ASUPRA PERFORMANȚELOR ALGORITMILOR DE SIMULARE A DEFECTELOR CIRCUITELOR LOGICE

Corneliu Popescu
corneliu_popescu@uoradea.ro

Daniela E. Popescu
depopescu@uoradea.ro

Adrian Nagy
adinagy@rdslink.ro

Rezumat: În această lucrare, se prezintă rezultatele comparative ale analizei de performanță pentru algoritmii de simulare a defectelor. Se consideră cei trei algoritmi de simulare a defectelor: simularea paralelă, simularea deductivă şi simularea concurentă. Această analiză reprezintă rezultatul fazei de validare a datelor în cadrul unui proiect mai amplu, finanțat prin programul INFOSOC. Titlul proiectului este: *Implementarea unui mediu de programare cu facilități de testare*. Elaborarea mediului integrat TDL („*Testing Digital Logic*") a presupus realizarea unui suport software capabil să satisfacă atât necesitățile de generare optimă a vectorilor stimuli de test, cât şi pe cele de simulare optimă, respectiv de evaluare a testabilității în vederea intervenției la nivelul structurii pentru creşterea testabilității acesteia dacă este cazul.

Modulele program ce implementează mediul TDL includ module program ce realizează mediul de simulare a defectelor circuitelor logice. Această lucrare prezintă modul de implementare a modulelor program, destinate simulării logice a defectelor, precum şi rezultatul analizei de viteză şi consum de resurse pentru cele trei module de simulare logică a defectelor implementate în cadrul mediului TDL.

Cuvinte cheie: testare, vectori de test, simulare a defectelor.

1. Introducere în problematica generării testelor

Procesul de generare a testelor include modelarea şi reducerea defectelor, generarea *pattern*-urilor de test, evaluarea gradului de acoperire a defectelor şi producerea dicționarului de defecte [2]. Procedura de generare a testelor este reprezentată în figura 1.

Primul pas constă în dezvoltarea unui *dicționar de defecte* pentru circuit (adică, în modelarea defectelor presupuse) şi în reducerea numărului de defecte în termenii relației de echivalență între defecte. În general, se adoptă modelul defectelor singulare s-a-1 (0) şi dicționarul de defecte se generează direct din descrierea logică a circuitului, printr-o aranjare tabelară a defectelor distinctibile.

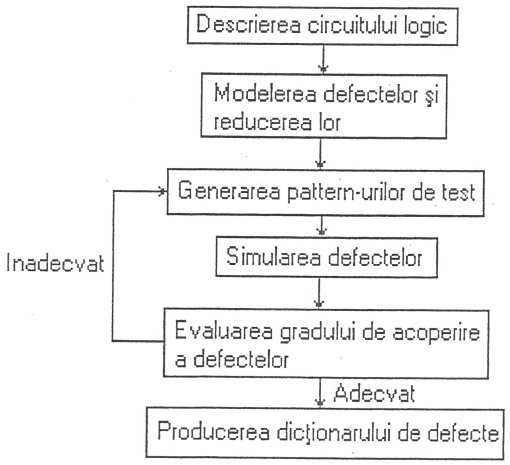

Figura 1. Procedura pentru generare a testelor

Pe urmă, se generează *pattern*-urile de test pentru testarea defectelor din dicționarul de defecte. Pe urmă, *pattern*-urile de test sunt simulate pe circuitele defectate, conform defectelor din dicționar, şi se evaluează, pe baza rezultatelor simulării (ce include liste de defecte testate şi netestate), gradul de acoperire a defectelor. Dacă acoperirea de defecte obținută nu convine, se reia procesul de generare a *pattern*-urilor de test pentru defectele netestate, până se obține o acoperire corespunzătoare a defectelor. În final, se completează dicționarul de defecte, specificându-se informații suficiente pentru detecția 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 colecții de aplicații software, de genul celor dezvoltate în cadrul aplicației TDL, dezvoltată în cadrul programului INFOSOC. Pentru a asigura un grad ridicat de acoperire a defectelor, la un preț de calcul redus, există o interacțiune 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 eficiente.

