dialog
class CStimulusDlg : public CDialog
{
// Construction
public:
        void SetSignalPen(CPen* penSignal);
        void SetSimulatorTime(double dTime);
        void SetStimulusHead(CStimulHead* obHead);
        CStimulusDlg(CWnd* pParent = NULL);  // standard constructor
// Dialog Data
        //{{AFX_DATA(CStimulusDLG)
        enum { IDD = IDD_INPUT };
        CStatic m_staticPos;
        CStatic m_staticStop;
        CStatic m_staticStart;
        CListCtrlEx     m_listInput;
        CStatic m_staticRect;
        int             m_radioType;
        //}}AFX_DATA
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CStimulusDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        // Generated message map functions
        // {{AFX_MSG(CStimulusDlg)
```


BUPT

```
		virtual BOOL OnInitDialog();
		afx_msg void OnInputAdd();
		afx_msg void OnInputDelete();
		afx_msg void OnInputDeleteAll();
		afx_msg void OnInputInsert();
		afx_msg void OnInputOwerwrite();
		afx_msg void OnPaint();
		afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
		afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
		afx_msg void OnMouseMove(UINT nFlags, CPoint point);
		afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
		afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
		//}}AFX_MSG
		DECLARE_MESSAGE_MAP()
private:
		int m_sx;
		void FillInputList();
		void DrawSignal(CDC* pDC);
		void AddInput(double dTime1, double dTime2, int iValue);
		int m_ixold;
		BOOL m_by;
		CPn m_penEDit;
		double m_dPos;
		CRect m_rcPos;
		BOOL m_bDrag;
		int m_ix;
		int m_iy;
		CPen* m_penSignal;
		BOOL m_bDown;
		int m_iCursorType;
		int m_dy;
		int m_dx;
		double m_dStop;
		double m_dStart;
		CFont* m_fontList;
		CRect m_rcStop;
		CRect m_rcStart;
		CRect m_rc;
		int m_iEditType;
		double m_dTime;
		CStimulHead* m_obHead;
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_STIMULUSDLG_H__C41093E2_B999_11D0_B539_000021847524__INCLUDED_)

// TabDlg.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CTabDlg dialog

class CTabDlg : public CDialog
{
// Construction
public:
		CTabDlg(CWnd* pParent = NULL);  // standard constructor
// Dialog Data
		//{{AFX_DATA(CTabDlg)
		enum { IDD = IDD_TAB };
		CString m_sTabStop;
		//}}AFX_DATA
// Overrides
		// ClassWizard generated virtual function overrides
		//{{AFX_VIRTUAL(CTabDlg)
		protected:
		virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
		//}}AFX_VIRTUAL
// Implementation
protected:
		// Generated message map functions
```


BUPT

```
        //{{AFX_MSG(CTabDlg)
                // NOTE: the ClassWizard will add member functions here
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
// TDL.h : main header file for the TDL application
//

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"    // main symbols

/////////////////////////////////////////////////////////////////
// CTDLApp:
// See TDL.cpp for the implementation of this class
//

class CTDLApp : public CWinApp
{
public:
        virtual BOOL PreTranslateMessage(MSG* pMsg);
        CTDLApp();
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTDLApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL
// Implementation
        //{AFX_MSG(CTDLApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        void ShowTipAtStartup(void);
private:
        void ShowTipOfTheDay(void);
};

/////////////////////////////////////////////////////////////////

// TDLDoc.h : interface of the CTDLDoc class
//
/////////////////////////////////////////////////////////////////

class CTDLDoc : public CDocument
{
protected: // create from serialization only
        CTDLDoc();
        DECLARE_DYNCREATE(CTDLDoc)
// Attributes
public:
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTDLDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~CTDLDoc();
#ifdef _DEBUG
        virtual void Assert Valid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
```


BUPT

```
// Generated message map functions
protected:
	//{{AFX_MSG(CTDLDoc)
	afx_msg void OnBuildAnalyse();
	afx_msg void OnFileExport();
	afx_msg void OnTdlWizard();
	afx_msg void OnBuildNextError();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

// CG: This file added by 'Tip of the Day' component.

/////////////////////////////////////////////////////////////////
// CTipDlg dialog

class CTipDlg : public CDialog
{
// Construction
public:
	CTipDlg(CWnd* pParent = NULL);	// standard constructor
// Dialog Data
	//{{AFX_DATA(CTipDlg)
	// enum { IDD = IDD_TIP };
	BOLL	m_bStartup;
	CString m_strTip;
	//}}AFX_DATA
	FILE* m_pStream;
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CTipDLg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation:
public:
	virtual ~CTipDlg();
protected:
	// Generated message map functions
	//{{AFX_MSG(CTipDlg)
	afx_msg void OnNextTip();
	afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
	virtual void OnOK();
	virtual BOOL OnInitDialog();
	afx_msg void OnPaint();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
void GetNextTipString(CString(CString& strNext);
};
// TxtView.h : header file
//

