KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Aleksej Drovnenkov

TESTINIŲ RINKINIŲ ATRINKIMO PROGRAMINĖS ĮRANGOS SUDARYMAS IR TYRIMAS

Magistro darbas

Vadovas
prof. dr.
Vacius Jusas

KAUNAS, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. Eduardas Bareiša
2007-05-28

TESTINIŲ RINKINIŲ ATRINKIMO PROGRAMINĖS ĮRANGOS SUDARYMAS IR TYRIMAS

Programų sistemų inžinerijos magistro baigiamasis darbas

Vadovas
prof. dr. Vacius Jusas
2007-05-24

Recenzentas
doc. dr. A. Lenkevičius
2007-05-24

Atliko
IFM-1/2 gr. stud.
A. Drovnenkov
2007-05-24

KAUNAS, 2007

KVALIFIKACINĖ KOMISIJA

Pirmininkas:

Sekretorius:

Nariai:

SUMMARY

CONSTRUCTION AND RESEARCH OF SOFTWARE FOR TEST PATTERNS SELECTION

Automated test pattern generation (ATPG) problem is being solved for a relatively long time. Its' point is to find optimal test vector sequences, which would cover most of all production-caused digital circuit faults and would run for the minimum amount of time.

One of the ways to test and generate test vectors for digital circuits is functional test method. Its' benefit is that system does not need to be aware of digital circuit's inner logical model, but has to deal only with the input, so that just the ideal model of the digital circuit can be used as a "black box". The program's algorithm can get ideal digital circuit's reaction for corresponding input test vector. This paper will mostly cover functional model approach to ATPG.

This paper covers automated test vector generation software basic theory with brief historical review, comparison of white box and black box models' testing and test vector generation algorithms. Also the software's static structures along with its components, system’s typical use cases are covered. The research part of the paper is focused mostly on the algorithms used, containing research methods which provide the results for the experiment part.

TURINYS

1. ĮVADAS

1.1. Santrauka

Automatinis testų rinkinių generavimas (pasaulyje priimtas angliškas sutrumpinimas – ATPG) yra pakankamai senai sprendžiama problema. Jos tikslas – surasti optimalų testinių vektorių sekas, kurios pilnai užtikrintų visas schemos gamybos etape padarytas klaidas per mažiausią laiką.

Vienas iš skaitmeninių schemų testavimo ir testų rinkinių sudarymo metodas yra funkcinis testavimo metodas. Jo privalumai yra tame, kad testų rinkinių sudarymo programa nežino schemos vidinės struktūros, o testuoja tik idealų schemos modelį, kuri yra pateikta juodos dėžės pavidale, tai yra programa gali gauti idealaus schemos reakciją į tam tikrą įvedimo signalų vektorių. Šiame darbe parinktas funkcinis testavimo metodas.

Šiame darbe aprašoma testinių rinkinių atrinkimo programinės įrangos teorinė bazė, automatinio testų rinkinio formavimo trumpa istorinė apžvalga, baltos ir juodos dėžės modelių pagristų formavimo algoritmų palyginimai. Aprašoma programų sistemos statinė struktūra bei jos komponentai, sistemos panaudojimo atvejai. Tyrimų dalyje aprašoma tyrimo metodika, siūlomi programos kokybės tobulinimo metodai. Eksperimentų dalyje aprašomi tyrimų eksperimentų rezultatai.

2. ANALITINĖ DALIS

2.1. Probleminė sritis

2.1.1. Problemos apibrėžimas

Skaitmeninių schemų gamybos procese vienas svarbiausių etapų yra kokybės aptikimas. Brokuotų schemų procentas yra pagrindinis gamybos kokybės rodiklis – esant dideliam brokuotų schemų skaičiui priimami gamybos proceso keitimo sprendimai – keičiamos technologijos, medžiagos, gamybos kontrolė. Tam, kad nustatyti schemos kokybę, yra naudojami testai, kuriuos sudaro testų vektorių rinkiniai. Testų vektorius – tai įėjimo signalų vektorius ir etaloninės (idealios) schemos atsakas – išėjimų signalų vektorius. Įėjimo signalai paduodami į testuojamą schemą, gaunamas schemos atsakas, kuris yra sulyginamas su idealiu atsaku. Jei nors vienas schemos išėjimo signalų vektorius su juo nesutampa, ši schema yra brokuota.

Jei schema yra palyginamai paprasta, lengva sudaryti tokius testų vektorių rinkinius, kurie patikrina visų schemos loginių mazgų korektišką veikimą. Tai gali būti rankiniu būdu parinkti testiniai vektoriai, arba pilnas galimų įėjimų signalų vektorių perrinkimas. Bet kada schemoje yra daugiau už 1000 loginių mazgų, rankinis testų rinkinių sudarymo būdas yra labai sudėtinga problema, o kai įėjimų skaičius yra didelis, pilno perrinkimo metodu sudaryti testų rinkiniai yra labai dideli – pavyzdžiui, kai schemoje yra 20 įėjimų, pilnas perrinkimas duoda 2^{20} = 1048576 testinių vektorių. Kadangi šiuolaikinėse schemose šitie skaičiai yra žymiai didesni, testavimui naudojami automatiškai sugeneruoti testinių vektorių rinkiniai.