Simularea defectelor este o parte importantă a procesului de generare a testelor, fiind utilizată în scopul generării dicționarelor de defecte și pentru verificarea vectorilor stimuli de test, necesari detecției și localizării defectelor logice. De asemenea, simularea defectelor este folosită și pentru determinarea acoperirii defectelor pentru un test dat, adică pentru găsirea tuturor defectelor detectate de un test dat.

În general, simulatoarele se pot împărți în două categorii: simulatoare realizate prin compilare și simulatoare bazate pe tabele și direcționate de evenimente. Ambele tehnici de simulare sunt de tip soft. Marele dezavantaj al acestora este că, pentru circuite mari, sunt consumatoare de timp și resurse. Scopul nostru este de a compara eficiența celor trei metode de simulare: paralelă, deductivă și concurentă.

2. Noțiuni teoretice legate de simularea defectelor circuitelor logice

În continuare, vom prezenta, pe scurt, cei trei algoritmi:

1. ***Simularea paralelă*** este o metodă veche și acceptată pe scară largă în automatizarea sistemelor de generare s testelor [3]. Această metodă se bazează pe proprietatea de procesare paralelă a operațiilor pe biți. Toate calculatoarele au mai multe instrucțiuni orientate pe biți, cum ar fi AND, OR, XOR, NOT. În timpul executării acestor instrucțiuni, biții unui cuvânt sunt manipulați în mod identic și independent. Instrucțiunile operează pe fiecare bit în parte, fiecare bit al operanzilor fiind evaluat simultan. Marele avantaj al utilizării acestor operații este faptul că putem optimiza consumul de memorie, memorând mai mulți termeni într-un cuvânt. Alt avantaj, la fel de important, este posibilitatea calculului paralel asupra acestor cuvinte. Dacă folosim un cuvânt pe n biți pentru simulare, vom putea procesa n probleme diferite în paralel.

   Dacă în circuitul nostru avem m defecte care trebuie simulate, atunci numărul de treceri este întregul mai mare sau egal decât m/n. Metoda se bazează pe o procedură de injecție a defectelor, care injectează efectul logic al defectului. Acest proces de injecție se face pentru fiecare linie de semnal S care se simulează și constă în asocierea a două măști mask(S) și fvalue(S). Fiecărui defect injectat îi corespunde o poziție de bit unică în măștile mask(S) și fvalue(S). Această poziție de bit trebuie să fie diferită de pozițiile de bit din măștile corespondente a celorlalte defecte rămase, ce se simulează în paralel. Valorile măștilor sunt definite astfel: mask(S) este 1 în caz de defect și 0 în rest, iar fvalue(S) dă valoarea defectului, 0 pentru s-a-0 și 1 pentru s-a-1. Aceste două măști sunt utilizate pentru injectarea defectelor în fiecare bit al cuvântului S și se obține un cuvânt mască S' folosind ecuația:

   $$S' = S \cdot \overline{Mask} + Mask \cdot FValue \qquad (1)$$

   unde $\cdot, +, \overline{\ }$ sunt operațiile logice AND, OR și NOT.

   După injectarea defectelor pentru fiecare linie de semnal, circuitul va fi simulat pentru cuvântul de n biți. Primul bit este rezervat pentru funcționarea corectă a circuitului. După simulare, se vor compara biții cuvântului obținut cu primul bit. Dacă un bit are valoare diferită de primul bit, atunci defectul corespunzător poate fi detectat și localizat.

2. **Metoda *simulării deductive*** a defectelor constă în simularea explicită a comportării circuitului fără defect și deducerea simultană din starea curentă bună a circuitului, a tuturor defectelor detectabile pe orice linie internă sau de ieșire, în timpul stării curente. Folosind această metodă, putem calcula toate defectele detectabile în același moment pentru fiecare vector de test aplicat, fiind nevoie doar de un singur pas de simulare.

   Simulatorul deductiv simulează circuitul fără defecte și calculează liste de defecte. O listă de defecte L_A asociată cu linia A conține numele sau indexul fiecărui defect ce produce o eroare pe linia respectivă, când circuitul este în starea logică curentă. Adică, fiecare defect din listă, inserat singur în circuit, ar cauza complementarea stării bune a semnalului de linie asociat. Listele de defecte sunt propagate prin circuit de la intrări până la ieșiri, generându-se o listă de defecte pentru fiecare semnal de linie. Aceste liste se actualizează când este necesar, la fiecare schimbare în starea logică a circuitului.