////////////////////////////////////////////////////
// CTxtView view

class CTxtView  : public CEditView
{
protected:
	int m_iTab;
	LOGFONT m_lfDefFont;
	CFont m_font;
	CTxtView();	// protected constructor used by dynamic creation
	DECLARE_DYNCREATE(CTxtView)
// Attributes
public:
	void TxtGotoLine(int iLine);
// Operations
public:
// Overrides
	// ClassWizard generated virtual function overrides
```


BUPT

```
	//{{AFX_VIRTUAL(CTxtView)
	public:
	virtual void OnInitialUpdate();
	protected:
	virtual void OnDraw(CDC* pDC);   // overriden to draw this view
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	//}}AFX_VIRTUAL
// Implementation
protected:
	virtual ~CTxtView();
#ifdef _DEBUG
	 virtual void Assert Valid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
	// Generated message map functions
protected:
	//{{AFX_MSG(CTxtView)
	afx_msg void ONChooseFront();
	afx_msg void OnTxtGoto();
	afx_msg void OnToolsSettab();
	afx_msg void OnUpdate();
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	afx_msg void OnKeyDown(UINT  nChar, UINT nRepCNT, UINT nFlags);
	//}}AFX_MSG
	afx_msg LRESULT OnUpdateLineInfo(WPARAM wParam, LRESULT lParam):
	DECLARE_MESSAGE_MAP()
private:
	 void UpdateLineInfo();
};