Kadangi šiuolaikinėse skaitmeninėse schemose yra milijonai loginių mazgų ir šimtai skaitmeninių įėjimų, testų rinkiniai turi būti sudaryti automatiškai. Klasikinis automatinio testų rinkinių generavimo uždavinys yra „sudaryti minimalų testų rinkinį pagal duotą idealios schemos juodos dėžės modelį, kuris aptiktų visus galimus schemos gedimus“ [2]. Testo rinkinio dydis (testinių vektorių skaičius rinkinyje) yra kritinis rodiklis, nuo jo priklauso schemos testavimo laikas, o jei šis laikas kelis kartus viršija schemos gamybos laiką, tai labai sulėtina gamybos procesą, taigi padidina gamybos kainą.

2.1.2. Istorinė apžvalga

Svarbiausios datos ir įvykiai automatinių testų rinkinių sudarymo sistemų ir algoritmų raidoje [3]:

- 1959 – Pirmas struktūrinis testavimas skirtas Honeywell Datamatic 1000 kompiuteriui (Eldredas)
- 1961 – Pirmoji publikacija apie "sustojimas ant 0" ir "sustojimas ant 1" gedimus (Geilis, Norbis, Rotas)
- 1962 – Pirmas sustojimų gedimų panaudojimas gedimų emuliacijai (Sešus ir Frimenas)
- 1963 – Teorinė sustojimų gedimų analize (Podžas)
- 1966 – IBM laboratorijoje Rotas sukūrė D-ALG algoritmą
- 1975 – Įrodyta, kad automatinė testų rinkinių sudarymo problema yra NP-pilna
- 1981 – Sukurtas PODEM algoritmas
- 1983 – Sukurtas FAN algoritmas
- 1987 – Sukurtas TOPS algoritmas
- 1988 – Sukurta SOCRATES automatinė testų rinkinių sudarymo sistema
- 1991 – Sukurta EST automatinė testų rinkinių sudarymo sistema
- 1993 – Sukurta TRAN automatinė testų rinkinių sudarymo sistema
- 1995 – Sukurtas rekursyvaus apmokymo algoritmas
- 1997 – Taferthoferis pasiūlė naują implikacijų paremtą algoritmą
- 1999 – Taferthoferis sukūrė savo algoritmo optimalesnę versiją

2.1.3. Gedimų tipai

Svarbiausios gedimų charakteristikos yra [4]:

- priklausomumas nuo laiko
- fizinė prigimtis
- specifikacijos pažeidimo pasekmių lygis

Pagal kiekvieną iš šių trijų charakteristikų gedimai skiriasi į du tipus, kas duoda 2^3=8 skirtingus gedimų tipus.

Gedimų priklausomumas nuo laiko

Ši charakteristika parodo gedimo atsitikimą laikui bėgant. Pagal šią charakteristiką gedimai skirstomi į nereguliarus ir pastovius. Pagal savo atsitiktinę prigimtį nereguliarieji gedimai yra žymiai sunkiau aptinkami gamybos testų procese. Projektavimas ir gamybos proceso inžinerija yra svarbiausi dalykai tokių gedimų skaičių minimizavimui. Tokiuose atvejuose, kai nereguliarieji gedimai negali būti pašalinti arba toleruoti, pasekmių sumažinimui naudojami tokie metodai kaip operatyvaus valdymo klaidų aptikimas arba klaidų kontrolė su klaidų korekcijos kontūru.

Fizinė prigimtis

Visi puslaidininkiniai prietaisai priklauso nuo jų gamybos procese sukeltų nedidelių nukrypimų. Šie nukrypimai atsiranda dėl mažų atsitiktinių pokyčių gamybos proceso parametruose, tokių kaip priemaišų koncentracija, oksidų sluoksnio storumo netiesiškumo, geometrijos pakeitimai dėl per mažo ar per didelio ėsdinimo, arba didelių fizinių charakteristikų pakeitimo dėl teršalų dėmių, dielektrikų skylių, plyšių, topologijos pakeitimų ir t. t.

Analoginių arba mišriųjų tipų schemoms, bet kurio schemos topologijos fizinio parametro pakeitimas, kuris sukelia schemos specifikacijoje nustatytų ribų peržengimą yra vadinamas gedimu.

Pagal šį apibrėžimą, visi nedideli fizinių parametrų pokyčiai, kurie sukelia schemos veikimo gedimus, klasifikuojami kaip lengvi gedimai. Gedimai su dideliais fizinių parametrų pokyčiais arba topologijos kitimais yra vadinami sunkiais gedimais.

