Cursul 6

6.0 Testarea defectelor blocaj singulare (DBS)

În cele ce urmează vom examina metodele de generare a testelor pentru DBS. Anterior am discutat utilitatea și larga aplicabilitate a modelului DBS. Multe metode de generare a testelor pentru alte modele de defecte extind principiile și tehnicile folosite pentru DBS.
Procesul de generare a testelor (GT) depinde în primul rând de tipul de experiment de testare pentru care sunt generați vectorii de test. Toate considerațiile ce urmează sunt dedicate unei testări off-line, la nivelul barelor (pinilor) de intrare/ieșire, folosind pentru testare: vectori de test stocați și comparația completă a rezultatelor ieșirii.

6.1. Elemente fundamentale.

Generarea testelor este o problemă complexă cu multe aspecte de interacție. Cele mai importante sunt:

- costul GT;
- calitatea testelor generate;
- costul aplicării testelor;

Costul GT depinde de complexitatea metodei de GT. GT aleatoare (GTA) este un proces simplu ce implică doar o generare aleatoare a vectorilor de test. Totuși, pentru atingerea unei bune calități a testului - calitate măsurată să zicem prin acoperirea defectelor prin testul generat - avem nevoie de o mulțime mare de vectori de test generați aleator. Chiar dacă GT (însăși) este simplă, determinarea calității testului - spre exemplu, prin simularea defectelor - poate fi un proces costisitor.
Mai mult, un test mai lung (constituit într-o secvență mai numeroasă) poate costa mai mult deoarece crește timpul experimentului de testare și necesarul de memorie al dispozitivului ce realizează experimentul de testare (testerul).

GTA în general nu ia în considerație funcția sau structura circuitului ce urmează a fi testat. Spre deosebire de GTA, GT deterministă (GTD) produce teste pornind de la un model al circuitului. Comparativ cu GTA, GTD este mult mai costisitoare, dar produce teste mai scurte și de calitate mai bună. GTD poate fi manuală sau automată. În continuare ne vom concentra atenția doar asupra GTD automate.

GTD poate fi orientată pe defect sau poate fi independentă de defect. În primul caz testele sunt generate pentru defecte specificate din universul de defecte (universul de defecte definit printr-un model explicit al defectelor). GTD independentă de defect nu ține seama de specificitatea unui defect țintă individual.

Costul GTD depinde de complexitatea circuitului. Metode de reducere a complexității cu efecte asupra testării sunt grupate sub denumirea generică de: metode de proiectare pentru testabilitate.

Figura 6.1. arată schema generală a unui sistem determinist de GTD. Testele sunt generate în baza unui model al circuitului și pentru un model al defectului precizat.

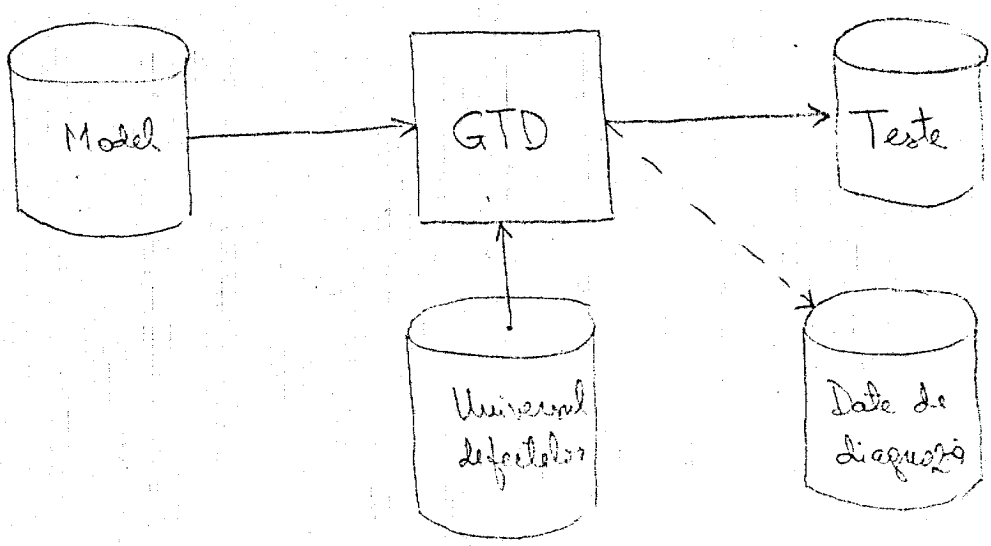

Figura 6.1 Sistem de generare a testelor deterministe.