   Simulatorul deductiv folosește o procedură direcționată de evenimente; ieșirea de stare a unui element este calculată numai când apare un eveniment la una din intrările sale. În cazul simulării deductive, există două tipuri de evenimente: evenimente logice, ce apar la schimbarea valorii logice a unui semnal de linie, și evenimente listă, ce apar când se schimbă o listă de defecte. Simulatorul va trebui să recalculeze lista de defecte asociată ieșirii unui element chiar dacă nu a apărut nici un eveniment logic la intrări.

3. **Metoda *simulării concurente*** a defectelor este o metodă care realizează o singură trecere de simulare pentru calculul tuturor defectelor detectabile la fiecare vector de test aplicat, în mod similar metodei deductive. De obicei, comportarea unui circuit defect este foarte puțin diferită de comportarea circuitului

fără defect [6]. Semnalele în circuitul defect şi în cel corect funcţional sunt uneori identice, aproape identice în majoritatea timpului şi doar rareori diferă substanţial. Simularea concurentă, care se bazează pe asemănările dintre circuitul defect şi cel fără defect, constă în simularea circuitului fără defect şi simularea concurentă a circuitului cu defect, numai când activitatea celor două diferă.

Se pot observa anumite similitudini între metoda de simulare deductivă şi cea concurentă. Dar există şi câteva deosebiri, cele mai importante fiind faptul că lista de defecte a simulării deductive este un subset al listei de defecte calculate de metoda concurentă şi faptul că simularea concurentă necesită resurse (memorie) mai multe decât simulare deductivă. La simularea concurentă atât circuitul fără defect, cât şi circuitul cu defect sunt simulate explicit; pe când la simularea deductivă, doar circuitul fără defect era simulat explicit, circuitul cu defect fiind simulat deductiv. O altă diferenţă între metoda de simulare deductivă şi cea concurentă este faptul că simulatorul concurent procesează doar circuitele active. La simularea deductivă, dacă apare un eveniment listă, este necesar să se reevalueze toate defectele din listă (datorită procesului complex al operaţiilor cu mulţimi). În contrast, simulatorul concurent nu trebuie să reevalueze un defect a cărui stare logică nu s-a modificat, trebuind resimulat doar circuitul defect activ (a cărui stare logică s-a modificat).

Marele dezavantaj al simulării concurente a defectelor este consumul mare de memorie, mai mare şi decât consumul în cazul simulării deductive. O altă problemă este faptul că nu se poate estima cantitatea de memorie necesară înaintea rulării; aceasta implicând probleme mari în cazul circuitelor cu multe porţi.

3. Implementarea algoritmilor şi rezultatele analizelor

Scopul proiectului INFOSOC, din care face parte această simulare, este dezvoltarea unui mediu care să permită descrierea circuitelor, vizualizarea grafică a acestora, simularea circuitelor, determinarea toleranţei la defecte, generarea vectorilor de test, testarea acoperirii acestor vectori de test, generarea de dicţionare de defecte.

Metodele de simulare a defectelor descrise mai sus (simulare paralelă, deductivă şi concurentă) sunt parte integrată în acest program. Având de-a face cu un proiect de o anvergură mare, structura de date necesară descrierii circuitelor este destul de mare. Fiecare element al circuitului are o structură de date care să-l descrie complet pentru toate operaţiile prevăzute în proiect: descriere grafică, simulare etc. Astfel, consumul de memorie este destul de ridicat doar în momentul descrierii circuitelor. Pentru micşorarea acestui consum, în momentul începerii simulărilor, se transferă doar un set minimal de date pentru fiecare componentă:

```
struct comp_sim
{
  int type;
  unsigned int inputs;
  unsigned int outputs;
  int in[9];
  bool *val_int[9];
  bool calculat[9];
  int out[9];
};
```

După cum se observă, numărul de intrări al unei componente şi numărul de componente care se pot lega la ieşire este limitat. Această structură minimală a elementelor este suficientă pentru a descrie complet circuitul. Preluarea datelor este o procedură (inclusă în algoritm) identică pentru cei trei algoritmi.

