Andrej Žemva, Baldomir Zajc
Fakulteta za elektrotehniko in računalništvo
61000 Ljubljana, Tržaška 25

TEST - Sistem za testiranje kombinacijskih vezij

TEST - System for Testing Combinational Circuits

POVZETEK — **V članku je predstavljen sistem za avtomatsko generiranje testnih vzorcev za odkrivanje enojnih napak v kombinacijskih vezjih. Od obstoječih sistemov se razlikuje v tem, da omeji vozlišča, na katerih lahko spremenimo prvotno odločitev v procesu generiranja testnih vzorcev le na vhode celic, izbranih za prenos signala napake in tako zmanjša porabo časa v postopku popravljanja napačnih odločitev. Sistem vključuje tudi možnost simulacije napak, ki omogoča odkrivanje ostalih napak z generiranim testnim vzorcem. V samem procesu je upoštevan tudi čas prehajanja signalov preko posameznih celic, tako da je ob koncu postopka testiranja dan tudi podatek o minimalno potrebnem stanju signalov na vhodih vezja. Eksperimentalni rezultati dobljeni na standardnih testnih vezjih pričajo o učinkovitosti opisanega sistema.**

ABSTRACT — **Test system for automatic test pattern generation for single stuck-at faults of combinational circuits is described. It differs from systems previous described in reducing the backtracking time by limiting the backtracking points only to the inputs of the gates selected to propagate the fault signal towards the primary outputs of the circuit. Fault simulation option is included into the test pattern system in order to detect other faults by the recently generated test pattern. Timing conditions such as delays during propagating the signals through the gates are taken into account and the minimal time duration of signals on primary inputs of the circuit is obtained. Experimental results on benchmark circuits demonstrate the efficiency of the developed system.**

1. Uvod

Najnovejši razvoj na področju izdelave vezij visoke stopnje integracije ima velik vpliv tudi na testiranje vezij. Zaradi vse ve cje kompleksnosti vezij in omejitve dostopa do notranjih vozlišč, predstavlja cena testiranja pomemben delež v celotni ceni načrtovanja vezja, saj je problem odkrivanja napak NP polni problem [1] in časovna zahtevnost problema v najslabšem primeru narašča eksponentno z velikostjo vezja. Eden izmed današnjih pristopov pri načrtovanja vezij je, da že v fazi načrtovanja upoštevamo problem testiranja (design for testability), s čimer kasneje pospešimo izvajanje testiranja. Bistveno pri teh metodah je , da omogočijo v procesu testiranja transformacijo sekvenčnega vezja v kombinacijsko vezje in tako prevedejo problem testiranja zgolj na testiranje kombinacijskih vezij. V praksi se torej še vedno pojavlja potreba po razvoju učinkovitih algoritmov za testiranje kombinacijskih vezij.

Prvi popoln algoritem za odkrivanje neredundantnih napak je bil D-algoritem [2], kateri pa se je kasneje izkazal za zelo neučinkovitega pri analizi ECAT (error correction and translation) vezij . To vrzel je zapolnil PODEM algoritem [3], kateremu je kasneje sledil FAN algoritem. Na principih, ki so bili prvič predstavljeni v omenjenih sistemih, bazirajo v glavnem vsi kasnejši pristopi.

V članku je predstavljen algoritem, ki je v svoji osnovi podoben D-algoritmu, vendar se razlikuje glede izbire vozlišč, kjer je dovoljeno popravljanje napačnih odločitev v procesu testiranja. Algoritem upošteva tudi časovne razmere in je tako dobljeni testni vzorec lahko vhodni podatek generatorju testnih vzorcev. Nadaljevanje članka je organizirano v naslednjem vrstnem redu. Opis algoritma je podan v drugem delu, v tretjem je opis simulacije napak, sledi prikaz upoštevanja časovnih razmer delu, rezultati in zaključek pa so podani v petem oziroma šestem delu.

2. Opis algoritma

Program lahko analizira poljubno kombinacijsko vezje, sestavljeno iz osnovnih večvhodnih logičnih AND, NAND, OR, NOR in EX-(N)OR vrat. Napake, ki nastanejo v procesu izdelave integiranega vezja, lahko modeliramo z znanim stuck-at-1(0) modelom [2]. Napaka X stuck-at-0 pomeni, da je v vozlišču X lahko le logično stanje '0'. Podobno je definirano tudi stanje v vozlišču z napako stuck-at-1.