/////////////////////////////////////////////////////////////////////////////
```


BUPT

Referinţe bibliografice

[ABRA96] M.Abramovici, M.A.Breuer, A.D.Friedman
Digital Systems And Testable Design
Computer Science Press, 1996

[OKLO82] Oklobodzija and Ercegovac
Testability Enhancement Of VLSI Using Circuit Structures
Proc. IEEE ICCC '82, pp.198-201, 1982

[AGAR81] V.K.Agarwal and A.S.Fung 1981
Multiple Fault Testing Of Large Circuits By Single Fault Test Sets
IEEE Trans. on Comp., vol. C-30, no. 11, November 1981 pp. 855-865

[BARZ81] Z. Barzilai, J. Savir, G. Markowsky, and M. G. Smith,
The Weighted Syndrome Sums Approach To VLSI Testing,
IEEE Trans. on Computers, Vol. C-30, No. 12, pp. 996-1000, December, 1981.

[BASS94A] Sulaiman Al-Bassam, Bella Bose
Dessign Of Efficient Balanced Codes
IEEE Trans. on Comp., vol. 43, no.3, March 1994, pp. 362-365

[BASS94B] Sulaiman Al-Bassam, Bella Bose 1994
Asymetric/Unidirectional Error Correcting And Detecting Codes
IEEE Trans. on Comp., vol. 43, no.5 May 1994, pp. 590-597

[BAYO83] M.A.Bayoumi, G.A. Jullien, and W.C. Miller.
An area-time efficient NMOS adder. INTERGRATION,
The VLSI Journal, vol.1 1983, pp317-334.

[BHAT89] Dhargab B. Bhattacharya, Sharad C. Seth
Design Of Parity Testable Combinational Circuits
IEEE Trans. on Comp., vol. 38, no. 11, December 1989, pp. 1580-1584

[BOND91] M.Bondjit, M.Nicolaidis
A Tool For Computation Of Output Code Spaces In Complex Self-Checking Systems
IEEE Trans. on Comp., December 1991, pp. 122-127

[BOSE86] Bella Bose
Burst Unidirectional Error-Detecting Codes
IEEE Trans. on Comp., vol. C-35, no. 4, April 1986, pp. 350-353

[BOSE91] Bella Bose
On Unordered Codes
IEEE Trans. on Comp., vol.40, no. 2, February 1991, pp. 125-131

[BOUB95] Samir Boubezari, Bosera Kaminska
A Deterministic Built-In Self Test Generator Based On Cellular Automata Structures
IEEE Trans. on Comp., vol. 44, no. 6, June 1995, pp. 805-816

[BREN82] R.P.Brent, and H.T. Kung
A regular layout for parallel adders
IEEE Trans. on Comp., vol. C-31, no. 3, 1982, pp 260-264.

[BREU88] M.A.Breuer, R.Gupta and J.C.Lien
Concurrent Control of Multiple BIT Structures
Proc. Intn'l.Test Conf. September 1988, pp. 431-442

[BURN94] Steven W. Burns, Niraj K. Jha
A Totally Self-Checking Checker For A Parallel Unordered Coding Scheme
IEEE Trans. on Comp., vol. 43, no. 4, April 1994, pp. 490-495

[CĂTU85] Vasile M. Cătuneanu, Ioan C. Bacivarof
Fiabilitatea sistemelor de telecominicaţii
Editura militară Bucureşti 1985

[CĂTU89] Vasile M. Cătuneanu, Angelica Bacivarof
Structuri electronice de înaltă fiabilitate. Toleranţa la defectări
Editura militară Bucureşti 1989

[CHAK91] S.Chakravarty, I.Gertsbakh and M.Lomonosov
On Computing Reliability-Measures Of Boolean Circuits
IEEE Transactions on Reliability, vol. 40, no. 5, December 1991, pp. 582-591

[CHAN90] P.K.Chan and M.DF.Schlag. 1990
Analysis And Design Of CMOS Manchester Adders With Variable Carry-Skip.
IEEE Trans. on Comp., vol. 39, no. 8, 1990, pp. 983-993

BUPT

[CHAN92] P.K.Chan, M.DF.Schlag, C.D.Thomborson and V.G.Oklobdzija. 1992
Delay Optimization Of Carry-Skip Adders And Block Carry-Lookahead Adders Using Multidimensional Dynamic Programming.
IEEE Trans. on Comp., vol. 42, no. 10, 1992, pp.1163-1170

[CHEN85] W. T. Cheng and J.H. Patel.
Ashortest Length Test Sequence For Sequential-Fault Detection In Ripple Carry Adders,
Proc. Intn'l. Conf. on Computer-Aided Design, pp. 71-73, November, 1985.

[CHEN87] C. L. Chen
Exhaustive Test Pattern Generation Using Cyclic Codes
IEEE Trans. on Computers, Vol. 37, No. 3, March, 1987, pp. 329-338

[CHEN88] C. I., Chen.
Exhaustive Test Pattern Generation Using Cyclic Codes
IEEE Trans. on Computers, Vol. C-37, No. 2., February, 1988, pp. 225-228

[CHIE94] Chien-In Henry Chen, Anup Kumar
Coments On Area-Time Optimal Adders Design
IEEE Trans. on Comp., vol. 43, no. 4, 1994, pp.507-512

[CHIH94] Chih-Tuang Lu, Cheng-Wen Wu 1994
Testing Iterative Logic Arrays For Sequential Faults With A Constant Number Of Patterns
IEEE Trans. on Comp., vol. 43, no. 4, April 1994, pp. 495-501

[CHOW94] Sulaiman Al-Bassam, Bella Bose
Asymetric/Unidirectional Error Correcting And Detecting Codes
IEEE Trans. on Comp., vol. 43, no.6, June 1994, pp. 759-764