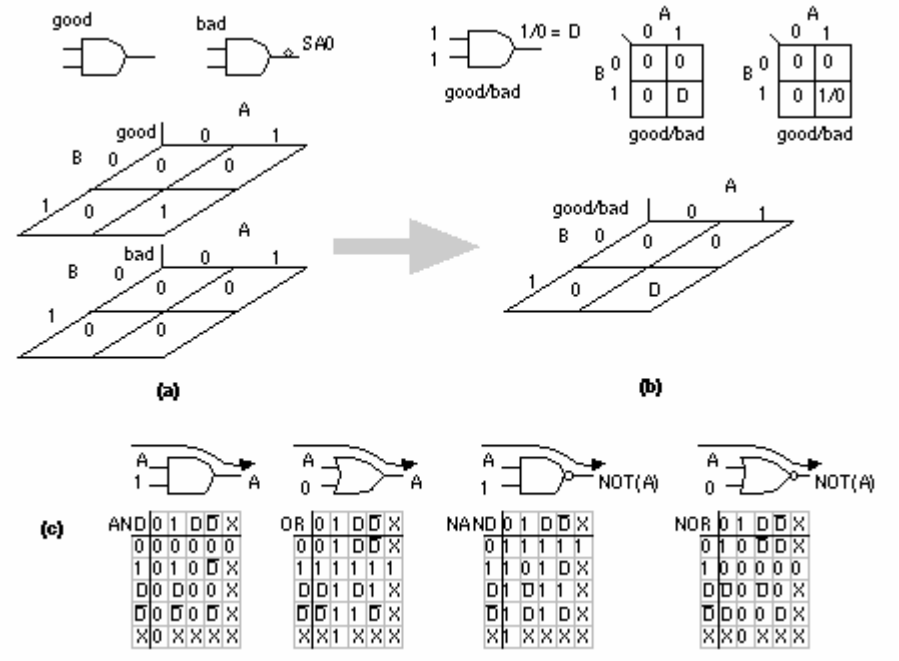
Specifikacijos pažeidimo pasekmių lygis

Ši charakteristika naudojama, kai reikia įvertinti schemai padaryta žalą dėl schemos naudojimosi specifikacijos pažeidimo. Pagal šią charakteristiką, gedimas yra vadinamas parametriniu gedimu, jei specifikacijos pažeidimas šukelė nedidelį peržengimą už specifikacijoje nustatytas ribas. Kai dėl gedimo atsiranda didelis skirtumas tarp laukiamos ir gautos reikšmių, jis vadinamas katastrofiniu.

2.1.4. Sprendimo algoritmai

D-ALG

D-ALG, taip pat žinomas kaip D-algoritmas arba D-CALCULUS, yra vienas iš pirmųjų teoriškai pagrįstų automatinių testų rinkinių sudarymo algoritmų. Jis buvo pasiūlytas IBM laboratorijoje 1966 m. Dž. P. Rotu [5]



2.1 pav. D-CALCULUS

1.1 pav. (a) ir (b) parodo D-CALCULUS sutrumpintą formą gedimų radimui. Simbolis D („detect“ – aptikti) indukuoja mazgo reikšmę kaip loginį nulį gerai veikiančiam loginiam elementui ir kaip loginį vienetą elementui su gedimais. Mes galime užrašyti D = 0/1. Bendru atveju rašoma *g/b* („good/bad“ – „gera/bloga“), kombinuota loginė reikšmė, kurioje *g* žymi gerai veikiantį elementą, o *b* – blogai veikiantį (pagal priimtas taisykles, gerai veikiančios schemos reikšmė rašoma pirma, o blogai veikiančios – antra). D reikšmės papildymas žymimas kaip D’ = 1/0. D’ nereiškia, kad loginio elemento gedimai nebuvo aptikti, o paprastai parodo loginį nulį gerai veikiančiam loginiam elementui ir loginį vienetą blogai veikiančiam. Mes galime pritaikyti Bulio algebrą D ir D’ reikšmėms kaip parodyta 1.1 pav. (c). Kombinuotos loginės reikšmės 1/1 ir 0/0 yra atitinkamai ekvivalentiškos loginiam vienetui ir loginiam nuliui. Mes naudosime loginę reikšmę “X”, kuri gali būti lygi vienai iš reikšmių 0, 1, D arba D’, bet mums nežinoma (arba kai kuriais atvejais nereikia žinoti) kuriai.

Jei mes norime paskleisti signalą nuo vieno arba daugiau loginio elemento įėjimo iki loginio elemento išėjimo, mes turime nustatyti kitus įėjimus į taip vadinamą „įjungimo reikšmę“. AND ir NAND loginiams elementams įjungimo reikšmė lygi loginiam vienetui, o OR ir NOR loginiams elementams įjungimo reikšmė lygi loginiam nuliui. 1.1 pav. (c) parodo įjungimo reikšmes. Priešingai, nustatant nors viename iš įėjimų kontrolinę reikšmė, kuri yra priešinga įjungimo reikšmei, priverčia fiksuotą reikšmę to loginio elemento išėjime. Kontrolinė reikšmė 0 AND loginiam elementui visada duoda loginį nulį išėjime, o NAND elementui – loginį vienetą išėjime. Kontrolinė reikšmė 1 visada duoda loginį vienetą OR loginio elemento išėjime ir loginį nulį NOR loginio elemento išėjime. Tam, kad surasti kontrolines ir įjungimo reikšmes sudėtingesniems loginiams elementams, mes galime naudotis jų paprastesniu AND, OR, NAND ir NOR loginių elementų atvaizdavimu.

Algoritminis testų rinkinių sudarymas susideda iš keturių žingsnių: 1) gedimo parinkimas, reikią parinkti gedimą, kuris turi būti aptiktas; 2) pradinis priskyrimas, tokio įvedimų signalų šablono radimas (naudojant kontrolines ir įjungimo reikšmes), kuris duotų D arba D' sugedusio loginio elemento išėjime; 3) stumdymasis į priekį, t. y. D arba D' perdavimas į išėjimą trumpiausiu keliu; 4) pateisinimas, t. y. reikšmių priskyrimas kitiems įėjimo signalams tam, kad atšaukti priskyrimus, padarytus 3 žingsnyje. Jei 4 žingsnyje atsiranda prieštaringumas, 3 žingsnis kartojamas, naudojant sekantį ilgesnį kelią. Šis rekursyvus ciklas yra kartojamas kol nesusikaupia pakankamai daug įėjimo signalų vektorių gedimui aptikti.

