



UNIUNEA EUROPEANĂ



GUVERNUL ROMÂNIEI



Instrumente Structurale
2007-2013



Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Sisteme Tolerante la Defecte

23. Toleranța la defecte aplicată circuitelor VLSI

TOLERANȚA LA DEFECTE APLICATĂ CIRCUITELOR VLSI

Toate structurile resiliente descrise prin modul de interconectare al acestora pot fi aplicate unei palete foarte largi de module, începând cu cele mai simple module logice combinaționale și până la cele mai complexe microprocesoare ori chiar plăci complete cu microprocesoare. Duplicarea completă a unor microprocesoare, care nu sunt proiectate să fie utilizate în aplicații critice, poate impune un efort complex, posibil prohibitiv de costisitor care, adesea, se dovedește nejustificat.

Pentru astfel de situații s-au dezvoltat modalități mai simple necesitând dotări suplimentare mai puțin costisitoare. Aceste tehnici se bazează pe ipoteza că un astfel procesor execută aplicații curente iar în cazul apariției unei erori aplicația, ori o parte din aceasta, poate fi reluată și re-executată dacă sunt îndeplinite două condiții esențiale:

- (1) eroarea este detectată iar cauza erorii este un defect de scurtă durată, tranzitoriu,
- (2) atunci când se reia execuția aplicației este foarte probabilă dispariția erorii.

Simularea defectelor, experimente de injectare a defectelor și analiza log-urilor de erori ale sistemelor operaționale au arătat că erorile, în mare lor majoritate, sunt provocate de defecte tranzitorii.

Tabelul 1.

Benchmark	Efectele iradierii cu ioni grei [KGT89]					
	permanent	Control flux temporar	latent	Control	Date	Altele
<i>Quicksort</i>	62%	8%	16%	8%	4%	2%

Defectele tranzitorii din microprocesoare afectează mai cu seamă controlul fluxului de date și instrucțiuni. Din aceste rațiuni, operarea tolerantă a defectelor în sistemele digitale impune tehnici de abordare concurente ale erorilor.

Clasificarea erorilor din sistemele cu microprocesoare

Investigarea tipurilor de erori ca și a efectelor acestora în sistemele cu microprocesoare constituie un curent constant de cercetare. Pe măsură ce crește complexitatea și viteza dispozitivelor digitale se dovedește necesară considerarea a noi defecte și tipuri de erori. În marea majoritate a cazurilor, complexitatea dispozitivelor sau incompleta cunoaștere a structurii interne ori chiar a operării constituie obstacole în constituirea modelelor defectelor la nivel coborât (nivelul porților ori nivelul transferului la nivel de registre).

Tabelul 2.

	Efectele injectării defectelor hardware asupra magistralei sistemului						
	Execuție invalidă program	Adresa codului operației	Adresă memorie invalidă	Adresă citire invalidă	Cod operație inexistent	Adresă scriere invalidă	Memorie inexistentă
[STDM82]	63%	59%	56%	44%	37%	19%	8%
[MQS90]		45%	72%		1%	9%	

Astfel, este dificil de stabilit o estimare analitică a consecințelor defectelor de nivel inferior asupra comportamentului sistemelor, nivel superior. Toate considerații sunt

aplicabile, cu precădere, defectelor tranzitorii deoarece această clasă de defecte nu a dobândit o deplină caracterizare.

Atunci când se cunoaște structura dispozitivului investigat se poate apela la tehnici bazate pe simulare. Această abordare s-a dovedit deosebit fructuoasă atunci când s-a aplicat în faza de proiectare a dispozitivelor digitale. În faza prototipului dispozitivului respectiv cele mai frecvente evaluări au la bază tehnici de injectare a defectelor. O altă sursă de analize sunt fișierele de erori, log-urile de sistem, din faza operațională a dispozitivelor digitale.

Tabelul 3.

Efectele simulării injectării defectelor în procesoarele RISC [ORG92].

	Erori controlul fluxului	Erori Date	Alte erori	Erori magistrale	Erori adresare	Erori de bază
<i>Efecte</i>	33%	60%	7%	28%	8%	37%

Câteva din rezultatele cunoscute ale cercetărilor asupra erorilor din microprocesoare sunt prezentate în tabelele 1, 2 și 3. Sunt considerate erorile relativ ușor de detectat din microprocesoare.

Tehnicile întrebuințate sunt, între altele, simularea [ORG92], injectarea defectelor prin iradiere cu ioni grei [KGT89], perturbațiile de putere și modificarea semnalelor de pe magistralele sistemelor [MKS90], [STDM82].