[CHUN94] Jan-Chung Lo
Reliable Floating-Point Arithmetic Algorithms For Error-Coded Operands
IEEE Trans. on Comp., vol. 43, no. 4, April 1994, pp. 400-412

[DAMP87] D.I.Damper, N.Burgess
Mos Test Pattern Generation Using Path Algebras
IEEE Trans. on Comp., vol. C-36, no. 9, September 1987, pp. 1123-1128

[DEMM94] Demmel, J.W. and X. Li.
Faster Numerical Algorithms Via Exception Handling
IEEE Trans. on Comp., vol. 43, no. 8, 1994, pp. 983-992

[DIMA95] V.V. Dimakopoulos, G.Sourtziotis, A. Paschalis, D.Nikolos 1995
On TSC Checkers For M-Out-Of N Codes
IEEE Trans. on Comp., vol. 44, no.8, August 1995, pp. 1055-1059

[DORA88] R.W.Doran
Variants on an improved carry lookahead-adder.
IEEE Trans. on Comp., vol. 37, no. 9, 1988,pp 1110-1113.

[DOTA90] Yoheved Dotan, Benjamin Arazi
Concurrent Logic Programing As A Hardware Description Tool
IEEE Trans. on Comp., vol. 39, no. 1, January 1990, pp. 72-88

[ELHU86] H. Elhuni, A. Vergis, and L. Kinney,
C-Testability Of Two-Dimensional Iterative Arrays,
IEEE Trans. on Comp.-Aided Design, Vol. Cad-5, No. 4, pp. 573-581, October, 1986.

[ERCE90] Miloš D. Ercegovac, Tomas Lang
On The Fly Conversion Of Redundant Into Conventional Representations
IEEE Trans. on Comp., vol. C-36, no. 7, July 1990, pp. 895-897

[FENW87] P.M.Fenwick
A fast-carry adder with CMOS transmission gates.
Computer Journal, vol. 30, no. 1, 1987, pp 77-79.

[FRIE73] A.D. Friedman
Easily Testable Iterative Systems
IEEE Trans on Computers, Vol. C-22, No.12, Dec.1973, pp.1061-1064

[RAO89] T.R.N.Rao, E. Fujiwara
Error-Control Coding for Computer Systems
Prentice Hall, 1989

[FUJI83] H.Fujiwara and T.Shimono
On The Acceleration Of Test Generation Algorithms
IEEE Trans. on Comp., vol. C-32, no. 12, December 1983, pp. 1137-1144

[FUJI90] Hidao Fujiwara

BUPT

Logic Testing And Design For Testability
Computer Systems Series The MIT Press, 1990

[FUJI91] Eiji Fujiwara, Masakatsu Yoshikawa
A Design Method For Cost-Effective Self-Testing Checker For Optimal D-Unidirectional Error Detections Codes
IEEE Trans. on Comp., 1991, pp. 122-127

[GAJS94] Daniel D. Gajski, L. Ramachandran
Introduction to High Level Synthesis
Design&Test of. Computers, Winter 1994, pp. 44-54

[GAST83] N.Gastanis and C.Halatsis
A New Design Method For M-Ou-Of-N TSC Checkers
IEEE Trans. on Comp., vol. C-32, no. 3, March 1983, pp. 273-283

[GELS87] P. P. Gelsinger,
Design And Test Of The 80386
IEEE Design &Test of Computers, Vol.4, No. 3, pp. 42-50, June, 1987.

[GLOS89] C. Gloster, Jr., and F. Brglez
Boundary Scan with Built-In Self Test
IEEE Design &Test of Computers, Vol. 6, No. 1, February, 1989, pp. 36-44

[GOEL81] P.Goel and B.C.Rosales
PODEM-X: An Automatic Test Generation System For VLSI Logic Structures
Proc. 18th Design Automation Conf. June 1981, pp. 260-268

[GOSL80] Gesling, J.B.
Design Of Arithmetics Units For Digital Computers, Springer-Verlag, New-York, 1980

[GUYO87] A.Guyot, B. Hochet, J-M. Muller
A way to build efficeint-carry-skip adders
IEEE Trans. on Comp., vol. C-36, no. 10, 1987, pp 1144-1152.

[HAMA84] V.C.Hamacher, Z.G. Vranesic and S.G.Zaky. 1984
Computer Organization, 2nd ed., McGraw-Hill New York

[HAN87]. T.Han. and D.A. Carlson.
Fast area-eficient VSLI adders.
In: Proceedings, 8th IEEE Symposium on Computer Arithmetic, 1987, pp 49-56.

[HASS93] Hassan Rajaci, Rassul Ayani
Design Issue in Parallel Simulation Languages
IEEE Design&Test of Computers, December 1993, pp. 52-63

[HATE91] Shin'ichi Hatekenaka, Takashi Nanya
A Design Method Of SFS And SCD Combinational Circuits
IEEE Trans. on Comp., vol. 40, 1991, pp. 168-173

[HEID91] Klaus D. Heidtmann
Arithmetic Spectrum Applied To Fault Detection For Combinational Networks
IEEE Trans. on Comp., vol. 40, no. 3, March 1991, pp. 320-324

[HEND61] H.C.Hendrickson,
Fast High-accuracy binary parallel addition.
IRE Transactions on Electronic Computers, EC-10, 1961, pp. 465-468.

[HERO90] D.Heron, M. Kruckenberg, S. Soren, and F. Wei
A modified Manchester carry-skp adder.
Technical Report No. UCSC-CRL-89-10. Computer Research Laboratory, University of California, Sanra Cruz, 1990

[HOHA82] Ion Hohan
Tehnologia şi fiabilitatea sistemelor
Editura didactică şi pedagogică Bucureşti 1982