PODEM algoritmas

PODEM algoritmas buvo pasiūlytas Goeliu 1981 m. [6]. Jis yra D-ALG modifikacija, kuri spręsdavo problemą su D-ALG algoritmo XOR loginių elementų apskaičiavimu.

Kiekvienam gedimui, kuris turi būti aptiktas, PODEM algoritmas tikrina visas įėjimų reikšmes, kol nesuranda galiojančio testų vektoriaus šiam gedimui. Tai pasiekiama priskiriant skirtingas reikšmes pirminiams loginės schemos įėjimams. Kiekvieną kartą, kai naujos reikšmės yra priskiriamos pirminiams įėjimams, nagrinėjama jų implikacija – ar jos dalyvauja gedimo aptikime, ar ne. Įėjimo vektoriai, kurie trukdo gedimo aptikimui, yra pakeičiami taip vadinamu „backpropagation“ („atvirkštinis perdavimas“) metodu. PODEM pakartotinai priskiria skirtingas reikšmes pirminiams įėjimams kol neatsiranda gedimą

aptinkantis testinis vektorius. PODEM taip pat sustoja kai visos įmanomos kombinacijos buvo perrinktos ir nebuvo surasta nei vieno tinkamo testinio vektoriaus, šiuo atveju gedimas žymimas kaip neatskleidžiamas.

FAN algoritmas

FAN algoritmas yra tolimesnis PODEM algoritmo patobulinimas [7] su kai kuriomis papildomomis savybėmis. Pavyzdžiui, jis išnaudoja loginės schemos topologijos turimą informaciją paieškos efektyvumo padidinimui. FAN algoritmas skiriasi nuo PODEM algoritmo keliomis ypatybėmis: jis sustabdo atvirkštinį perdavimą kai kuriose vidinėse linijose, jis vykdo kelis atvirkštinius perdavimus, jis leidžia tiesiogines ir atvirkštines implikacijas ir jis iš karto nustato unikaliai atpažintus loginius lygius.

Monte-Carlo metodas

Monte-Carlo metodo pavadinimas kyla iš Monako kunigaikštystės Monte Karlo miesto vardo, kuris yra gerai žinomas pasaulyje kaip Europos lošimų namų sostinė. Šis metodas yra plačiai naudojamas statistikoje, skaitiniuose metoduose, kvantinėje mechanikoje ir daugelyje kitų mokslo šakų, kur kiekvieno sistemos elemento išnagrinėti yra neįmanoma. Šio metodo pagrindas – atsitiktinių skaičių generacija, funkcijos reikšmės nustatymas, taikant tuos skaičius kaip funkcijos argumentus, ir tyrimas statistiniais metodais. Šis metodas taikomas ir automatiniame testų rinkinių sudaryme [8] – įėjimo vektorius nėra apskaičiuojamas, o sudaromas atsitiktinai iš loginių nulių ir vienetų. Paduodant šį testinį vektorių į loginės schemos įėjimą, analizuojamas išėjimas, priklausomai nuo jo, įėjimo vektorius gali būti modifikuotas (dažniausiai, vienas iš loginių nulių paverčiamas vienetu ir atvirkščiai) arba sugeneruotas atsitiktinai iš naujo. Vienas iš galimų Monte-Carlo algoritmo taikymų automatiniam testinių vektorių sudarymui yra parodytas 1.1 lentelėje.

2.1 lentelė. Monte-Carlo algoritmo pseudokodas

Start
Select target Fault Set;
Generate initial Population;

```
Evaluate (Population);
For i=1 to (New Population = Ø) Do
While Fit < Fault Set do
For j=1 to population size Do
Select Poll Mates using roulette wheel mechanism;
Crossover (p1, p2, c1, c2);
Mutate (c1);
Mutate (c2);
Add c1 and c2 to New Population
End for;
Fit = evaluate (New Population);
Population = New Population;
End for;
Solution = Population;
End
```

Algoritmų klasifikacija, palyginimas ir analizė.

D-ALG, PODEM ir FAN algoritmai, o taip pat ir jų modifikacijos reikalauja daug procesorinio laiko, atminties bei kitų resursų (pavyzdžiui, tinklo resursų, jei problema sprendžiama lygiagrečiai keliuose kompiuteriuose), be to gali atsirasti nenumatytų sunkumų, ypač sudėtingose skaitmeninėse schemose. Iš esmės, kai kuriose dideliuose schemose šių algoritmų naudojimas jau neįmanomas arba nepraktiškas. [9]

Pseudoatsitiktinių testų vektorių rinkinių sudarymas kartu su gedimų modeliavimu yra paprasta alternatyva algoritminiems metodams. Tai įtraukia įvedimų vektorių sudarymą naudojant palyginamai greitą pseudoatsitiktinių skaičių generatorių ir gedimų modeliavimo našumą nustatant, ar gauti vektoriai aptinka norimą gedimą. Tikslinio gedimo charakteristikos turi didelę įtaką pseudoatsitiktinių testų vektorių rinkinių sudarymo efektyvumui, jis dažniausiai naudojamas testų sudarymo proceso pradžioje, tam, kad apimti lengvai aptinkamus gedimus [11], kiti gedimai po to gali būti aptikti algoritminiais metodais.