După preluarea datelor, algoritmul de simulare paralel face o estimare a numărului de biţi necesari simulării circuitului: $nr\_biti = \det ect\_nr\_\text{int}\, rari \cdot 2 + 1$, aceasta dându-ne valoarea descrisă cu m în partea teoretică. În nod normal, am folosi cuvinte cu $n = 32$ de biţi şi ar fi necesare m/n treceri pentru o simulare completă. Însă, acest lucru necesită o împărţire pe cuvinte şi poziţionarea fiecărui bit la locul potrivit prin deplasări la stânga a operandului. S-a observat că acest proces de pregătire necesită un timp destul de îndelungat. La aceasta se adaugă timpii necesari prelucrării datelor finale, evaluării fiecărui bit prin comparaţie cu cel mai semnificativ bit (rezervat simulării funcţionării corecte a circuitului). Aceste două operaţii trebuie efectuate pentru fiecare trecere. Din acest motiv, se va folosi un singur cuvânt (şir de biţi) cu o lungime variabilă care va necesita o singură prelucrare pregătitoare şi una la final pentru interpretarea rezultatelor. Lungimea acestui şir va fi egală cu

m. Pentru a păstra caracterul paralel al algoritmului, şirul va fi format din valori booleene; operaţiile efectuându-se concomitent pe toţi biţii.

Pentru fiecare poartă, se vor calcula măştile *mask(S)* şi *fvalue(S)*, după aceasta se aplică ecuaţia (1). Măştile *mask(S)* şi *fvalue(S)*, fiind evaluate doar la simularea anumitei porţi, sunt alocate dinamic, iar după efectuarea injectării defectelor resursele ocupate de măşti sunt eliberate.

Transformarea în şir de biţi, prezentată anterior, mai are un avantaj şi anume faptul că algoritmul va avea o singură trecere [5]. Astfel, am mărit consumul de memorie pentru o trecere, dar am redus numărul de treceri.

Implementarea algoritmilor de simulare deductiv şi concurent este similară. Ambele metode au la bază operaţiile cu mulţimi. Acestea sunt relativ greu de implementat în comparaţie cu operaţiile pe biţi din cazul algoritmului de simulare paralel. Astfel, am definit operaţiile pe mulţimi necesare:

```
Adun(multime *unu, multime *doi );
Scad(multime *unu, multime *doi );
Reuniune(multime *unu, multime *doi );
Intersectie(multime *unu, multime *doi );
```

Aceste mulţimi conţin indexurile defectelor. Defectele sunt numerotate de la *1* la n. Pentru fiecare linie de semnal, care este o intrare sau ieşire dintr-o poartă, se va asocia o astfel de mulţime, o listă a defectelor. Circuitul este descris prin porţi, astfel liniile vor fi considerate intrările acestor porţi. Elementele care descriu ieşirile şi intrările din circuit sunt şi ele considerate porţi.

Porţile sunt numerotate în momentul descrierii circuitului. Această numerotare nu corespunde neapărat ordinii de parcurgere a circuitului, ci ordinii de declarare a fiecărui element al circuitului. Pentru ca ordinea de parcurgere şi cea de declarare a circuitului să corespundă, este necesară efectuarea unei etape separate de renumerotare, de nivelare ierarhică a circuitului. Această etapă a fost eliminată deoarece ordinea de parcurgere a circuitului este de la intrări spre ieşiri, prin fiecare poartă care are toate intrările calculate, independent de numerotarea porţilor. Totuşi, nivelarea ierarhică a circuitului ar simplifica algoritmul, această etapă de prelucrare fiind luată în considerare pentru îmbunătăţirea performanţelor pentru o versiune ulterioară.

