ABORDAREA CONCURENTĂ A DEFECTELOR

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 combinaţionale şi 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 aplicaţii critice, poate impune un efort complex, posibil prohibitiv de costisitor care, adesea, se dovedeşte nejustificat.

Pentru astfel de situaţii s-au dezvoltat modalităţi mai simple necesitând dotări suplimentare mai puţin costisitoare. Aceste tehnici se bazează pe ipoteza că un astfel procesor execută aplicaţii curente iar în cazul apariţiei unei erori aplicaţia, ori o parte din aceasta, poate fi reluată şi re-executată dacă sunt îndeplinite două condiţii esenţiale:

(1) eroarea este detectată iar cauza erorii este un defect de scurtă durată, tranzitoriu,
(2) atunci când se reia execuţia aplicaţiei este foarte probabilă dispariţia erorii.

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

Tabelul 1.
Efectele iradierii cu ioni grei [KGT89]

Benchmark	Control flux			Control	Date	Altele
	permanent	temporar	latent			
Quicksort	62%	8%	16%	8%	4%	2%

Defectele tranzitorii din microprocesoare afectează mai cu seamă controlul fluxului de date şi instrucţiuni. Din aceste raţiuni, 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 şi a efectelor acestora în sistemele cu microprocesoare constituie un curent constant de cercetare. Pe măsură ce creşte complexitatea şi viteza dispozitivelor digitale se dovedeşte necesară considerarea a noi defecte şi tipuri de erori. În marea majoritate a cazurilor, complexitatea dispozitivelor sau incompleta cunoaştere a structurii interne ori chiar a operării constituie obstacole în constituirea modelelor defectelor la nivel coborât (nivelul porţilor ori nivelul transferului la nivel de registre).

Tabelul 2.
Efectele injectării defectelor hardware asupra magistralei sistemului

	Execuţie invalidă program	Adresa codului operaţiei	Adresă memorie invalidă	Adresă citire invalidă	Cod operaţie 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 consecinţelor defectelor de nivel inferior asupra comportamentului sistemelor, nivel superior. Toate consideraţii sunt

aplicabile, cu precădere, defectelor tranzitorii deoarece această clasă de defecte nu a dobândit o deplină caracterizare.
Atunci când se cunoaşte 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 fişierele de erori, log-urile de sistem, din faza operaţională 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
Efecte	33%	60%	7%	28%	8%	37%

Câteva din rezultatele cunoscute ale cercetărilor asupra erorilor din microprocesoare sunt prezentate în tabelele 1, 2 şi 3. Sunt considerate erorile relativ uşor de detectat din microprocesoare.
Tehnicile întrebuinţate sunt, între altele, simularea [ORG92], injectarea defectelor prin iradiere cu ioni grei [KGT89], disturbaţiile de putere şi modificarea semnalelor de pe magistralele sistemelor [MKS90], [STDM82].

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

Defectele tranzitorii pot fi cauzate în primul rând de zgomotul sursei de alimentare (variaţii ale frecvenţei de comutaţie), interferenţe electro-magnetice (reţele de comunicaţii fără fir şi 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 menţionate câteva deficienţe tipice, cum ar fi:

- Depăşirea numărului liniilor de intrare în anumite porţi,
- Circuitele funcţionând cu semnale marginale,
- Oscilaţiile circuitelor cu stări (oscilaţii între stările active şi cele inactive) etc.

O concluzie esenţială a acestor studii a fost determinarea celor mai eficiente mecanisme de detecţie a erorilor.
Printre acestea, în afară de verificarea datelor, sunt:

- verificarea fluxului de comenzi (controlul),
- verificarea modului de adresare al memoriei şi
- verificarea codurilor instrucţiunilor.

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

Tehnici de detecţie concurentă a erorilor

Aşa 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 şi prin testare pe durata procesului de manufacturare.
Testele clasice planificate sistematic, în afara rulajului normal al unui microprocesor, sunt consumatoare importante de timp şi nu sunt capabile să detecteze defectele tranzitorii. Sunt de preferat tehnici concurente *on-line* de detecţie a erorilor [KK07].

Se dovedeşte că unica abordare pentru creşterea dependabilităţii microprocesoarelor sunt tehnicile de toleranţă a defectelor. Deoarece dependabilitatea unui sistem este deosebit de sensibilă faţă de defectele nedetectate ori impropriu tratate [Sos94] aplicarea tehnicilor de detecţie eficientă a erorilor este de extrem interes.