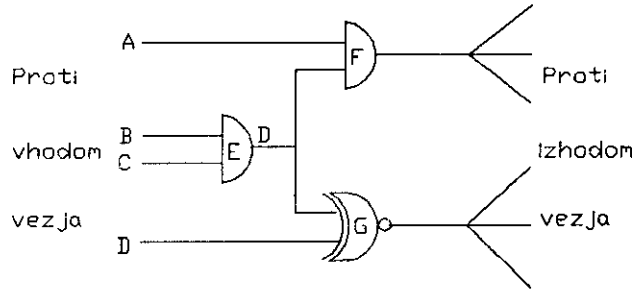
Osnovni cilj vsakega testnega algoritma je generacija takih vhodnih signalov, ki povzročijo, da je na vsaj eni izhodni celici vezja (PO primary output) stanje različno glede na to ali je v vezju napaka ali ne. Tako moramo v procesu generiranja testnih vzorcev izpolniti dve zahtevi.

V fazi aktivacije napake je potrebno na vhodih vezja (PI primary inputs) postaviti taka stanja, ki povzročijo v vozliču z napako stanje nasprotno predpostavljeni napaki. Tako moramo v primeru napake stuck-at-0 na izhodu AND vrat postaviti vse vhode v stanje '1' oziroma vsaj en vhod v stanje '0', če predpostavimo napako stuck-at-1. Stanje na izhodu AND vrat je sedaj signal napake in ima po dogovoru [2] vrednost D, ki predstavlja vrednost '1' v pravilnem vezju oziroma vrednost '0' v vezju z napako. $\bar{D}$ je komplement vrednosti D, medtem ko X predstavlja nedoločeno vrednost.

V fazi propagacije pa vodimo signal napake proti vsaj enemu izhodu vezja, saj so to edina vozlišča v vezju, kjer lahko opazujemo njihova stanja. Vsem nedoločenim vhodom celice, katero izberemo za prenos signala napake, priredimo stanje, ki omogoči prenos signala napake preko izbrane celice. V primeru, da je izbrana celica tipa (N)OR, je potrebno nedoločene vhode postaviti v stanje '0' oziroma v stanje '1', če je izbrana celica tipa (N)AND. Upoštevati pa je potrebno tudi možnost, da vrednost, ki je enaka vrednosti signala napake, katerega prenašamo, prav tako omogoči prenos signala napake na izhod celice. Vsaka druga določitev stanj na vhodih celice ima za posledico, da je signal na izhodu celice neodvisen od signala napake na njenem vhodu. Pri celicah tipa EX-(N)OR pa obstaja še več možnosti in je zato analiza vezij s to vrsto celic toliko bolj problematična.

Obe fazi se v procesu generiranja testnih vzorcev lahko izvedeta tudi v obratnem vrstnem redu in v opisanem algoritmu se najprej izvede faza propagacije napake proti izhodom vezja in na koncu, če je to potrebno, še faza aktivacije napake.

V samem procesu generiranja testnih vzorcev algoritem gradi odločitveno drevo, kjer je v vsakem vozlišču drevesa možnih več odločitev. Začetna odločitev je poljubna, vendar se lahko v kasnejšem procesu testiranja izkaže, da je bila odločitev napačna in je potrebno poskusiti z alternativno možnostjo (bactracking step). Za čim hitrejše izvajanje algoritma je potrebno zmanjšati število napačnih odločitev oziroma zminimizirati čas med popravljanjem napačnih odločitev. Osnovna razlika med D in PODEM algoritmom je izbira točk, kjer je možno spremeniti prvotno izbrane odločitve. Medtem, ko D algoritem dovoljuje spreminjanje odločitev v vsakem vozlišču odločitvenega drevesa, dovoljuje PODEM algoritem spreminjanje stanj le na vhodih vezja. Opisani algoritem pa se od njiju razlikuje v tem, da je možno odločitve spreminjati le na vhodih celice izbrane za prenos signala napake. Predpostavimo primer (slika 1).