[HORT89] P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller, and H. C. Card
Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test
IEEE Trans. on Comp.-Aided Design, Vol.8, No. 8, August, 1989, pp. 842-859

[ITO91] Hideo Ito
A 2-Rail Logic Combinational Circuit With Easy Detection Of Stuck-Open And Stuck-On Faults In Fets'
IEEE Trans. on Comp., vol. 40, 1991, pp. 252-257

[JHA89] N.K.Jha
A Totally Self-Checking Checker For Borden's Code
IEEE Transactions on Computer-Aidded Design, vol.8, no. 7, July 1989, pp. 731-736,

BUPT

[JONE89A] W. B. Jone and C. A. Papachristou
A Coordinated Approach to Partitioning and Test Pattern Generation for Pseudoexhaustive Testing Proc. 26th Design Automation Conf., June, 1989, pp. 525-530

[JONE89B] W. B. Jone, C. A. Papachristou, and M. Pereina
A Scheme for Overlaying Concurrent Testing of VLSI Circuits
26th Design Automation Conf., June, 1989, pp. 513-536

[KANA95] Ghani A. Kanawati, Nasser A. Kanawati, Jacob A. Abraham
FERRARI: A Flexible Software-Based Fault And Errors Injection System
IEEE Trans. on Comp., vol. 44, no. 2, February 1995, pp. 248-260

[KANT93] Vitit Kantabutra
A Recursive Carry-Lookahead/Carry-Select Hybrid Adder
IEEE Trans. on Comp., vol. 42, no. 12, December 1993 pp. 1495-1499

[KAPU92] Rohit Kapur, M. Ray Mercer
Bounding Signal Probabilities For Testing Measurement Using Conditional Syndromes
IEEE Trans.Comput., vol.41, no.12, pp 1580 - 1587 December 1992

[KARK94] Jounès Karkouri, El Mostapha Aboulhamid, Eduard Cerny, Allain Verreault 1994
Use Of Fault Dropping For Multiple Fault Analysis
IEEE Trans. on Comp., vol. 43, no. 1, January 1994, pp. 98-103

[KINO80] K.Kinoshita, Y.Takamatsu and M.Shibata
Test Generation For Combinational Circuits By Structure Description Functions
Proc.10th Int.Symp.Fault-Tolerant Computing pp. 152-154

[KORE89] L. Koren and S.-W Fu
Computer Arithmetic Algorithms, Prentice Hall, Englewood Cliffs. N.J, 1989

[KORN94] Peter Kornerup
Digital-Set Conversions: Generalizations And Applications
IEEE Trans. on Comp., vol. 43, no. 5, May 1994, pp. 622-629

[KUMA94] Vipin Kumar, Ananth Grama, Anshul Gupta, George Karypsis
Introduction To Parallel Computing. Design And Analysis Of Algorithms
The Benjamin/Cummings Publishing Company, Inc. 1994

[KUND90] Sandip Kundu, Sudhamar Reddy
Embedded Totally Self-Checking Checkers: A Practical Design
IEEE Design&Test of Computers, August 1990, pp. 5-12

[LAPR85] J.C.Laprie
Dependable computing and fault tolerance: Concepts and terminology
Proceedings, 9th Symposium on Computer Arithmetic, 1985, pp. 96-103

[LEIG92} F.T. Leighton
Introduction To Parallel Algorithms And Architectures: Arrays, Trees, Hypercubes.
Morgan Kaufmann, San Mateo, Calif.

[LEVE80] Y.H. Levendel and P.R.Menon
Comparison Of Fault Simulation Methods-Treatment Of Unknown Signal Values
Jurnal of Digital Systems, Vol.4, Winter 1980, pp. 443-459

[LING81} H. Ling,
High-speed binary adder.
IBM Journal of Research and Development, vol. 25, no. 3, 1981, pp 156-166.

[LOMB90] F. Lombardi, W.-K.Huang 1990
Fault Detection and Design Complexity in C-Testable VLSI Arrays
IEEE Trans. on Comp., vol. 39, no. 12, December 1990, pp. 1477-1481

[LYNC92} T.Lynch, and E.E. Swartzlander
A Spanning Tree Carry Lookahead Adder.
IEEE Trans. on Comp., vol. 41, no. 8, 1992, pp.45-58

[MAJU95] Amitava Majundar, Sarma B.K.Vrudhala
Fault Coverage And Test Length Estimation For Random Pattern Testing
IEEE Trans. on Comp., vol. 45, no. 2, February 1995, pp. 234-247

[MARK87] George Markowsky
Boundintg Signal Probabilities In Combinational Circuits
IEEE Trans. on Comp., vol. C-36, no. 10, October 1987, pp. 1247-1250

{MARK90] P.W. Markstein
Computation Of Elementary Functions On The IBM RISK System/6000 Processor
IBM J.of Research and Development 34:1, 1990, pp. 111-119

[MART80] N.M.Martin, and S.P.Hufnagel

BUPT

Conditional-Sum Early Completion Adder Logic.
IEEE Trans. on Comp., vol. C-29, no.8, 1980, pp. 753-756

[MCCL81] E. J. McCluskey and S. Bozorgui-Nesbat
Design for Autonomous Test
IEEE Trans. on Computers, Vol. C-30, No. 11, November, 1981, pp. 866-875

[MCCL84] E. J. McCluskey
Verification Testing - A Pseudoexhaustive Test Technique