Algoritminiai testų rinkinių sudarymo metodai vadinami *deterministiniais* metodais, nes testų rinkiniai buvo sistematiškai suformuoti su garantuotu rezultatu parinktiems gedimams.

Pseudoatsitiktiniai testų rinkinių sudarymo metodai (kaip Monte-Karlo) yra *tikimybinių* metodų pavyzdžiai, nes testų rinkiniai sudaromi atsitiktinai ir jų efektyvumas paprastai patvirtinamas gedimų modeliavimu. Pseudoatsitiktinių ir algoritminių testų rinkinių sudarymo metodų kombinacija yra *mišriojo* testų rinktinių sudarymo metodo pavyzdys. Mišrieji metodai yra labai efektyvūs, nes maksimaliai išnaudoja algoritminių ir pseudoatsitiktinių metodų privalumus.

1.2 lentelėje yra parodytas aprašytų algoritminių metodų efektyvumas [10]. Kaip atskaitos taškas parinktas D-ALG algoritmo našumas, nes jis yra lėčiausias iš nagrinėtų. Šiais laikais egzistuoja daug efektyvesni metodai, kurie naudoja heuristiką, pavyzdžiui Taferthoferio metodas. Pseudoatsitiktiniai testų rinkinių sudarymo metodai šiame palyginime nedalyvauja, nes jų rezultatas yra ne visada pilnas, veikimo laikas neturi griežtos priklausomybės nuo loginės schemos sudėtingumo ir jų veikimo efektyvumas negali būti prognozuotas iš anksto.

2.1 lentelė. Algoritminių metodų efektyvumo palyginimas

Algoritmas	Efektyvumas
D-ALG	1
PODEM	7
FAN	23

2.1.5. ISCAS'85 ir ISCAS'89 ATPG efektyvumo įvertinimo schemos

Automatinių testinių vektorių rinkinių sudarymo algoritmo efektyvumas yra šios probleminės srities kertinis akmuo. 1980-ųjų metų viduryje, kai skaitmeninės pasidarė pigesnės ir sudėtingesnės, atsirado poreikis etaloninių schemų modelių, kurie padėtų įvertinti metodų veikimo laiką, aptiktų gedimų procentą, šių charakteristikų priklausomybę nuo įėjimų ir išėjimų skaičių, loginių elementų skaičių ir ryšių tarp jų. Tokių schemų rinkiniai buvo pasiūlyti ISCAS grupe.

ISCAS'85 schemų rinkinys

1985 metais ISCAS grupė pasiūlė schemų rinkinį, kuriame buvo 10 kombinacinių schemų, visos skirtingos paskirties, sudėtingumo ir dydžio. Šios schemos ir dabar tarnauja kaip kombinacinių schemų testų sudarymo efektyvumo nustatymo priemonės. Schemų parametrai nurodyti 1.3 lentelėje.

2.3 lentelė. ISCAS‘85 schemų parametrai

Vardas	Funkcija	Loginių elementų skaičius	Įėjimų skaičius	Išėjimų skaičius	Gedimų skaičius
C432	Priority Decoder	160	36	7	524
C499	ECAT	202	41	32	758
C880	ALU and Control	383	60	26	942
C1355	ECAT	546	41	32	1574
C1908	ECAT	880	33	25	1879
C2670	ALU and Control	1193	233	140	2747
C3540	ALU and Control	1669	50	22	3428
C5315	ALU and Selector	2307	178	123	5350
C6288	16-bit Multiplier	2406	32	32	7744
C7552	ALU and Control	3512	207	108	7550

ISCAS‘89 schemų rinkinys

1989 metais ISCAS išleido papildomą schemų rinkinį, kurį sudaro 25 nuoseklios schemos. 1.4 lentelėje nurodyti ISCAS89 schemų parametrai.

2.4 lentelė. ISCAS‘89 schemų parametrai

Schemos vardas	Giluma	Trigerių skaičius	Įėjimų skaičius	Išėjimų skaičius	Gedimų skaičius
s208	11	8	11	2	215
s298	8	14	3	6	308
s344	6	15	9	11	342
s349	6	15	9	11	350

s382	11	21	3	6	399
s386	5	6	7	7	384
s400	11	21	3	6	426
s420	19	16	19	2	430
s444	11	21	3	6	474
s510	2	6	19	7	564
s526	11	21	3	6	555
s641	6	19	35	24	467
s713	6	19	35	23	581
s820	4	5	18	19	850
s832	4	5	18	19	870
s838	21	32	35	2	857
s953	3	29	16	23	1079
s1196	4	18	14	14	1242
s1238	4	18	14	14	1355
s1423	10	74	17	5	1515
s1488	5	6	8	19	1486
s1494	5	6	8	19	1506
s5378	36	179	35	49	4603
s9234	26	211	19	22	3938
s35932	35	1728	35	320	39094

2.2. Egzistuojantys sprendimai

2.2.1. S25 Fastpass ATG

ATGET programinė sistema automatiškai sudaro funkcinius testų vektorių rinkinius PAL, FPLA, FPGA, FPLS ir kitų architektūrų skaitmeninėms schemoms. S25 FASTpass – tai efektyvus žemos kainos paprastų testų generatorius, kuris aptinka dažniausiai gamyboje pasitaikančius gedimus.