Întrucât defectele tranzitorii nu produc defecte permanente în sisteme, şansele de recuperare sunt importante, cu atât mai mult atunci când acoperirea erorilor este mare iar latenţa este mică (evitându-se răspândirea efectelor erorilor şi 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
- Aplicaţia

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

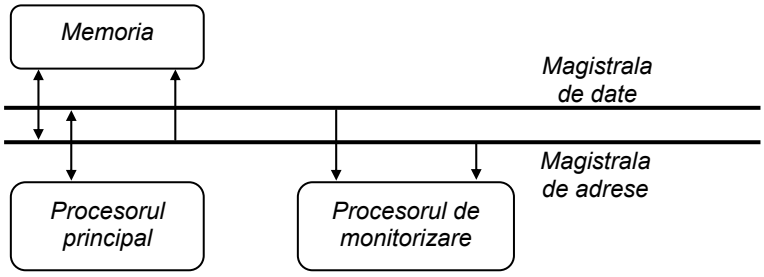


Figura 1. Detecţia erorilor prin utilizarea unui procesor de monitorizare.

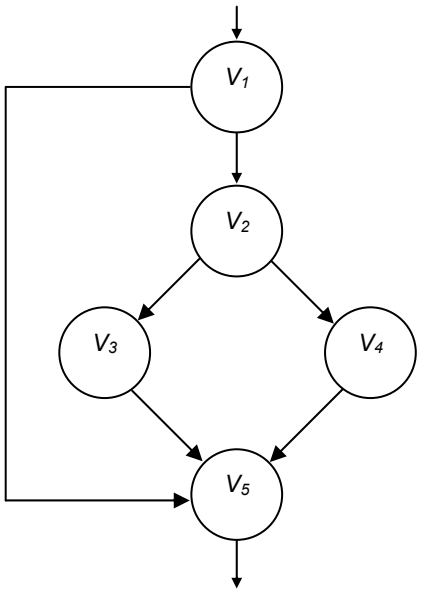
Dar, dacă este implementabil un mecanism care să detecteze erorile ce pot să apară pe durata execuţiei unei instrucţiuni, atunci instrucţiunea respectivă trebuie re-executată, preferabil după un anumit timp care să permită dispariţia defectului tranzitoriu.
O astfel de abordare care vizează doar re-execuţia unei singure instrucţiuni, la un anumit moment, are un impact mult mai mic asupra performanţei, comparativ cu re-execuţia în întregime a programului.
Au fost dezvoltate şi alte tehnici, care implică costuri mici, pentru tratarea concurentă a erorilor.

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

Procesorul de monitorizare

Un astfel de procesor realizează concurent, la nivelul sistemului, detecţia erorilor prin monitorizarea magistralelor sistemului care conectează procesorul şi memoria (aşa 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 şi erori software care să aibă drept urmare fie execuţia unor instrucţiuni eronate, fie a unor căi necorespunzătoare prin aplicaţii.



(a)

```
acceptă semnătura(V1);
fie
  acceptă semnătura (V2);
  fie
    acceptă semnătura (V3);
  ori
    acceptă semnătura (V4);
  acceptă semnătura (V5);
ori
  acceptă semnătura (V5);
```

(b)

```
acceptă şi verifică semnătura (V1);
fie
  acceptă şi verifică semnătura (V2);
  fie
    acceptă şi verifică semnătura (V3);
  ori
    acceptă şi verifică semnătura (V4);
  acceptă şi verifică semnătura (V5);
ori
  acceptă şi verifică semnătura (V5);
```

(c)

Figura 2.
(a) Graful fluxului de control.
(b) Verificarea fluxului de control.
(c) Verificarea nodurilor şi a fluxului de control.

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

Informaţia 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 instrucţiuni, fără ramificaţii, din secvenţa de calcul. Într-un astfel de bloc nu există instrucţiuni de salt spre instrucţiuni din bloc ori spre alte instrucţiuni din afara blocului şi nici nu există alte salturi din program care să ajungă la o instrucţiune din blocul considerat.

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

Pe durata execuţiei aplicaţiei, semnăturile nodurilor curent aflate în execuţie 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 execuţia programului, aşa cum ar fi calea $\{V_1, V_3\}$, spre exemplu. Dar, orice eroare dintr-un nod (din secvenţa corespunzătoare nodului) nu va putea fi identificată, prin acest sistem de monitorizare.

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

Ca şi în cazul precedent, semnăturile sunt stocate în procesorul de monitorizare şi sunt, apoi comparate cu semnăturile determinate pe durata execuţiei 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).