Testele generate includ atât stimulii ce vor fi aplicați cât și răspunsul așteptat al circuitului corect (liber de defecte). Împreună GTD produc deasemenea date de diagnoză ce pot fi folosite pentru localizarea defectelor.

6.2. GTD pentru DBS în circuitele combinaționale (CC).

În această secțiune vom considera numai circuite combinaționale la nivel de poartă compuse din porți SI, SI-NU, SAU, SAU-NU și NU.

6.2.1. GTD orientate pe defecte.

• Circuite libere de ramificații.

Circuitele libere de ramificații vor fi folosite numai ca vehicul de introducere a principalelor concepte a GTD pentru circuite generale. Cei doi pași fundamentali în generarea unui test pentru un DBS: linia l b-l-v (v fiind 0 sau 1) sunt:

- activarea (excitarea) defectului b;
- propagarea erorii rezultate la o linie primară de ieșire (LPE).

Activarea defectului înseamnă poziționarea valorilor liniilor primare de intrare (LPI) astfel încât să se cauzeze (în circuitul liber de defecte) aducerea liniei l la v valoarea v' (valoarea opusă valorii blocajului). Aceasta constituie o instanțiere a problemei justificării unei linii, problemă ce tratează găsirea unei atribuiri a valorilor LPI a.î. să rezulte valoarea dorită pentru o anumită linie din circuit. Pentru a urmări propagarea erorii trebuie să considerăm valori atât în circuitul liber de defecte N cât și în circuitul defect N_f definit de defectul f.

Valorile logice compozite ce reprezintă erorile — 1/0 și 0/1 — sunt tradițional notate prin simbolurile D și respectiv D'. Celelalte două valori compozite — 0/0 și 1/1 — sunt notate 0 și respectiv 1. Orice operație logică între două valori compozite poate fi realizată prin prelucrarea separată a celor două valori. Spre exemplu: $D' + 0 = 0/1 + 0/0 = 0+0/1+0 = 0/1 = D'$.

Acestor patru valori binare compozite adăugăm o a cincea valoare (x) ce reprezintă o valoare compozită nespecificată, adică orice valoare din mulțimea $\{0, 1, D, D'\}$. În practică operațiile logice cu valori compozite sunt definite prin tabele (a se vedea figura 6.2).

v/v_f	
0/0	0
1/1	1
1/0	D
0/1	D'

(a)

SI	0	1	D	D'	x
0	0	0	0	0	0
1	0	1	D	D'	x
D	0	D	D	0	x
D'	0	D'	0	D'	x
x	0	x	x	x	x

(b)

SAU	0	1	D	D'	x
0	0	1	D	D'	x
1	1	1	1	1	1
D	D	1	D	1	x
D'	D'	1	1	D'	x
x	x	1	x	x	x

(c)

Figura 6.2. Valori logice compozite și operații cu variabile logice pentavalente.

Figura 6.3 descrie un algoritm de generare a unui test pentru 1 b-l-v. Sunt inițializate la x toate valorile liniilor și apoi procedează în cei doi pași de bază, reprezentați de rutinele Justify și Propagate.

```
begin
    pune toate liniile la valoare x
    Justify (l, v')
    if v=0 then Propagate (l, D)
    else Propagate (l, D')
end
```

Figura 6.3 generarea testului pentru defectul 1 b-l-v într-un circuit liber de ramificații.

Justificarea unei linii (Figura 6.4) este un proces recursiv în care valoarea ieșirii fiecărei porți este justificată de valorile intrărilor porții și astfel recursiv până când se ajunge la LPI. Să considerăm o poartă (NAND) SI-NU cu k intrări. Există o singură cale de justificare a unui 0 la ieșire; în schimb se pot alege 2^k-1 combinații de intrare ce pot justifica o valoare 1 a ieșirii.

leg mai simplă cele ar fi atribuirea valorii 0 unei singure intrări (aleasă arbitrar) lăsând celelalte intrări nespecificate. Aceasta corespunde alegerii unuia din cele k cuburi primitive în care această poartă ia valoarea 1.

```
Justify (l, val)
begin
    poziționează linia l la valoarea val;
    /* l este ieșirea unei porți */
    c = valoarea de control a porții l;
    i = inversiunea porții l;
    ival = val ⊕ i;
    if (ival = c')
        then for every intrare j a porții l
            Justify (j, ival);
    else
        begin
            alege o intrare (j) a porții l;
            Justify (j, ival);
        end
end
```

Figura 6.4 Justificarea unei linii într-un circuit liber de ramificații.