Jo privalumai:

- Žema kaina
- Greitas testų sudarymas
- Jis yra idealus tokiems gedimams, kaip neužprogramuotos dalys, klaidingos krypties moduliai, neteisingų loginių elementų išdėstymas, katastrofinis prietaiso gedimas, aptikti.
- Kokybiški testų vektoriai
- Funkcionalus, nereikia papildomos informacijos.
- Automatinis konfliktų detektorius
- Yra lengvai įsisavinamas
- Meniu vartotojo sąsaja
- Darbas pagal scenarijų
- Pilnas vartotojo vadovas
- Galimybė peržiūrėti neaptinkamų gedimų sąrašą

2.2.2. Verilog XL

Cadence kompanijos Verilog XL – tai ne tik ATPG skirta programinė sistema. Jos funkcijos yra žymiai platesnės – nuo schemos laiko veikimo emuliacijos iki pramoninės kokybės schemų testavimo.

Sistemos privalumai:

- Aptinka visus gedimus, kuriuos galima aptikti
- Grafinė vartotojo sąsaja
- VHDL, Verilog ir kitų schemų aprašymų kalbų palaikymas
- Integracija su daugelių schemos testavimo, emuliacijos ir projektavimo įrankiais
- Pramoninė programinės įrangos kokybė

2.3. Įgyvendinimo problemos

2.3.1. Uždavinio sunkumas.

1975 m. buvo įrodyta, kad klasikinis automatinio testų rinkinių sudarymo uždavinys yra NP-pilnas [12]. Tai reiškia, kad pilnas testų sudarymo laikas priklauso nuo įėjimų skaičiaus nepolinomiškai. Buvo parodyta [13], kad uždavinio sunkumas yra

$O(m*2^n)$,

kur m – gedimų skaičius, o n – įėjimų skaičius.

Tam. kad išvengti per didėlio skaičiavimo laiko, šiame darbe bus naudojami ne determenistiniai, o tikimybiniai metodai.

2.3.2. Vartotojo sąsajos sudarymas

Vartotojo sąsaja bus sudaryta su UNIX suderinamoje sistemoje, naudojant PHP scenarijus.

2.3.3. Vartotojų valdymas

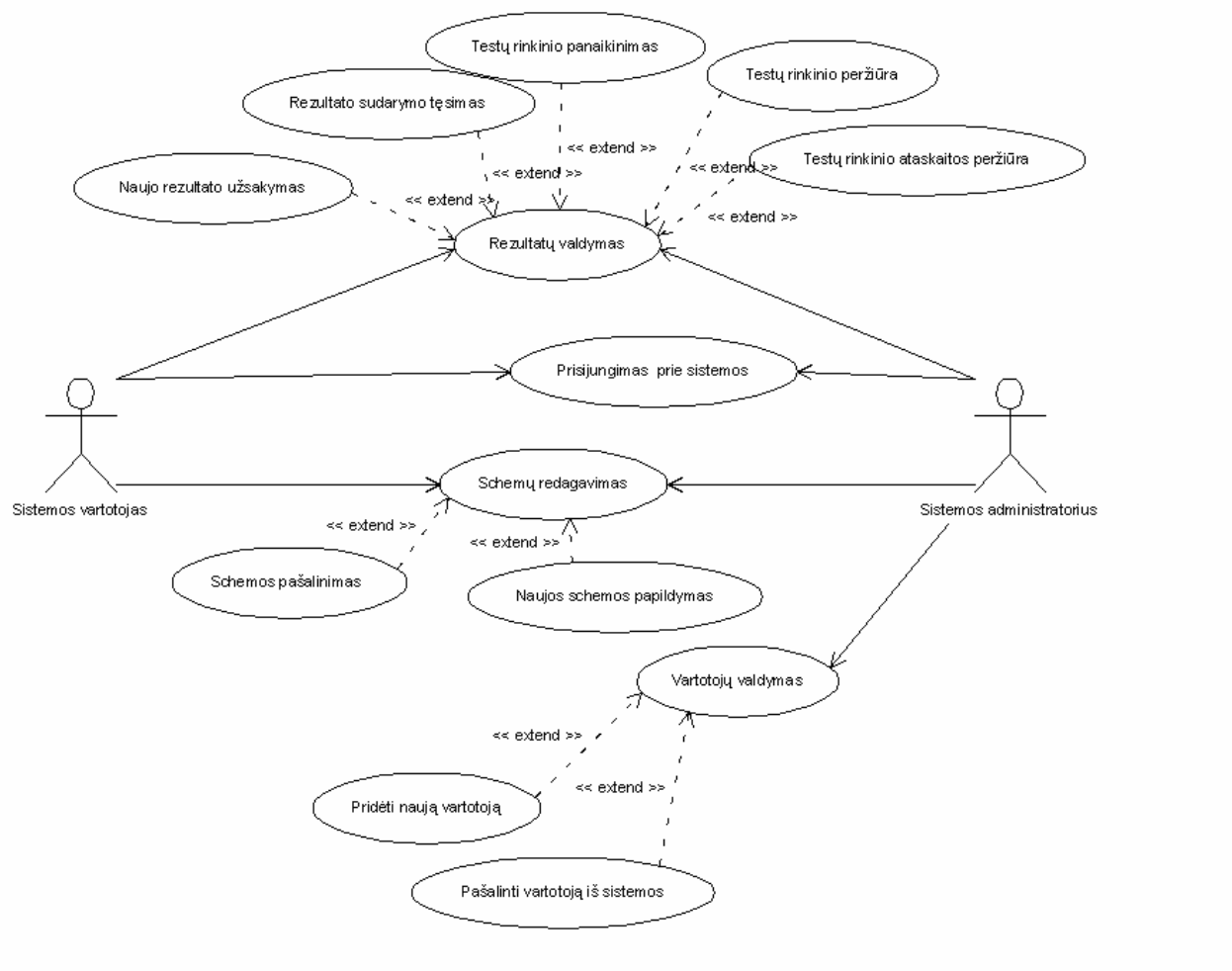
Vartotojų informacija bus saugoma duomenų bazių valdymo sistemoje. Kaip tokia sistema parinkta MySQL.