Slika 1

Vzemimo, da imamo signal napake 'D' na izhodu AND vrat E. V fazi propagacije sledi izbira celice, preko katere želimo prenesti signal napake. Naj bo najprej izbrana celica AND F. Hevristika, na podlagi katere je izbrana določena celica, je opisana v tretjem delu. V naslednjem koraku je potrebno nedoločena stanja na vhodih celice postaviti v stanje, ki omogoči prenos signala napake preko izbrane celice. Najprej poskusimo postaviti vhod A v stanje '1'. Na vhodih vezja (PI) je

sedaj potrebno postaviti ustrezne signale. da ugodimo zahtevi po želenem stanju v vozlišču A. Proces določitve vhodnih stanj (backtrace step) je realiziran podobno kot v PODEM algoritmu. Ob vsaki novi postavitvi določenega stanja v vezju se vedno izvedejo tudi vse posledice na ostalih vozliščih vezja (implication step), da bi se potencialni konflikti med želenim stanjem in stanjem, ki v vezju že eksistira, čim hitreje odkrili. Predpostavimo, da smo uspešno določili vhodne signale in so sedaj AND vrata F tista logična vrata, s katerimi ponovimo opisani postopek. Če pa se v nadaljnjem izvajanju izkaže, da iz celice F ni bilo mogoče voditi signala napake proti izhodom vezja (PO), potem vsa vozlišča v vezju, katerih stanje je posledica postavitve vhoda A celice F v '1', dobijo svoja prvotna stanja in poskusi se z alternativno postavitvijo stanja vozlišču A, ki je enako signalu napake, ki ga prenašamo preko celice (v tem primeru signal 'D'). Ker pa obstaja majhna verjetnost, da bi bila ta nastavitev sploh možna se najprej izvede funkcija, ki preveri smiselnost take nastavitve. Ta nastavitev je možna le v primeru, če obstaja nedoločena pot oziroma je pot določena le s signalom napake od izhodiščnega vozlišča z napako pa do vozlišča, kjer želimo postaviti to stanje (v tem primeru vozlišča A). Šele v primeru, če se tudi ta odločitev izkaže za neuspešno, se izbere naslednja celica za prenos signala napake (v tem primeru celica EX-NOR celica G).

Postopek se izvaja tako dolgo dokler signala napake ne pripeljemo do ene izmed izhodnih celic oziroma ne poskusimo z vsemi možnostmi. V prvem primeru se nato preveri, če trenutna nastavitev vhodnih signalov že aktivira napako. V tem primeru je proces generacije testnega vzorca končan, v nasprotnem primeru pa se izvede še operacija (backtrace step) na enak način kot pri določitvi stanj na vhodih celic, izbranih za prenos signala napake.

3. Simulacija napak

V procesu generiranja testnih vzorcev odločitve bazirajo na parametrih (testability measures). Ti parametri so lahko statični, če upoštevajo le strukturo vezja, oziroma dinamični, če na njihovo vrednost vplivajo tudi trenutna stanja v vezju. Pomembni so tako pri izbiri celice za prenos signala napake kot tudi pri izbiri vhodov celice za izpolnitev zahteve po želenem stanju na izhode celice. V prvem primeru, ko skušamo izbrati celico za prenos signala napake, je potrebno najti celico, ki leži hkrati čim bližje izhodu vezja in hkrati omogoča čim enostavnejšo določitev stanj na njenih vhodih. Eksperimentalne meritve [5] so pokazale, da v splošnem najboljše rezultate dosegajo sistemi, ki upoštevajo parametre na način kot jih uporablja sistem SCOAP [6]. Upoštevanje dinamičnih parametrov, ki odsevajo trenutna stanja v vezju, v praksi ne pride v poštev, saj je poraba računalniškega časa za njihov vsakokratni izračun izrazito prevelika.

Dejansko pa nas v praksi ne zanima testni vzorec za točno določeno napako, temveč niz testnih vzorcev s katerimi odkrijemo čim več potencialnih napak v vezju. Tako lahko z generiranim testnim vzorcem odkrijemo tudi druge napake v vezju in to toliko več, kolikor manj je nedoločenih stanj v generiranem testnem vzorcu. Torej se lahko pri vseh odločitvah pri izbiri celice za prenos signala napake upošteva podatek, ali je za izbrano celico že bil generiran test v enem od prejšnjih poskusov. Na ta nain deluje tudi opisani algoritem.