Cel mai important rezultat al acestor studii este faptul că defectele tranzitorii și intermitente constituie principala cauză a defectării sistemelor digitale. Se estimează că aceste categorii de defecte apar cu o frecvență de 10 până la de 30 de ori mai mare decât defectele permanente. Aceasta conduce la concluzia că 90% din defectele calculatoarelor sunt tranzitorii și intermitente [SL81].

Defectele tranzitorii pot fi cauzate în primul rând de zgomotul sursei de alimentare (variații ale frecvenței de comutație), interferențe electro-magnetice (rețele de comunicații fără fir și altele), ionizări ale razelor cosmice ori particule alfa din materialele utilizate pentru încapsularea circuitelor.

Într-o mai mică măsură, defectele intermitente, pot fi cauzate de erori ale proiectării. Printre acestea sunt menționate câteva deficiențe tipice, cum ar fi:

- Depășirea numărului liniilor de intrare în anumite porți,
- Circuitele funcționând cu semnale marginale,
- Oscilațiile circuitelor cu stări (oscilații între stările active și cele inactive) etc.

O concluzie esențială a acestor studii a fost determinarea celor mai eficiente mecanisme de detecție a erorilor.

Printre acestea, în afară de verificarea datelor, sunt:

- verificarea fluxului de comenzi (controlul),
- verificarea modului de adresare al memoriei și
- verificarea codurilor instrucțiunilor.

Este de reținut că, verificarea codurilor instrucțiunilor este eficientă, mai cu seamă, în cazul în care nu sunt utilizate toate codurile posibile pentru codificarea instrucțiunilor procesorului.

Tehnici de detecție concurrentă a erorilor

Așa cum s-a dovedit pe cale experimentală, cea mai mare parte a defectelor sunt defecte tranzitorii. Acest tip de defecte sunt dificil de evitat prin proiectare și prin testare pe durata procesului de manufacturare.

Testele clasice planificate sistematic, în afara rulaajului normal al unui microprocesor, sunt consumatoare importante de timp și nu sunt capabile să detecteze defectele tranzitorii. Sunt de preferat tehnici concurente *on-line* de detecție a erorilor [KK07].

Se dovedește că unica abordare pentru creșterea dependabilității microprocesoarelor sunt tehnicile de toleranță a defectelor. Deoarece dependabilitatea unui sistem este deosebit de sensibilă față de defectele nedetectate ori impropriu tratate [Sos94] aplicarea tehnicilor de detecție eficientă a erorilor este de extrem interes.

Întrucât defectele tranzitorii nu produc defecte permanente în sisteme, șansele de recuperare sunt importante, cu atât mai mult atunci când acoperirea erorilor este mare iar latența este mică (evitându-se răspândirea efectelor erorilor și compromiterea recuperării datelor).

Din punctul de vedere al nivelului la care se face verificarea, detectoarele concurente de erori pot fi clasificate în trei mari clase:

- Circuitul
- Sistemul
- Aplicația

Cea mai simplă tehnică de acest tip mandatează executarea oricărui program de două ori urmând ca utilizarea rezultatelor să fie condiționată de congruența acestora. Această abordare este, adesea, numită *redundanța temporală* și are drept urmare directă reducerea cu 50% a performanței sistemului de calcul. Tehnica aceasta nu impune detectarea erorii.

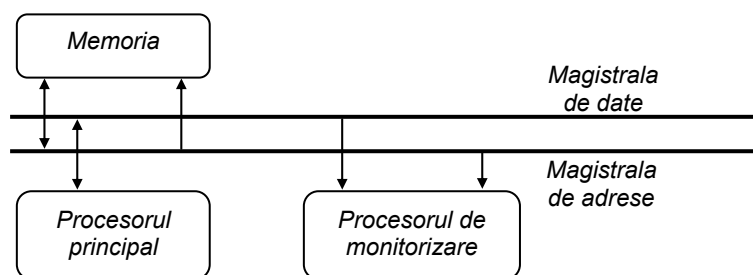


Figura 1. Detecția erorilor prin utilizarea unui procesor de monitorizare.

Dar, dacă este implementabil un mecanism care să detecteze erorile ce pot să apară pe durata execuției unei instrucțiuni, atunci instrucțiunea respectivă trebuie re-executată, preferabil după un anumit timp care să permită dispariția defectului tranzitoriu.