3. PROJEKTINĖ DALIS

3.1. Sistemos aprašymas.

Internetinė skaitmeninių schemų testų rinkinių sudarymo programinė sistema – tai sistema, kuri leidžia vartotojui prisijungus prie interneto, sukurti jam pateiktai schemai testinių vektorių rinkinį. kuris pakankamai gerai aptiktų visus galimus gedimus schemoje. Projekto tikslas – sukurti tokią sistemą, įdiegti ją, pateikti jos dokumentaciją vartotojams, administratoriams ir prižiūrėtojams. Projektas gali būti naudojamas komercinėse įmonėse skaitmeninių schemų gamybos proceso tobulinime ir schemų projektavime. Taip pat jį gali pasinaudoti mokslo įstaigos, kaip priemonė moksliniams darbams atlikti arba kaip studentų apmokymo priemonė.

3.2. Panaudos atvejai



3.1 pav. Panaudos atvejų diagrama

3.2.1. Panaudos atvejų aktoriai:

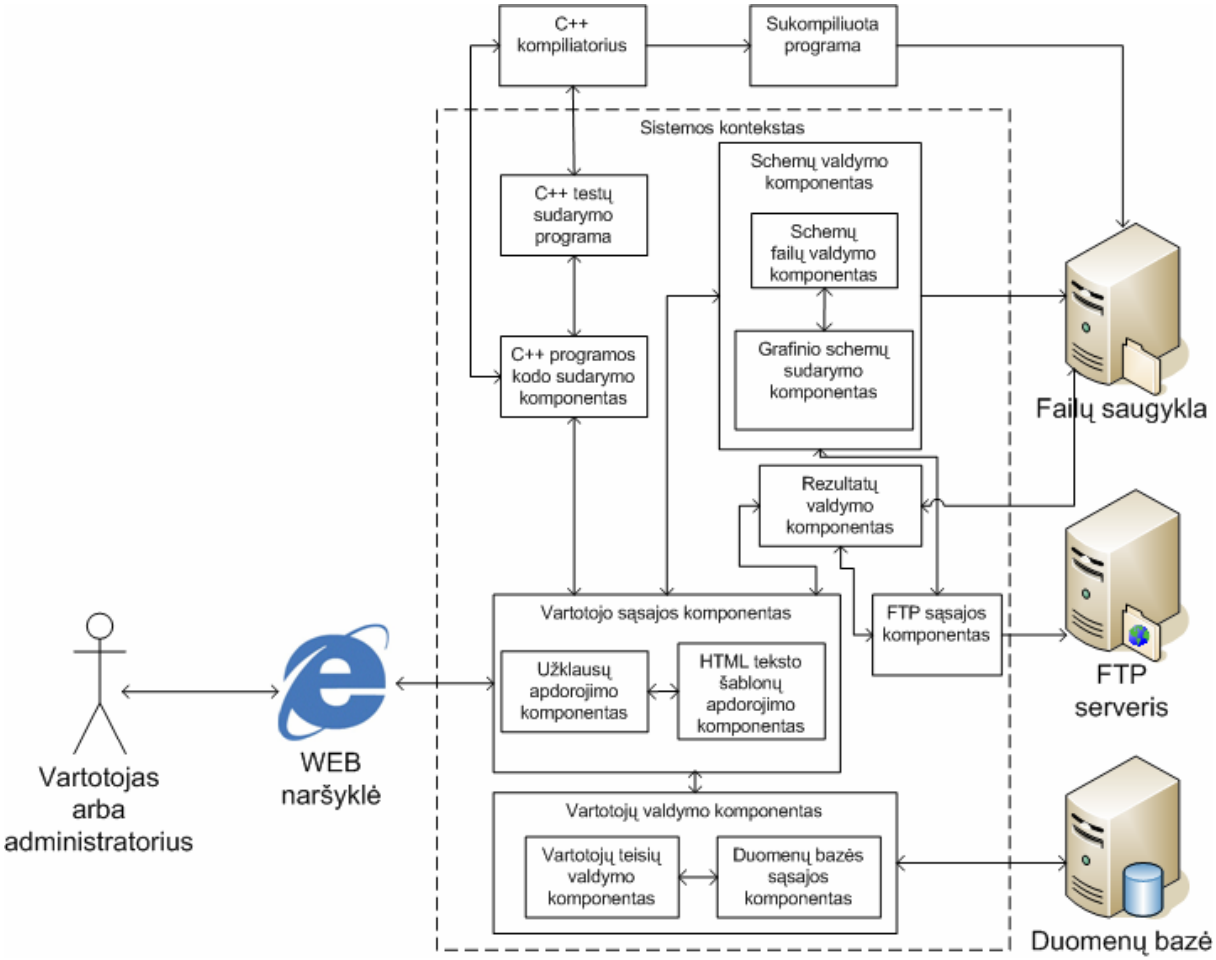
- Sistemos vartotojas – tai tiesiogiai su sistema dirbantis žmogus, kuris naudoja visas jos funkcijas, tačiau negali valdyti vartotojus.
- Sistemos administratorius– tai vartotojas, kuris prižiūri sistemą, išskiria vartotojams teisės, sukuria arba trina vartotojų užrašus.