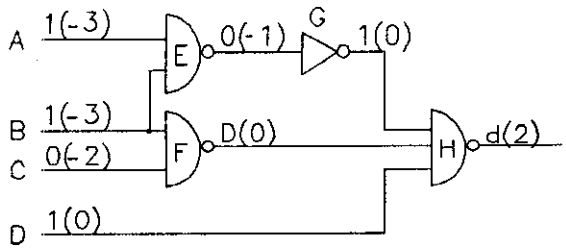
Če v primeru na sliki 1 predpostavimo, da je bil za celico F že generiran test za napako stuck-at-0, program najprej izbere celico G, čeprav bi bila pot preko celice F morebiti lažja. Kljub temu lahko v končnem testnem vzorcu ostane eden ali več vhodov nedoločenih. Algoritem v tem primeru nedoločenim vhodom naključno priredi vrednosti '1' oziroma '0' z namenom, da generirani testni vzorec odkrije še več preostalih napak.

Prvotna izvedba z vsakokratnim preverjanjem medsebojne neodvisnosti generiranega testnega vzorca s prejšnjimi testnimi vzorci in eliminacija nepotrebnega testnega vzorca se je izkazala za časovno neučinkovito glede na število ugotovljenih nepotrebnih testnih vzorcev.

4. Upoštevanje časovnih razmer

Za vsak prehod signala preko logičnih vrat je potreben določen končni časovni interval, zato je važen tudi podatek, kolikšen čas je potreben od trenutka nastavitve vhodov vezja v stanja, ki jih je generiral testni algoritem, pa do trenutka, ko se informacija o pravilnosti delovanja vezja pojavi na izhodu. Te podatke je enostavno dobiti vzporedno s potekom generacije testnih vzorecev. Maksimalni čas prehoda signalov namreč določa maksimalno frekvenco generatorju testnih signalov. Ti časi so pri različnih testnih vzorcih različni in v primeru možnosti generacije testnih vzorcev v neenakih časovnih intervalih ta podatek tudi uporabimo. Podatki o času prehoda signalov preko posameznih celic so zapisani v knjižnici celic, katere lahko uporabnik poljubno spreminja. Upoštevanje časovnih razmer je prikazano na primeru (slika 2). Na

izhodu vsake celice je podano trenutno stanje ter čas v katerem se to stanje pojavi (v oklepaju).



Slika 2

Naj bo napaka stuck-at-0 na izhodu NAND vrat F ter čas, v katerem se pojavi signal napake, enak 0. Zaradi enostavnosti vzemimo, da je število časovnih enot, potrebnih za prenos signala preko celice, enako številu vhodov celice. Signal napake lahko prenesemo le preko NAND vrat H. Vrednosti, ki jih moramo postaviti na preostale vhode v času 0, so '1'. Logična enica na izhodu inverterja ob času 0 zahteva na vhodu stanje '0' ob času -1, kar pa ima zopet za posledico stanji '1' na vhodih A in B ob času -3. Na podoben način določimo tudi preostala stanja in ustrezne čase v vezju. Minimalni čas trajanja testnega vzorca je tako razlika med časovnim trenutkom, ko se na izhodu vezja pojavi signal napake in med najbolj negativnim časom, v katerem se mora pojaviti ustrezni vhodni signal.

5. Rezultati

Predstavljeni program smo preiskusili na standardnih ISCAS testnih vezjih [9]. Rezultati, prikazani v tabeli 1, so dobljeni na osebnem računalniku PC-AT 80386 25MHz. Zaradi fizično omejenenih zmogljivosti osebnega računalnika, algoritma ni bilo mogoče preiskusiti tudi na dveh največjih testnih vezjih c6288 (16 bitni množilnik) ter c7552 (ALE in kontrolno vezje).