Propagarea unei erori către o LPE a circuitului necesită sensibilizarea unei căi unice de la l la o LPE. Fiecare poartă de pe această cale are exact o singură intrare sensibilizată la defect. Evident trebuiesc poziționate toate celelalte intrări la valori necontrol pentru poarta respectivă.

Exemplul 6.1: Să generăm un test pentru defectul f b-l-ø în circuitul din figura 6.6(a). Problemele inițiale sunt: Justify (f, 1) și Propagate (f, D). Justify (f, 1) este rezolvată

```
Propagate (l, err)
/* err is D or D' */
begin
    l = err;
    if l este LPE then return
    k = fanout(l);
    c = valoare_de_control(k);
    i = inversiune(k);
    for every intrare j a portii k altă decât l
        Justify (j, c');
        Propagate (k, err ⊕ i);
end
```

Figura 6.5 Propagarea erorii într-un circuit liber de ramificații.

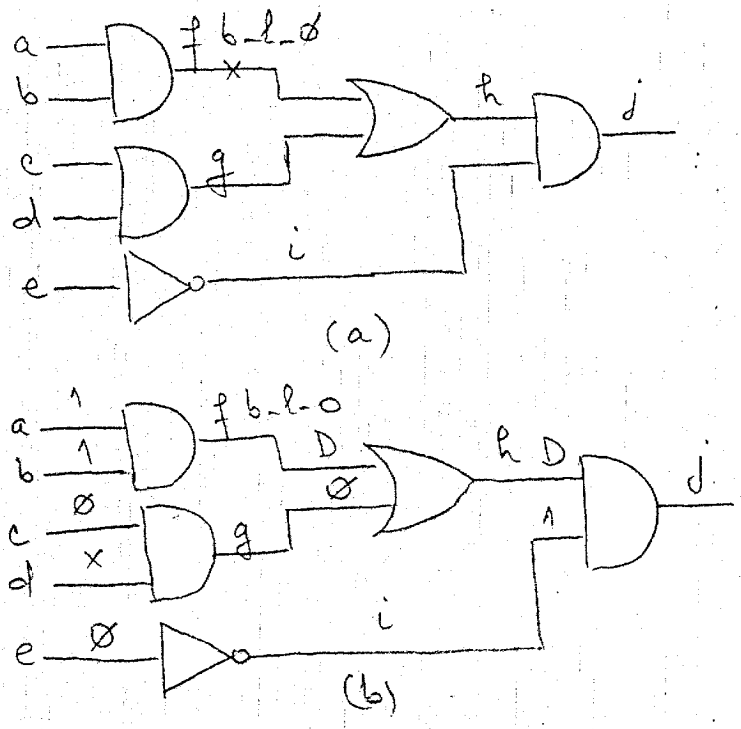

Figura 6.6.

prin $a=b=1$. Propagate (f,D) necesită Justify $(g,\emptyset)$ și Propagate (h,D). Rezolvăm Justify $(g,\emptyset)$ prin alegerea unei intrări a porții g – să zicem c – și punând-o la $\emptyset$. Propagate (h,D) conduce la Justify $(i,1)$ care conduce la $e=\emptyset$. Acum eroarea ajunge la LPE j.

Figura 6.6 (b) arată valorile rezultante. Testul generat este 11Ø×Ø (d poate fi arbitrar Ø sau 1) □

Este important de observat că într-un circuit liber de ramificații fiecare problemă de justificare a unei linii poate fi rezolvată independent de toate celelalte, deoarece mulțimile de LPI ce sunt eventual poziționate (setate) pentru ca prin respectivele atribuiri să se satisfacă justificarea unor linii diferite sunt mutual disjuncte unele față de celelalte.

- Circuite cu ramificații

Considerăm în continuare cazul general al circuitelor cu ramificații în comparație cu cazul particular anterior examinat. În principiu trebuie să atingem aceleași scopuri de bază: activarea defectului și propagarea erorii. Dacă nu activarea defectului se reduce într-o problemă de justificare a unei linii. O primă diferență cauzată de ramificații este aceea că acum pot fi mai multe căi de propagare a unei erori către o LPE. Dar o dată aleasă o cale reducem din nou problema propagării erorii la un set de probleme de tipul justificare-linie.

*** Dificultatea fundamentală cauzată de ramificațiile reconvergente este aceea că, în general, problemele de tipul justificare a liniilor ce rezultă nu mai sunt independente.