O astfel de abordare care vizează doar re-execuția unei singure instrucțiuni, la un anumit moment, are un impact mult mai mic asupra performanței, comparativ cu re-execuția în întregime a programului.

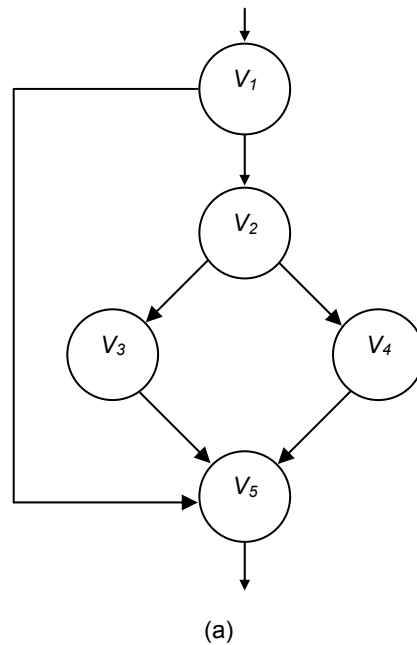
Au fost dezvoltate și alte tehnici, care implică costuri mici, pentru tratarea concurrentă a erorilor.

În acest sens s-a introdus un procesor simplu destinat să monitorizeze fluxul de date și comenzi la nivelul procesorului principal. Un astfel de procesor de monitorizare mai este numit și procesor de veghe (*watchdog*).

Procesorul de monitorizare

Un astfel de procesor realizează concurrent, la nivelul sistemului, detecția erorilor prin monitorizarea magistralelor sistemului care conectează procesorul și memoria (așa cum se poate vedea în figura 1). Această monitorizare se focalizează, în primul rând, asupra verificării controlului fluxului de date. Se verifică, astfel, faptul că procesorul principal execută blocuri corecte de cod, în ordinea corespunzătoare.

Monitorizarea poate detecta erori hardware dar și erori software care să aibă drept urmare fie execuția unor instrucțiuni eronate, fie a unor căi necorespunzătoare prin aplicații.



acceptă semnătura(V_1);
 fie
 acceptă semnătura (V_2);
 fie
 acceptă semnătura (V_3);
 ori
 acceptă semnătura (V_4);
 acceptă semnătura (V_5);
 ori
 acceptă semnătura (V_5);

(b)

acceptă și verifică semnătura (V_1);
 fie
 acceptă și verifică semnătura (V_2);
 fie
 acceptă și verifică semnătura (V_3);
 ori
 acceptă și verifică semnătura (V_4);
 acceptă și verifică semnătura (V_5);
 ori
 acceptă și verifică semnătura (V_5);

(c)

Figura 2.

(a) Graful fluxului de control.

(b) Verificarea fluxului de control.

(c) Verificarea nodurilor și a fluxului de control.

Pentru realizarea monitorizării fluxului de control, este necesar ca procesorul să primească informația care trebuie verificată. Această informație facilitează verificarea în timp real a corectitudinii execuției programelor prin procesorul principal.

Informația care este furnizată procesorului de monitorizare (Figura 2.(a)), este dedusă din *Graful Fluxului de Control*, denumire abreviată *GFC*. Acest graf reprezintă fluxul de control ce trebuie să fie executat de procesorul principal.

Un nod din acest graf reprezintă un bloc compact de instrucțiuni, fără ramificații, din secvența de calcul. Într-un astfel de bloc nu există instrucțiuni de salt spre instrucțiuni din bloc ori spre alte instrucțiuni din afara blocului și nici nu există alte salturi din program care să ajungă la o instrucțiune din blocul considerat.

O ramificație reprezintă un flux de control permis, corespunzând adesea unei instrucțiuni de salt. Etichetele din graf, numite semnături, sunt atribuite nodurilor din *GFC* și sunt stocate în procesorul de monitorizare împreună cu *GFC*.

Pe durata execuției aplicației, semnăturile nodurilor curent aflate în execuție sunt trimise procesorului de monitorizare de la procesorul principal. În această manieră, procesorul de monitorizare poate verifica validitatea căii curente din *GFC* corespunzător.

Programul executat de monitorul de monitorizare pentru *GFC* din figura 2.(a) este cel din figura 2.(b). Acest program de verificare va detecta orice cale invalidă din execuția programului, așa cum ar fi calea $\{V_1, V_3\}$, spre exemplu. Dar, orice eroare dintr-un nod (din secvența corespunzătoare nodului) nu va putea fi identificată, prin acest sistem de monitorizare.