Având în vedere faptul că simularea concurentă poate fi uşor implementată prin tehnica *table lookup* şi faptul că lista de defecte a simulării deductive este un subset al listei de defecte calculate de metoda concurentă, am încercat o abordare similară pentru ambele metode. Fiecare poartă este descrisă printr-o ecuaţie care defineşte ieşirea în funcţie de intrări. Aceste ecuaţii au ca operanzi mulţimi, acestea fiind chiar listele de defecte. Ecuaţiile sunt „memorate" într-un tabel. Implementarea acestui tabel s-a făcut printr-o structură de tip *case*:

```
case poarta of
  "and": ecuatia1;
  "or" : ecuatia2;
  . . .
 else
   tratarea erorilor de program;
end;
```

Astfel, tabelul este inclus în program prin *hardcoding*.

În momentul apariţiei unui eveniment listă sau logic (modificarea unei valori logice), la simularea deductivă se face o nouă parcurgere, iar la cea concurentă se vor reevalua listele asociate elementului ce a generat evenimentul.

Vectorii de test, folosiţi pentru simulări, pot fi generaţi automat de către program pentru o căutare exhaustivă sau se poate face simularea doar pentru un singur vector stimuli de test, introdus de utilizator. La căutarea exhaustivă, vectorul de test este determinat prin alegerea tuturor combinaţiilor biţilor de intrare (2^n combinaţii pentru n intrări). De exemplu, pentru un sumator pe 1 bit, avem două porţi de intrare şi un bit de transport, adică trei posibile intrări, rezultând un număr de 2^3 vectori de test. Astfel, vom avea 8 vectori de test, circuitul fiind simulat pentru fiecare vector în parte.

Afişarea datelor se face în mod text fiind suficient de sugestivă. Pentru o versiune ulterioară, se poate trece la descrierea grafică a rezultatelor.

Rezultatele au fost obținute, rulând programele de simulare, pe o platformă cu procesor Pentium 4 (1,7 GHz), cu memorie de 256 Mb RAM şi sistem de operare Windows 2000 (programul poate fi rulat doar pe sisteme de operare Windows pe 32 de biți).

Pentru a obține rezultate necesare analizei performanțelor, s-a pornit de la circuitele sumatoare. Cel mai simplu circuit sumator este cel prezentat în figura 2, el conținând doar 5 porți logice plus porțile asociate intrărilor şi ieşirilor [1]. Prin multiplicarea acestei structuri, s-au obținut sumatoare de tip Ripple Carry Adders de 2, 4, 8, 16, 32, 64 biți pentru care datele încep să fie concludente [4].

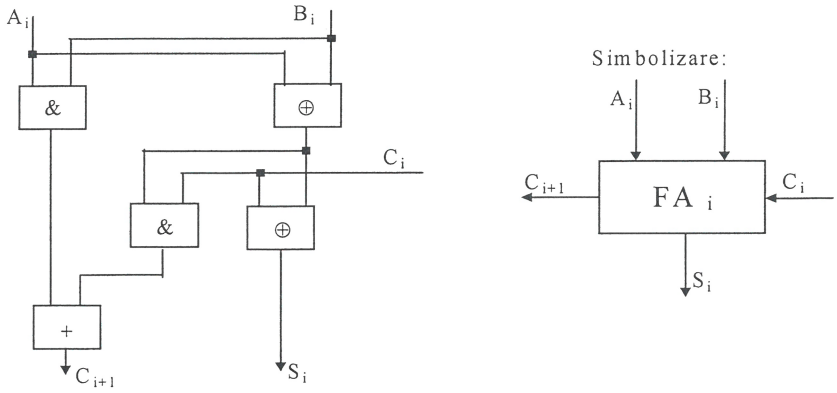

Figura 2.

În Tabelul 1, sunt prezentate rezultatele obținute pentru un vector de test introdus de către utilizator (un singur vector ales aleator):

Tabelul 1

	Simulare Paralelă		Simulare Deductivă		Simulare Concurentă	
Circuit	Timp de Simulare	Memorie Consumată	Timp de Simulare	Memorie Consumată	Timp de Simulare	Memorie Consumată
Sumator pe 1 bit	1,2s	10kb	2,4s	16kb	3,6s	30kb
Sumator pe 2 biți	3,1s	18kb	5,6s	40kb	8,2s	76kb
Sumator pe 4 biți	9s	33kb	11,4s	96kb	18s	182kb
Sumator pe 8 biți	24,1s	71kb	24,4s	194kb	38,3s	372kb
Sumator pe 16 biți	54,4s	156kb	53s	412kb	82s	792kb
Sumator pe 32 biți	136,3s	320kb	130,2s	850kb	181,2s	1666kb
Sumator pe 64 biți	372s	652kb	296s	1708kb	377,6s	3420kb