Exemplul 6.2 Să considerăm circuitul redundant din figura 6.7 și defectul G_1 b-l-1. Pentru a activa defectul trebuie să justificăm $G_1=Ø$. Acum avem alegerea propagării erorii printr-o cale ce trece prin G_5 sau prin G_6. Să presupunem că am decis să alegem prima variantă. Atunci avem de justificat $G_2=1$. Setul rezultant de probleme - Justify $(G_1, Ø)$ și Justify $(G_2, 1)$ - nu pot fi satisfăcut simultan. Aceasta demonstrează că propagarea erorii prin G_5 a fost greșită. Nu ne rămâne decât să încercăm decizia celelalte: propagarea erorii prin G_6. Aceasta necesită

$G_4=1$, care este eventual satisfăcut de $c=1$ și $c=0$.
Testul rezultant este $111x\emptyset$ □

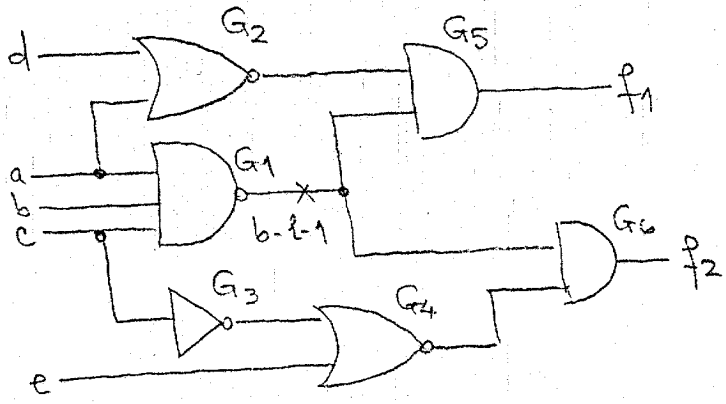

Figura 6.7

Acest exemplu arată că este necesar să se exploreze alternative pentru propagarea erorii. Similar, putem avea de încercat variante diferite pentru justificarea liniilor.

Exemplul 6.3 Să considerăm defectul h b-1-1 din circuitul desenat în Figura 6.8. Pentru a activa acest defect $h=\emptyset$. Există o unică cale de propagare a erorii, și anume prin p și S. Pentru aceasta avem nevoie de: $e=f=1$ și $g=x=1$. Valoarea $g=1$ poate fi justificată de $l=1$ sau de $k=1$. Întâi să încercăm $l=1$. Aceasta conduce la $c=d=1$. Totuși aceste două atribuiri împreună cu cea anterior specificată $e=1$, ar implica $h=\emptyset$ ceea ce conduce la o inconsistență. În consecință justificarea $g=1$ prin $l=1$ a fost incorectă. Deci avem de ales alternativa $k=1$ care implică $a=b=1$. Acum rămâne numai problema de justificare a liniei $x=1$. Ori $m=1$ ori $n=1$ conduce la o soluție consistentă □

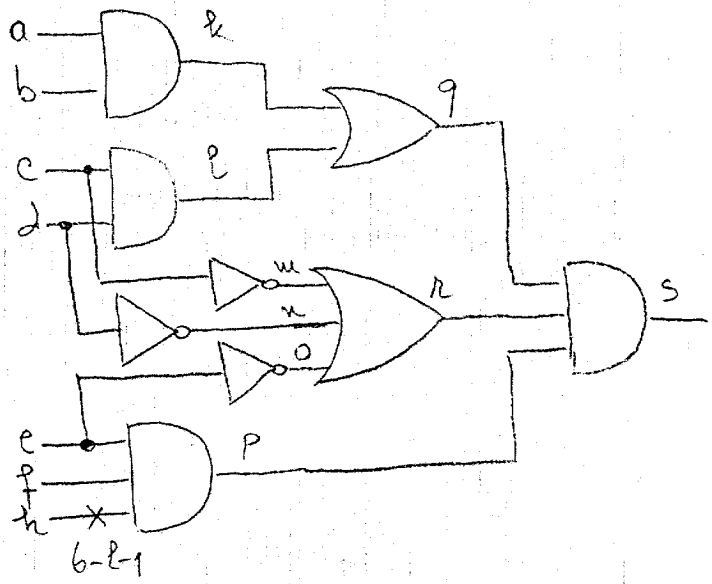

Figura 6.8

- Backtracking

S-a putut remarca faptul că în căutarea unei soluții de găsire a unui test apare un proces de decizie. Ori de câte ori există anumite alternative de justificare a unei linii sau de propagare a unei erori, alegem o alternativă și o încercăm. Dar procedând astfel putem alege o decizie care să conducă la o inconsistență (se folosește deseori termenul de contradicție sau conflict). În consecință în căutarea noastră după un test trebuie să folosim o strategie de backtracking, care să ne permită o explorare sistematică a spațiului complet al tuturor soluțiilor posibile și să putem recupera după decizii incorecte. Recuperarea implică restaurarea stării de calcul din starea dinaintea deciziei incorecte.