În vederea creșterii rezoluției procesorului de monitorizare ca să se poată detecta și erorile instrucțiunilor individuale se pot utiliza semnături calculate în locul semnăturilor atribuite. Semnătura unui nod dat poate fi determinată prin sumarea (modulo 2) a tuturor instrucțiunilor din nod prin orice altă sumă de control, ori cod similar.

Ca și în cazul precedent, semnăturile sunt stocate în procesorul de monitorizare și sunt, apoi comparate cu semnăturile determinate pe durata execuției programului de procesorul de monitorizare. Programul care va fi executat de procesorul de monitorizare pentru *GFC* din figura 2.(a) cu semnăturile calculate va fi cel din figura 2.(c).

Funcționalitatea procesorului de monitorizare poate fi, în principiu, extinsă să acopere o largă mulțime de erori ale datelor prin includerea unor *asertiuni* în programul executat de procesorul de monitorizare.

Aserțiunile sunt teste de rezonabilitate care verifică relațiile predeterminate dintre variabilele din program și sunt, în acest sens, o generalizare a testelor de acceptanță.

Aceste asertiuni le introduce programatorul în program și sunt mai degrabă o parte a software-ului de aplicație.

O aserțiune, ca o entitate care servește pentru verificarea corectitudinii execuției unui program, este o relație *invariantă* între variabilele programului. Aserțiunea este

inserată, de programator, în diverse puncte din program semnificând intenția acestuia de a-i asocia valoarea de adevăr în raport cu variabilele asociate respectivei aserțiunii. Aserțiunile pot fi definite în baza specificațiilor ori în raport cu anumite proprietăți ale algoritmului.

Utilizarea aserțiunilor pentru detectarea defectelor din hardware ori din software a fost propusă în [And79] și în [MMA84], spre exemplu.

Eficiența utilizării aserțiunilor a fost dovedită prin măsurători [AB81]. Anumite proiecte propuse să utilizeze aserțiunile sunt sofisticate și au un grad înalt de complexitate, deoarece s-a intenționat nu doar detectarea erorilor dar și recuperarea ultimei stări verificate [LS84].

Creșterea performanței prin verificarea aserțiunilor pe procesorul de monitorizare poate fi întrucâtva diminuată din cauza transferurilor valorilor relevante pentru aplicație de la procesorul principal spre procesorul de monitorizare. Aceste transferuri pot fi, uneori foarte frecvente.

Mai mult, procesorul de monitorizare, prin introducerea aserțiunilor, devine mai complicat deoarece acesta trebuie să aibă capacitatea să execute instrucțiuni aritmetice și logice, care până în acest punct nu erau necesare.

Există și soluții ingenioase care reușesc să țină la un nivel rezonabil efortul de calcul al procesorului de monitorizare în cazul aserțiunilor. Codul aserțiunilor este depus în memoria locală a procesorului de monitorizare ca o bibliotecă de funcții. Dacă procesorul de monitorizare verifică aplicațiile în care fluxul de date este cunoscut și invariant atunci, procesorul de monitorizare poate să capteze datele necesare prin supravegherea magistralei de date a procesorului verificat. Astfel de metode au fost propuse pentru comutarea telefoanelor și pentru sistemele de control a zborurilor.

În sistemele de uz general transferul datelor poate fi organizat printr-o memorie partajată ori printr-o coadă de mesaje. Prima abordare necesită tehnici de sincronizare mai complicate. Înaintea execuției programului acesta este modificat. Expresia aserțiunii este înlocuită printr-o singură instrucțiune care transferă valorile argumentelor aserțiunii și identificatorul funcției de verificare (funcția aserțiunii) procesorului de monitorizare.

Codul aserțiunii este descărcat într-o memorie locală a procesorului de monitorizare. Pe durata execuției programului, procesorul de monitorizare recepționează datele și execută funcția cerută. Dacă instrucțiunea logică reprezentată prin funcția aserțiunii este falsă atunci se semnalează o eroare.

Este posibilă evitarea utilizării aserțiunilor prin aplicarea unor tehnici superioare de detecție a erorilor (cum ar fi codurile evaluate de paritate etc.) implementate la nivelul procesorului de monitorizare.

Prin introducerea procesorului de monitorizare s-a introdus un bloc de circuite de verificare care este independent de circuitele verificate – sunt evitate astfel, erorile comune ori corelate dintre cele două blocuri. Aceasta este o trăsătură esențială a utilizării procesorului de monitorizare.