Funcţionalitatea procesorului de monitorizare poate fi, în principiu, extinsă să acopere o largă mulţime de erori ale datelor prin includerea unor *aserţiuni* în programul executat de procesorul de monitorizare.
Aserţiunile sunt teste de rezonabilitate care verifică relaţiile predeterminate dintre variabilele din program şi sunt, în acest sens, o generalizare a testelor de acceptanţă.

Aceste aserţiuni le introduce programatorul în program şi sunt mai degrabă o parte a software-ului de aplicaţie.
O aserţiune, ca o entitate care serveşte pentru verificarea corectitudinii execuţiei unui program, este o relaţie *invariantă* între variabilele programului. Aserţiunea este

inserată, de programator, în diverse puncte din program semnificând intenţia acestuia de a-i asocia valoarea de adevăr în raport cu variabilele asociate respectivei aserţiuni.
Aserţiunile pot fi definite în baza specificaţiilor ori în raport cu anumite proprietăţi ale algoritmului.
Utilizarea aserţiunilor pentru detectarea defectelor din hardware ori din software a fost propusă în [And79] şi în [MMA84], spre exemplu.
Eficienţa utilizării aserţiunilor a fost dovedită prin măsurători [AB81]. Anumite proiecte propuse să utilizeze aserţiunile sunt sofisticate şi au un grad înalt de complexitate, deoarece s-a intenţionat nu doar detectarea erorilor dar şi recuperarea ultimei stări verificate [LS84].

Creşterea performanţei prin verificarea aserţiunilor pe procesorul de monitorizare poate fi întrucâtva diminuată din cauza transferurilor valorilor relevante pentru aplicaţie de la procesorul principal spre procesorul de monitorizare. Aceste transferuri pot fi, uneori foarte frecvente.
Mai mult, procesorul de monitorizare, prin introducerea aserţiunilor, devine mai complicat deoarece acesta trebuie să aibă capacitatea să execute instrucţiuni aritmetice şi logice, care până în acest punct nu erau necesare.

Există şi soluţii ingenioase care reuşesc să ţină la un nivel rezonabil efortul de calcul al procesorului de monitorizare în cazul aserţiunilor. Codul aserţiunilor este depus în memoria locală a procesorului de monitorizare ca o bibliotecă de funcţii. Dacă procesorul de monitorizare verifică aplicaţiile în care fluxul de date este cunoscut şi 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 şi 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 execuţiei programului acesta este modificat. Expresia aserţiunii este înlocuită printr-o singură instrucţiune care transferă valorile argumentelor aserţiunii şi identificatorul funcţiei de verificare (funcţia aserţiunii) procesorului de monitorizare.

Codul aserţiunii este descărcat într-o memorie locală a procesorului de monitorizare. Pe durata execuţiei programului, procesorul de monitorizare recepţionează datele şi execută funcţia cerută. Dacă instrucţiunea logică reprezentată prin funcţia aserţiunii este falsă atunci se semnalează o eroare.

Este posibilă evitarea utilizării aserţiunilor prin aplicarea unor tehnici superioare de detecţie a erorilor (cum ar fi codurile evoluate 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 esenţială a utilizării procesorului de monitorizare.

O protecţie similară poate fi introdusă în structurile duplex prin utilizarea diversităţii tehnologice – utilizarea unor procesoare manufacturate distinct, provenind de la producători diferiţi.

Separarea, diferenţierea, dintre procesorul de monitorizare şi procesorul principal devine tot mai dificil de realizat în tendinţa actuală a microprocesoarelor de nivel înalt, în care simpla monitorizare a magistralei procesor - memorie este insuficientă ca să se determine care instrucţiuni vor fi eventual executate şi care au fost pregătite speculativ şi vor fi pierdute, avortate.
Suportul procesării simultane a mai multor fire creşte complexitatea proiectării procesorului de monitorizare.

Procesarea multi-fir pentru toleranţa defectelor

Procesoarele actuale de vârf realizează performanţe superioare atât prin exploatarea procesării în bandă de asamblare cât şi prin paralelism. Tehnologiile nanometrice de realizarea a circuitelor digitale facilitează mult paralelismul prin existenţa unor unităţi funcţionale multiple ceea ce face posibilă execuţia suprapusă a cât mai multor instrucţiuni este posibil.

Din cauza datelor şi a dependenţelor de control, cea mai mare parte a aplicaţiilor au limite severe asupra paralelismului care poate fi iniţiat în fiecare fir de execuţie. Studii efectuate asupra unor colecţii de programe etalon (benchmark-uri) au arătat că în medie se pot suprapune doar aproximativ 1,5 instrucţiuni.