Ime vezja	Št. vrat	Št. nap.	Št. t. vz.	Št.ned. nap.	Pokr. [%]	Pov. čas za nap. [ms]
C432	160	524	57	4	99,23	48,1
C499	202	758	61	8	98,94	34,7
C880	383	942	69	0	100	28,4
C1355	546	1574	92	11	99,30	102,3
c1908	880	1879	106	14	99,25	112,6
c2670	1193	2747	190	115	95,82	132,9
c3540	1669	3428	296	151	95,60	274,6
c5315	2307	5350	231	73	98,64	74,8

Tabela 1.

V prvem stolpcu tabele je ime vezja, nato sledi število vrat ter število napak, število testnih vzorcev, število napak, za katere program ni generiral testa, razmerje med številom neodkritih napak ter celotnim številom napak, saj se v objavljenih člankih pojavljajo različni podatki o številu redundantnih napak za ista vezja. Rezultati so dobljeni na način opisan v tretjem in četrtem delu. Po vsakem uspešno generiranem testnem se je izvršila simulacija ostalih napak v vezju, za katere kasneje testnega vzorca ni bilo potrebno generirati. Doseženi rezultati so povsem primerljivi z do sedaj objavljenimi rezultati [8], [9]. Celice so bile v procesu generiranja testnih vzorcev od začetnega nivoja (vhodi vezja) proti višjim nivojem. Povprečen čas za napako je kvocient med celotnim časom (vključno z branjem vhodne datoteke s podatki o vezju (netlist) in zapisom rezultatov v izhodno datoteko) ter številom napak za katere je bilo možno določiti testni vzorec.

6. Zaključek

Predstavljeni algoritem je bilo tudi preprosto programsko realizirati, saj narava problema omogoča rekurzivno realizacijo, kar pa je tudi vzrok, da največjih vezij ni bilo možno preiskusiti na osebnem

računalniku. Program vsebuje približno 2400 vrstic izvorne kode v programskem jeziku C in ga je možno z manjšimi modifikacijami izvajati tudi na večjih računalniških sistemih. Program je še v razvojni fazi, zato lahko pričakujemo tudi boljše rezultate. Predvsem bo potrebno podrobneje analizirati vpliv vrstnega reda napak za katere želimo generirati testni vzorec. Analizirati velja tudi kompromis med številom testnih vzorcev in pa čas, potreben za testiranje vezja. Knjižnica osnovnih logičnih celic je fleksibilna in jo lahko po potrebi enostavno dopolnimo tudi z drugimi celicami oziroma spremenimo obstoječe parametre.

ZAHVALA

Avtorja članka se zahvaljujeta F. Brglezu in D. Bryanu, MCNC za ISCAS testna vezja.

Literatura

[1] O. H. Ibarra and S. K. Sahni: "Polynomially Complete Fault Detection Problems", *IEEE Trans. Comput.*, vol C-24, pp. 242-249, March 1975

[2] J. P. Roth: "Diagnosis of automata failures: A calculus and a method", *IBM J. Res. Develop.*, vol. 10, pp. 278-291, July 1966.

[3] P. Goel: "An implicit enumeration algorithm to generate tests for combinational circuits", *IEEE Trans. Comput.*, vol. C-30, pp. 215-222, Mar. 1981.

[4] H. Fujiwara and T. Shimono: "On the acceleration of test generation algorithms", *IEEE Trans. Comput.*, vol. C-32, pp. 1137-1144, Dec. 1983.

[5] S. J. Chandra and J. H. Patel: "Experimental evaluation of testabilty measures for test generation", *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 93-97, Jan. 1989.

[6] L. H. Goldstein and E. L. Thigpen: "SCOAP: Sandia controllability observability analysis program", *Proc. 17th IEEE Design Automation Conf.*, pp. 190-196, 1980.

[7] M. H. Schulz, E. Trischler and T. M. Sarfert: "SOCRATES: A Higly Efficient Automatic Test Pattern Generation System", *IEEE Trans. Computer-Aided Design*, vol. 7 pp. 126-137, Jan. 1988.

[8] Y. Takamatsu and K. Kinoshita: "CONT: A concurrent test generation system", *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 966-972, Sept. 1989.

[9] F. Brglez and H. Fujiwara: "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," in *Proc IEEE Int. Symp. Circuits Syst.*, June 1985.