Analizând rezultatele din Tabelul 1, rezultă că simularea paralelă a defectelor este mult mai convenabilă la număr mic de porți atât sub aspectul timpului de simulare, cât şi sub aspectul consumului de memorie, în raport cu celelalte tipuri de simulări: deductivă şi concurentă. Pentru un număr de porți de valoare aproximativă 150, este mai performantă simularea deductivă a defectelor.

În Tabelul 2, sunt prezentate rezultatele obținute în cazul testării exhaustive (numărul vectorilor de test este egal cu 2^n unde n este numărul de intrări în circuit):

Tabelul 2

	Simulare Paralelă		Simulare Deductivă		Simulare Concurentă	
Circuit	Timp de Simulare	Memorie Consumată	Timp de Simulare	Memorie Consumată	Timp de Simulare	Memorie Consumată
Sumator pe 1 bit	10,1 s	12kb	10.4s	19kb	12,6s	32kb
Sumator pe 2 biți	102.3s	21kb	73,1s	45kb	85,6s	79kb
Sumator pe 4 biți	48min	36kb	32min	106kb	30min	186kb
Sumator pe 8 biți	6h 54min	75kb	2h 10min	212kb	1h 56min	378kb
Sumator pe 16 biți	26h	161kb	6h 25min	433kb	4h 40min	808kb
Sumator pe 32 biți	100h	328kb	11h	892kb	8h	1702kb
Sumator pe 64 biți	16 zile	665kb	24h	1824kb	14h	3492kb

Valorile pentru consumul de memorie, prezentate în acest tabel, reprezintă valori maxime. Datorită alocării dinamice, memoria este refolosită pe cât posibil, consumul oscilând după necesități. Acest consum de memorie este măsurat cu un program independent de simulator, valorile astfel obținute fiind reale, cele sesizate de către sistem.

Valorile de timp reprezintă diferența dintre momentul începerii algoritmului şi cea a finalizării lui, acest interval incluzând timpii necesari pregătirii datelor şi generării vectorilor de test.

Prin urmare, cu cât volumul datelor cu care se face simularea este mai mare, cu atât devine mai convenabilă – din punctul de vedere al timpului implicat de procesul de testare – simularea concurentă a defectelor.

4. Concluzii

Pentru circuitele mici, cea mai bună metodă de simulare (cea mai rapidă şi consum de memorie mai mic) s-a dovedit a fi metoda de simulare paralelă. Când circuitele conțin mai multe porți, acest algoritm tinde să încetinească. De asemenea, se poate observa faptul că simularea paralelă necesită un spațiu de memorie mai mic decât celelalte două metode de simulare. Pentru circuitele mai mari, se observă o eficiență mai mare la simularea deductivă şi concurentă, raportul viteză – consum de memorie fiind mai bun. De asemenea, se observă eficiența algoritmului deductiv şi, mai ales, al celui concurent în cazul căutării exhaustive (recalculări puține în cazul evenimentelor listă şi logice).

Bibliografie

1. **OMONDI, A. R.**: Computer Arithmetic Systems Algorithms, Architecture and Implementation, Prentice Hall International, 1994, 430p.
2. **VLĂDUȚIU, M., CRIŞAN, M.:** Tehnica testării echipamentelor automate de prelucrare a datelor, Editura Facla, Timişoara 1989, 345p.
3. **ABRAMOVICI, M., BREUER, M.A., FRIEDMAN, A.D.:** Digital Systems and Testable Design, Computer Science Press, 1996, 620p.
4. **JOHN P. HAYES**: Computer Architecture and Organization, McGraw-Hill Book Company.
5. **POPESCU, D.E.:** Testarea circuitelor digitale, Editura Universității din Oradea, 2003, 235p.
6. **FUJIWARA H.:** Logic Testing and Design for Testability, Computer Systems Series The MIT Press, 1996, 360p.