În mod uzual atribuirile realizate ca urmare a unei decizii determină unic (implică) alte valori. Procesul de calcul al acestor valori și verificarea consistenței acestora cu valori anterior determinate se numește implicație.

Figura 6.9 arată modul în care au progresat valorile calculate pentru exemplul 6.3, făcând distincție între valorile rezultate din decizii și cele generate prin implicație. Implicațiile inițiale urmează din soluția unică a activării defectului și din problemele de propagare a erorii.

În majoritatea algoritmilor de backtracking trebuie să înregistrăm toate valorile atribuite ca rezultat al unei decizii pentru a avea posibilitatea ștergerii acestora deîndată ce decizia a condus la inconsistență.

În exemplul 6.3 toate valorile ce rezultă din decizia $l=1$ sunt șterse atunci când are loc backtracking-ul.

Decizii	Implicații	Observații
	$h=D'$ $e=1$ $f=1$ $p=D'$ $R=1$ $q=1$ $o=\emptyset$ $s=D'$	Implicații inițiale
$l=1$		Pt. a justifica $q=1$
	$c=1$ $d=1$ $m=\emptyset$ $n=\emptyset$ $r=\emptyset$	Contradicție!
$k=1$	$a=1$ $b=1$	Pt. a justifica $q=1$
$m=1$	$c=0$ $l=0$	Pt. a justifica $R=1$

Figura 6.9. Deciziile și implicațiile pentru exemplul 6.3.

Figura 6.10 schițează o schemă de algoritm GTD cu backtracking. Problema inițială de rezolvat în generarea unui test pentru defectul l s-a-v constă din justificarea valorii v' pe linia l și propagarea erorii din linia l la o LPE. Ideia de bază este că nepotând rezolva problema direct, transformăm recursiv problema în subprobleme și încercăm să rezolvăm întâi aceste subprobleme. Soluționarea unei probleme poate avea drept rezultat SUCCES sau INSUCCES.

```
Solve(.)
begin
    if Implică_și_verifică() = INSUCCES then return INSUCCES;
    if (eroare la LPE and toate liniile sunt justificate)
            then return SUCCES;
    if (nici o eroare nu poate fi propagată la o LPE)
            then return INSUCCES;
    alege o problemă nerezolvată;
    repeat
        begin
            alege o cale neîncercată de soluționare a problemei;
            if Solve() = SUCCES then return SUCCES;
        end
    until toate căile de rezolvare ale problemei au fost încercate;
    return INSUCCES;
end
```

Figura 6.10 Schiță generală a unui algoritm de GTD cu backtracking.

Întâi algoritmul tratează toate problemele ce au soluții unice și pot fi soluționate, în consecință, prin implicație. Este cazul procedurii Implică-și-verifică, care face deasemenea o verificare a consistenței. Procedura Solve() raportează INSUCCES dacă verificarea consistenței falizează; în schimb se raportează SUCCES atunci când se atinge scopul dorit: anume atunci când s-a propagat o eroare la o LPE și când toate problemele de justificare a liniilor au fost soluționate. Chiar și într-o stare consistentă algoritmul poate determina că nici o eroare nu poate fi propagată mai departe către o LPE și atunci nu are nici un rost să se continue căutarea raportându-se INSUCCES.

Dacă Solve nu poate determina imediat SUCCES sau INSUCCES atunci alege o problemă curent nerezolvată. Aceasta poate fi ori o justificare de linie ori o problemă de propagare a unei erori. În acest punct există anumite căi alternative de soluționare a problemei alese. Algoritmul selectionează una dintre căi și încearcă să rezolve problema la următorul nivel de recursivitate. Procesul continuă până când se găsește o soluție sau toate alegerile posibile au faliat. Dacă activarea inițială a lui Solve faliază atunci algoritmul încetează să genereze un test pentru defectul specificat.

Alegerea unei probleme nerezolvate pentru soluționare și selecția unei căi nencercate de soluționare pot fi — în principiu — arbitrare. Adică, dacă defectul este detectabil vom genera un test pentru respectivul defect independent de ordinea în care problemele și soluțiile sunt abordate. Totuși procesul de alegere poate afecta în mare măsură eficiența algoritmului. Există câteva GTD ai căror algoritmi au structura euristică a lui Solve.