O protecție similară poate fi introdusă în structurile duplex prin utilizarea diversității tehnologice – utilizarea unor procesoare manufacturate distinct, provenind de la producători diferiți.

Separarea, diferențierea, dintre procesorul de monitorizare și procesorul principal devine tot mai dificil de realizat în tendința actuală a microprocesoarelor de nivel înalt, în care simpla monitorizare a magistralei procesor - memorie este insuficientă ca să se determine care instrucțiuni vor fi eventual executate și care au fost pregătite speculativ și vor fi pierdute, avortate.

Supportul procesării simultane a mai multor fire crește complexitatea proiectării procesorului de monitorizare.

Procesarea multi-fir pentru toleranța defectelor

Procesoarele actuale de vârf realizează performanțe superioare atât prin exploatarea procesării în bandă de asamblare cât și prin paralelism. Tehnologiile nanometrice de realizarea a circuitelor digitale facilitează mult paralelismul prin existența unor unități funcționale multiple ceea ce face posibilă execuția suprapusă a cât mai multor instrucțiuni este posibil.

Din cauza datelor și a dependențelor de control, cea mai mare parte a aplicațiilor au limite severe asupra paralelismului care poate fi inițiat în fiecare fir de execuție. Studii efectuate asupra unor colecții de programe etalon (benchmark-uri) au arătat că în medie se pot suprapune doar aproximativ 1,5 instrucțiuni.

Prin prisma acestui rezultat, cea mai mare a timpului, marea majoritate a unităților funcționale vor rămâne în așteptare, fiind ne-utilizate.

Un salt calitativ important prin comparație cu paralelismul este introducerea simultaneității procesării multi-fir.

Elementul cheie al procesării multi-fir constă în execuția în același timp, pe durata aceluiași impuls de ceas, a unor instrucțiuni aparținând mai multor fire.

Arhitectura microprocesorului trebuie dezvoltată corespunzător pentru ca să poată asigura suportul corespunzător. Un registru numărător program este necesar pentru fiecare dintre firele executate simultan în sistem.

Dacă setul instrucțiunilor specifică o arhitectură cu un set de k -registre fizice, atunci execuția simultană a n fire va cere cel puțin nk registre fizice (câte un set de registre fizice pentru fiecare fir de execuție). Arhitecturile avansate au și registre care nu sunt vizibile extern prin setul de instrucțiuni. Spre deosebire de registrele arhitecturale, registrele interne sunt partajate de toate firele de execuție simultane care partajează de asemenea și existența unei cozi comune.

Este necesar să se implementeze o politică corespunzătoare pentru pregătirea și pentru lansarea instrucțiunilor ca și pentru atribuirea registrelor interne și ale altor resurse, astfel încât nici un fir să nu fie defavorizat.

Diferența dintre rularea unei aplicații pe un multiprocesor cu n procesoare și pe un microprocesor care rulează n fire constă din modul în care sunt atribuite resursele.

În cazul multiprocesoarelor, tradiționale, fiecare procesor rulează un fir independent iar acel fir va avea acces doar la unitățile funcționale și la registrele pe care le va redenumi, asociate respectivului procesor.

În cazul sistemelor multi-fir există un set de fire care au acces la o mulțime de unități funcționale și care redenumesc registre. Utilizarea acestor entități va depinde de disponibilul paralelism din cadrul unui fir la un anumit moment. Aceasta situație se poate schimba în timp, deoarece cerințele de resurse din cadrul unui fir și nivelul de paralelism inerent se modifică în fiecare fir, simultan, de execuție.

Utilizarea proceselor de execuție multi-fir în scopul detecției defectelor deschide oportunități de implementare a unor facilități de toleranță la defecte.

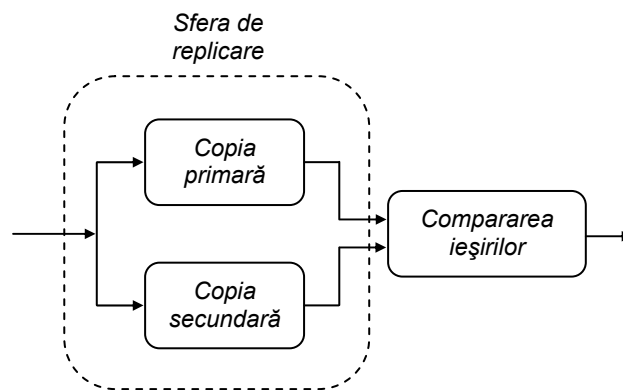


Figura 3. Sfera de replicare a celor două fire de execuție

În acest scop, pentru fiecare aplicație se deschid două fire independente de execuție. Aceste fire execută același cod și se are în vedere să primească aceleași intrări.