Prin prisma acestui rezultat, cea mai mare a timpului, marea majoritate a unităţilor funcţionale vor rămâne în aşteptare, fiind ne-utilizate.
Un salt calitativ important prin comparaţie cu paralelismul este introducerea simultaneităţii procesării multi-fir.

Elementul cheie al procesării multi-fir constă în execuţia în acelaşi timp, pe durata aceluiaşi impuls de ceas, a unor instrucţiuni aparţinând 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 instrucţiunilor specifică o arhitectură cu un set de *k*-registre fizice, atunci execuţia simultană a *n* fire va cere cel puţin *nk* registre fizice (câte un set de registre fizice pentru fiecare fir de execuţie). Arhitecturile avansate au şi registre care nu sunt vizibile extern prin setul de instrucţiuni. Spre deosebire de registrele arhitecturale, registrele interne sunt partajate de toate firele de execuţie simultane care partajează de asemenea şi existenţa unei cozi comune.

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

Diferenţa dintre rularea unei aplicaţii pe un multiprocesor cu *n* procesoare şi pe un microprocesor care rulează *n* fire constă din modul în care sunt atribuite resursele.

În cazul multiprocesoarelor, tradiţionale, fiecare procesor rulează un fir independent iar acel fir va avea acces doar la unităţile funcţionale şi 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 mulţime de unităţi funcţionale şi care redenumesc registre. Utilizarea acestor entităţi va depinde de disponibilul paralelism din cadrul unui fir la un anumit moment. Aceasta situaţie se poate schimba în timp, deoarece cerinţele de resurse din cadrul unui fir şi nivelul de paralelism inerent se modifică în fiecare fir, simultan, de execuţie.

Utilizarea proceselor de execuţie multi-fir în scopul detecţiei defectelor deschide oportunităţi de implementare a unor facilităţi de toleranţă la defecte.

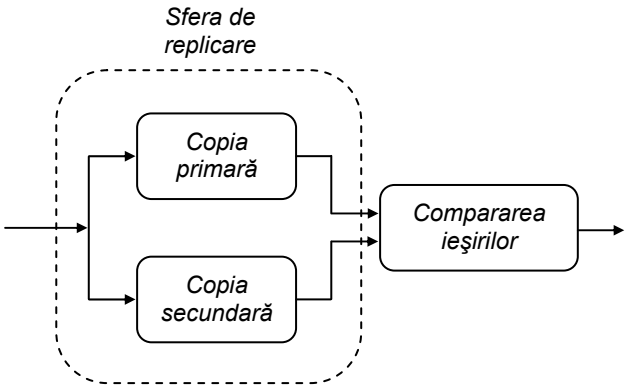


Figura 3. Sfera de replicare a celor două fire de execuţie

În acest scop, pentru fiecare aplicaţie se deschid două fire independente de execuţie. Aceste fire execută acelaşi cod şi se are în vedere să primească aceleaşi intrări.

Dacă operarea decurge corect atunci aceste fire produc aceleaşi ieşiri. O divergenţă a ieşirilor denotă un defect şi se impun măsurile de recuperare corespunzătoare.
Ideea de bază este furnizarea aceluiaşi nivel de protecţie, împotriva defectelor tranzitorii, ce poate fi oferit într-o abordare tradiţională care ar rula, în mod independent, două copii ale aceleiaşi aplicaţii.

Pentru micşorarea penalizării cauzate prin re-execuţie, a doua execuţie a aplicaţiei urmează întotdeauna prima execuţie. Cele două execuţii se vor numi copia primară şi respectiv copia secundară a aplicaţiei. Avantajul acestei abordări constă în faptul că se poate transmite informaţie de la copia primară, spre copia secundară pentru creşterea vitezei de rulare a secundarei dar şi pentru micşorarea consumului de resurse al acesteia.

Pentru cele două fire de execuţie independente dar identice, ca aplicaţie, se vor aloca două seturi distincte de componente hardware.
Se vor aloca două seturi distincte de registre arhitecturale, astfel încât un defect dintr-un registru utilizat într-un fir de execuţie nu avea nici un impact asupra execuţiei celuilalt fir.

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

Fluxul de date traversează suprafaţa acestei sfere de replicare. Entităţile replicate sunt stabilite în funcţie de costuri şi de suprasarcina pe care acestea le implică dar şi în raport cu eficienţa implicată.
Toate elementele ce nu sunt cuprinse în sfera de replicare dar ar putea fi ţinta unor potenţiale 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.