IEEE Trans. on Computers, Vol. C-33, No. 6, June, 1984, pp. 541-546

[MEAD80] C.Mead, and L. Conway.
Introduction to VLSI Systems. Addison-Weslwy, reading, Massachusetts, 1980

[MENO88] P.R.Menon, Y.H. Levendel and M.Abramovici
Critical Path Tracing In Sequential Circuits
Proc. Intn'l Conf. On Computer-Aided Design, November 1988, pp. 162-165

[MONT90] R.K.Montoye, E.Hokonek, and S.L. Runyan.
Design Of The Floating-Point Execution Unit In The IBM RISC System/6000.
IBM Jurnal of Research and Development, vol.34, no.1, 1990, pp. 69-70

[MULL94] Jean-Michel Muller 94
Some Characterizations Of Functions Computable In On-Line Arithmetic
IEEE Trans. on Comp., vol. 43, no. 6, June 1994, pp. 752-755

[NGAI84] T.F.Ngai, M.J. Irwin, and S. Rawat.
Regular area-time efficient carry-lookahead adders.
Journal of parallel and Distributed Computing, vol. 3, no. 3, 1984, pp 92-105.

[NGAL85] T.F.Ngal, and M.J.Irwin
Regular, Area-Time Efficient Carry-Lookahead Adders.
Proc. Seventh IEEE Symposium on Computer Arithmetic, 1985, pp. 9-15

[NICO87] M. Nicolaidis
Evaluation Of A Self-Checking Version Of The MC 68 000 Microprocessor
Microprocessing and Microprogramming Vol.20, pp. 235-247, 1987

[NIKO88] D.Nikolos, A.M.Paschalis, C.Halatsis and G.Philokyprou
Efficient Design Of Totally Self-Checking Checkers For All Low-Cost Arithmetic Codes
IEEE Trans. on Comp., vol. 37, no. 2, July 1988, pp. 807-814

[NIKO91] Dimitris Nikolos
Theory And Design Of T- Error Correcting / D-Error Detecting (D>T) And All Unidirectional Error Detecting Codes
IEEE Trans. on Comp., vol. 40, no. 2, February 1991, pp. 132-142

[OMON94] Amos R. Omondi
Computer Arithmetic Systems
Prentice Hall International Series in Computer Science, 1994

[PART81] R. Parthasarathy and S. M. Reddy,
A Testable Design Of Iterative Logic Arrays.
IEEE Trans. on Computers, Vol. C-30, no. 11, pp. 833-841. November, 1981.

[PATA83] O. Patashnik
Circuit Segmentation for Pseudoexhaustive Testing
Center for Reliable Computing Technical Report No. 83-14, Stanford University, 1983.

[PATT94] D.A.Patterson and J.L.Hannessy
Computer Organization and Design: The Hardware/Software Interface,
Morgan Kaufmann Publishers, Inc., San Francisco, California Cap.4, 1994

[PATT96] D.A. Patterson and J.L.Hannessy
Computer Organization And Design: The Hardware/Software Interface
Morgan Kaufmann Publishers Inc., San Francisco, California Anexa scrisă de Goldberg, 1996

[PIES87] Stanislaw J.Piestrak 1991
Concurrently Totally Self-Checking Microprogram Control Unit With Duplication Of Microprogram Sequencer
Microprocessing and Microprogramming Vol.20, pp. 271-281, 1987

[POME94] Irith Pomeranz, Sudhakar M. Reddy
Application Of Homing Sequences To Synchronous Sequential Circuit Testing
IEEE Trans. on Comp., vol. 43, no. 5, May 1994, pp. 569-580

[POME95A] Irith Pomeranz, Sudhakar M. Reddy

BUPT

On Fault Simulation For Synchronous Sequential Circuits
IEEE Trans. on Comp., vol. 44, no. 2, February 1995, pp. 335-340

[POME95B] Irith Pomeranz, Sudhakar M. Reddy
INCREADYBLE: A New Search Strategy For Design Automation Problems With Applications To Testing
IEEE Trans. on Comp., vol. 44, no. 6, June 1995, pp. 891-895

[POPE92A] D.Popescu, C.Popescu
Izolarea defectelor cu ajutorul modulelor comutatoare de testare
Analele universitatii din Oradea, vol.I., Fascicola de Electrotehnica si Energetica 1992, pag.270-275

[POPE92B] D.Popescu, C.Popescu
Metoda de proiectare pentru testabilitate bazata pe proiectarea ierarhica si pe utilizarea ATPG-ului
Analele universitatii din Oradea, vol.I., Fascicola de Electrotehnica si Energetica 1992, pag.276-281

[POPE94A] D.Popescu, C.Popescu
Limitarea probabilitatilor de semnal pentru masuratori de testabilitate utilizând sindromuri conditionale
Analele universitatii din Oradea, Fascicola de Electrotehnica si energetica 1994, pag.580-584

[POPE94B] D.Popescu, C.Popescu
An Algorithm For Solving The Generalize Probability Problem For Boolean Circuits
Proceedings of FEI '25 Conference on Electronic Computers and Informatics, September 22-23, 1994 Kosice Herl'any Slovakia, pp.126-132

[POPE94C] D.Popescu, C.Popescu
A Boolean Algebric Algorithm To Compute The Reliability
Proceedings of FEI '25 Conference on Electronic Computers and Informatics, September 22-23, 1994 Kosice Herl'any Slovakia, pp.109-114