Dacă operarea decurge corect atunci aceste fire produc aceleași ieșiri. O divergență a ieșirilor denotă un defect și se impun măsurile de recuperare corespunzătoare.

Idea de bază este furnizarea aceluiasi nivel de protecție, împotriva defectelor tranzitorii, ce poate fi oferit într-o abordare tradițională care ar rula, în mod independent, două copii ale aceleiași aplicații.

Pentru micșorarea penalizării cauzate prin re-execuție, a doua execuție a aplicației urmează întotdeauna prima execuție. Cele două execuții se vor numi copia primară și respectiv copia secundară a aplicației. Avantajul acestei abordări constă în faptul că se poate transmite informație de la copia primară, spre copia secundară pentru creșterea vitezei de rulare a secundarei dar și pentru micșorarea consumului de resurse al acesteia.

Pentru cele două fire de execuție independente dar identice, ca aplicație, se vor alocă două seturi distincte de componente hardware.

Se vor alocă două seturi distincte de registre arhitecturale, astfel încât un defect dintr-un registru utilizat într-un fir de execuție nu avea nici un impact asupra execuției celuiilalt fir.

Aceste considerente conduc la introducerea unei *sfere de replicare*. Unitățile replicate pentru cele două fire de execuție se spune că aparțin acestei sfere, iar cele care nu sunt replicate sunt exterioare acestei sfere (Figura 3).

Fluxul de date traversează suprafața acestei sfere de replicare. Entitățile replicate sunt stabilite în funcție de costuri și de suprasarcina pe care acestea le implică dar și în raport cu eficiența implicată.

Toate elementele ce nu sunt cuprinse în sfera de replicare dar ar putea fi ținta unor potențiale erori tranzitorii pot fi protejate prin alte mijloace, cum ar fi codurile redundante auto-detectoare (auto-corectoare) etc.

Bibliografie

- [AB81] D.M. Andrews and J.P. Benson. An automated program testing methodology and its implementation. In *Proc. 5th Int. Conf. Software Eng.*, paginile 254-261, 1981.
- [And79] D.M. Andrews. Using executable assertions for testing and fault tolerance. In *Proc. 9th Int. Symposium on Fault Tolerant Computing*, paginile 102-105, 1979.
- [KGT89] J. Karlsson, U. Gunneflo, and J. Torin. The effect of heavy-ion induced single event upsets in the mc6890e microprocessor. In *Fault Tolerant Computing Systems*, volume 214 of *Informatik Fachberichte*, paginile 296-307. Springer Verlag, Berlin, 1989.
- [KK07] I. Koren, and C.M. Krishna. *Fault-tolerant Systems*, Elsevier Morgan Kaufmann, 2007.
- [LS84] Y.H. Lee and K.G. Shin. Design and evaluation of a fault tolerant multiprocessor using hardware recovery blocks. *IEEE Trans. on Comp.*, 33:113-124, February 1984.
- [MEM85] A. Mahmood, A. Ersoz, and E.J. McCluskey. Concurrent system-level error detection using a watchdog processor. In *Proc. 15th Int. Symposium on Fault Tolerant Computing*, paginile 145-152, 1985.
- [MM88] A. Mahmood, and E. J. McCluskey. Concurrent error detection using watchdog processors – a survey. *IEEE Trans. On Comp.*, 37(2):160-174, 1988.
- [MMA84] A. Mahmood, E. J. McCluskey, and D.M. Andrews. Writing executable assertions to test flight software. In *Conf. rec. 18th Annual Asilomar Conf. Circuits and System Conf.*, paginile 262-266, 1984.
- [MQS90] H. Madeira, G. Quadros, and J.G. Silva. Experimental evaluation of a set of simple error detection mechanism. *Microprocessing and Microprogramming*, 30:315-520, 1990.
- [SL81] D. P. Siewiorek and L. K. Lai. Testing of digital systems. *Proceedings of the IEEE*, 69:1321-1333, October 1981.
- [Sos94] J. Sosnowski. Transient fault tolerance in digital systems. *IEEE Micro*, paginile 24-35, February 1994.
- [STDM82] M.E. Schmid, R.L. Trapp, A.E. Davidoff, and G.M. Mason. Upset exposure by means of abstraction verification. In *proc. 12th Int. Symposium on Fault Tolerant Computing*, paginile 237-244, 1982.