[POPE95A] D.Popescu, C.Popescu
Simularea defectelor de întârziere pentru procesarea paralelă a pattern-urilor de test
Analele universitatii din Oradea, Fascicola de Electrotehnica 1995 secţiunea D, pag.114-119

[POPE95B] D.Popescu, C.Popescu
Coduri detectoare şi corectoare a erorilor unidirecţionale pe byte
Analele universitatii din Oradea, Fascicola de Electrotehnica 1995secţiunea D, pag.120-125

[POPE95C] D.Popescu, C.Popescu
The Use Of Miller's Cube Reprezentation In Reliability Analysis
113 th Pannonian Applied Mathematical meeting, Oktober 11-15, 1995, pag159-162

[POPE96] D.Popescu, C.Popescu
Nonrobust Path Delay Fault Simulation By Parallel Processing Of Patterns
International Symposium on SystemsTheory Robotics, Computers & Process Informatics, Craiova 1996, Section Computer Science & Engineering

[POPE97A] D.Popescu, C.Popescu
Implicaţiile căilor critice în analiza circuitelor digitale
Analele universit.Oradea, 1997, fascicola fac. de Electrot. şi Informat.

[POPE97B] D.Popescu, C.Popescu
.Sequential Multipler with Carry Save Adder
119 th Pannonian Applied Mathematical meeting, Herl'any
Applied & Computing Matematics Oktober 23-26, 1997, pag167-171

[POPE98A] D.Popescu, C.Popescu, M.Pater
An Algorithm For Determining The Minimal Test Set For The Carry-Lookahead Adder (CLA)
Analele Universităţii din Oradea, Fascicola de Electrotehnică, Secţiunea D, pag.87-92

[POPE98B] D.Popescu, C.Popescu, M.Pater
The C-Testability Analyze Of The Block Carry-Lookahead Adder (BCLA)
Analele Universităţii din Oradea, Fascicola de Electrotehnică, Secţiunea D, pag.81-86

[POPE98C] D.Popescu, C.Popescu
The C-Testability Analyze Of The Ripple-Block Carry-Lookahead Adder (RCLA))
International Symposium on SystemsTheory Robotics, Computers & Process Informatics,

BUPT

Craiova 1998, Section Computer Science & Engineering
[POPE98D] D.Popescu, C.Popescu
Analiza C-testabilităţii sumatorului Carry Completion Adder (CCA)
A&Q98, International Conference and Quality Control,
QUALITY, Design and Development, vol. Q, Cluj , 1998 pp. 117-121
[POPE98E] D.Popescu, C.Popescu
The C-Testability Analyze For An Hybrid Pyramidal Carry-Lookahead Adder
FEI International Conference Electronic Cpmputers & Informatics'98, 8-9 Oktober, Slovakia,
[POPE98F] C.Popescu, D.Popescu, S.Ungureanu
The Analyze On The Test Pattern Generated By A Multiple Input Signature Register
Analele Universităţii din Oradea, Fascicola de Electrotehnică, Secţiunea D, pag.85-89
[POPE98G] D.Popescu, C.Popescu
The Pyramidal Carry Lookahead Adder (PyCLA)
Buletinul ştiinţific al Universităţii Politehnica Timişoara, Seria Automatică şi Calculatoare,
Transactions on Automatic and Computer Science, Special Issue Dedicated on Technical Informatics CONTI'98, October 29-30, 1998,Timişoara, pp. 151-159
[PSAC87] A.M.Pschalis, C.Halatsis and G.Philokyprou
Efficient Design Of Totally Self-Checking Checkers For All Low-Cost Arithmetic Codes
Microprocessing and Microprogramming Vol.20, pp. 271-281, 1987
[RAJS93] Janusz Rajski, Jerzy Tyszer
A Recursive Pseudoexhaustive Test Pattern Generation
IEEE Trans. on Comp., vol. 42, no. 12, December 1993, pp. 1517-1521
[SALU88] K. K. Saluja, R. Sharma, and C. R. Kime
A Concurrent Testing Technique for Digital Circuits
IEEE Trans. on Comp.-Aided Design, Vol. 7, No. 12, December, 1988, pp. 1250-1259
[SAVA95] Y.Savaria, F.Darlay
Producing Reliable Initialization And Test Of Sequential Circuits With Pseudorandom Vectors
IEEE Trans. on Comp., vol. 44, no. 10, Octomber 1995, pp. 1251-1256
[SAVI84] I.Savir, P.H.Bardell
On Random Pattern Testlength
IEEE Trans.Reliab., vol.C33, pp 467-474 Jun. 1984
[SETH90] Sharad C. Seth, Vishwani D.Agrawal, Hassan Farhat 1990
A Statistical Theory Of Digital Testability
IEEE Trans. on Comp., vol. 39, no. 4, April 1990, pp. 582-586
[SHEN84] J. P. Shen and F.J. Ferguson.
The Design Of Easily Testable VLSI Array Multipliers.
IEEE Trans. on Computers, vol. C-33., no. 6, pp. 554-560. June, 1984.
[SHEN94] Meng-Lieh Shen, Chung L.L. 1994
Simplifying Sequential Circuit Test Generation
IEEE Design&Test of Computers, Fall 1994, pp. 28-38
[SHI91] Zhi-gang Shi, Zheng-hui Lin 1991
Test Generation Algorithm For Incomplete Scan Design Circuits With Tri-State Devices
IEEE Trans. on Comp., vol. 40, 1991, pp. 206-211
[SMIT78] J.E.Smit and G.Metze
Strongly Fault Secure Logic Network
IEEE Trans. on Comp., vol. C-27, no. 6, June 1978, pp. 495-499
[SRID81A] T.Sridhar and J.P.Hayes
A Functional Approach to Testing Bit-Sliced Microprocessors
IEEE Trans. on Computers, Vol. C-30, No. 8, August, 1981, pp. 563-571
[SRID81B] T.Sridhar and J.P.Hayes
Design of Easily Testable Bit-Sliced Systems
IEEE Trans. on Computers, Vol. C-30, No. 11, November, 1981, pp. 842-854
[STAN87] Stanislaw J.Piestrak 1987
Design Of Fast Self-Testing Checkers For A Class Of Berger Codes
IEEE Trans. on Comp., vol.C-36, no. 5, May 1987, pp. 629-634
[STAN94] Stanislaw J.Piestrak

BUPT

Design Of Residue Generators Of Multioperand Modular Adders Using Carry-Save Adders
IEEE Trans. on Comp., vol. 43, no. 1, January 1994, pp. 68-77

[SWAR90] E.ED. Swaetzlander
Computer Arithmetic,
IEEE Computer Society Press, Los Alamitos, Calif., 1990

[TANG83] D. T. Tang and L. S. Woo
Exhaustive test Pattern Generation with Constant Weigth Vectors
IEEE Trans. on Computers,Vol. C-32, No. 12, Decemner, 1983, pp. 1145-1150

[TANG84A] D. T. Tang and C. L. Chen
Logic Test Pattern Generation Using Linear Codes
IEEE Trans. on Computers, Vol C-33, No. 9, September, 1984, pp. 845-850

[TANG84B] D. T. Tang and C. L. Chen
Iterative Exhaustive Pattern Generation for Logic Testing
IBM Journal of Research & Development, Vol. 28, March, 1984, pp. 212-219

[TEOD88] Dan Teodorescu
Automatizări microelectronice
Editura tehnică Bucureşti 1988

[TING93] Ting-Ting Y. Lin 1993
A New Framework For Designing: Built-In Test Multichip Modules With Pipelined Test Strategy
IEEE Design&Test of Computers, December 1993, pp. 38-51

[TOKA91] Hiroshi Tokahashi, Nobukage Iuchi, Yuzo Takamatsu
Test Generation For Combinational Circuits With Multiple Faults
IEEE Trans. on Comp., vol. 40, 1991, pp. 212-217

[TOMA93] Donald E.Thomas, Jay K.Adams, Herman Schmit 1993
A Model And Methodology For Hardware-Software Codesign
IEEE Design&Test of Computers, September 1993, pp. 6-15

[TUNG87] Wu-Tung Cheng, Janak H. Patal
A Minimum Test Set For Multiple Fault Detection In Ripple Carry Adders
IEEE Trans. on Comp., vol. C-36, no. 7, July 1987, pp. 891-895

[TURR89] S.Turrini
Optimal Group Distribution In Carry-Skip Adders.
Proceedings, 15th Symposium on Fault Tolerant Computing, June 19-21, 1985, pp. 2-11

[TYAG93] A. Tyagi
A Reduced-Area Scheme For Carry-Select Adders
IEEE Trans. on Comp., vol. 42, no. 10, 1993, pp. 1163-1170

[UDEL86] J. G. Udell, Jr.
Test Set Generation for Pseudoexhaustive BIST
Proc. Intn'l Conf. on Computer-Aided Design, Noveber, 1986, pp. 52-55

[UNGE95] Stephen Unger
Hazards, Critical Races And Metastability
IEEE Trans. on Comp., vol. 44, no. 6, June 1995, pp. 754-768

[VASA85] N. Vasanthavada and P. N. Marinos,
An Operationally Efficient Scheme For Exhaustive Test-Pattern Generation Using Linear Codes,
Proc. Intn'l Test Conf., pp. 476-482, November, 1985.

[VLĂD89A] Mircea Vlăduţiu
Tehnologie de ramură şi fiabilitate
curs litografiat IPTVT 1989

[VLĂD89B] Mircea Vlăduţiu, Marius Crişan
Tehnica testării echipamentelor automate de prelucrare a datelor
Editura Facla, Timişoara 1989

[WAGN87] Kenneth D. Wagner, Carry K.Chin, Eduard J.McCluskay
Pseudorandom Testing
IEEE Trans. on Comp., vol. C-36, no. 3, March 1987, pp. 332-343

[WANG87] L. T. Wang and E. J. McCluskey
Circuits for Pseudoexhaustive test Pattern Generation
IEEE Trans. on Comp.-Aided Design, Vol. 7, no. 1, January, 1987, pp. 91-99

[WAST93] N. Waste and Eshraghian

BUPT

Principles of CMOS VLSI Design: A Systems Perspective
2nd ed., Addison Wesley, Reading Mass, 1993

[WEI90]. Wei, B.W.Y. and C.D. Thomson
Area-time optimal adder designs.
IEEE Trans. on Comp., vol. 39, no. 5, 1990, pp 666-675.

[WOLF89] W.H.Wolf
How To Build A Hardware Description And Measurement System On An Object Oriented Programming Language
IEEE Transactions on Computer-Aided Design, vol. 8, no. 3, March 1989, pp. 288-301

[ZARO90] Christopher J.Zarowski, Howard C.Card
On Addition And Multiplication With Hensel Codes
IEEE Trans. on Comp., vol. 39, no. 12, December 1990, pp. 1417-1423

BUPT