3.2.2. Panaudos atvejų tikslai:

- Prisijungimas prie sistemos
    - Tikslas – patvirtinti, ar duota vartotojo vardas/slaptažodis pora atitinka duomenų bazės įrašą

- Schemos pašalinimas
    - Tikslas – pašalinti vartotojui nebereikalingą schemą
- Naujos schemos papildymas
    - Tikslas – papildyti vartotojui schemų sąrašą
- Naujo rezultato užsakymas
    - Tikslas – sukompiliuoti ir paleisti programą naujiems rezultatams gauti
- Rezultato sudarymo tęsimas
    - Tikslas – sukompiliuoti ir paleisti programą naujiems papildomiems rezultatams gauti
- Testų rinkinio panaikinimas
    - Tikslas – pašalinti testų rinkinį
- Testų rinkinio peržiūra
    - Tikslas – peržiūrėti testų rinkinį
- Testų rinkinio ataskaitos peržiūra
    - Tikslas – peržiūrėti testų rinkinio ataskaitą
- Testų rinkinio papildymas į FTP serverį
    - Tikslas – papildyti duotą testų rinkinį FTP serveryje jo tolimesnei analizei
- Laisvas prisijungimas
    - Tikslas – papildyti sistemos vartotojų sąrašą
- Pašalinti vartotoją iš sistemos (tik administratoriams)
    - Tikslas – pašalinti vartotoją iš sistemos vartotojų sąrašo
- Laisvo prisijungimo patvirtinimas (tik administratoriams)
    - Tikslas – patvirtinti laisvą prisijungimą

3.3. Sistemos komponentai ir jų statinis vaizdas



3.2 pav. Sistemos komponentų diagrama

3.3.1. Trumpas sistemos konteksto komponentų aprašymas

- Vartotojo sąsajos komponentas – tai posistemė, kuri gauna HTTP protokolu perduotas užklausas ir grąžina HTML formatuotus dokumentus. Šis komponentas sąveikauja su vartotoju tarp vartotojo kompiuterio įdiegtą WEB naršyklę per HTTP protokolą internetu
- Vartotojų valdymo komponentas atsako už vartotojų autorizaciją, teisių valdymą ir saugojimą
- Schemų valdymo komponentas – tai posistemė, kuri leidžia vartotojui valdyti schemų failus arba sukurti savo schemą su grafiniu redaktoriumi

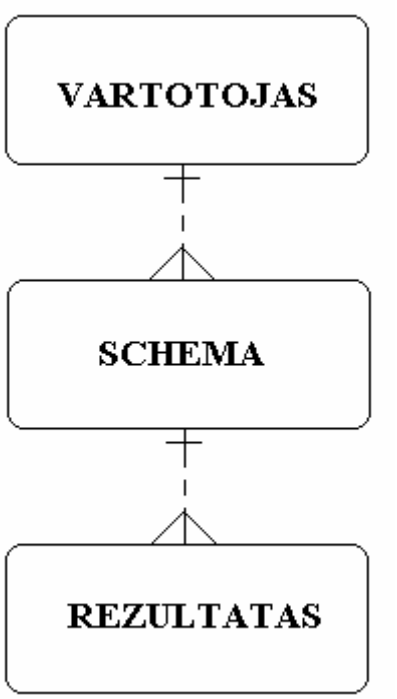
- C++ programos kodo sudarymo komponentas – tai posistemė, kuri sudaro C++ programos kodo nustatymų .h antraštę bei valdo programos kompiliaciją ir paleidimą
- C++ testų sudarymo programa – tai C++ kalba parašyta programos dalis, kuri kartu su nustatymų .h failu ir schemos .h failu sukompiliuojama į programą, kuri sudaro testinius rinkinius. Ši programa turi testų sudarymo algoritmus, ataskaitos sudarymo funkcijas, bet neįtraukia pačios skaitmenines schemos logikos kodo
- Rezultatų valdymo komponentas – tai posistemė, kuri leidžia daryti operacijas su rezultatų failais
- FTP sąsajos komponentas atsako už schemų ir rezultatų patalpinimą į FTP serverį

3.3.2. Išoriniai komponentai ir aktoriai

- Vartotojas arba administratorius – tai aktorius, kuriam sukurta sistema, turintis prieigą prie interneto
- WEB naršyklė – tai programinė įranga, kuri per HTTP interneto protokolą gauna iš serverio HTML formatuotą hipertekstą ir atvaizduoja jį grafiškai bei pagal vartotojo atliktus veiksmus suformuoja HTTP užklausas ir perduoda jas į serverį
- C++ kompiliatorius – tai programa, kuri pagal duotą C++ kalba parašytą tekstą bei nustatymus suformuoja paleidžiamą programą arba pranešą apie kompiliacijos klaidą
- Failų saugykla – tai katalogas, kurio nustatymai leidžia PHP scenarijams rašyti į jį, skaityti iš jo ir paleisti jame programas
- FTP serveris – tai FTP protokolo standartą palaikantis serveris, prie kurio sistema gali prisijungti ir papildyti jį failais
- Duomenų bazė – tai MySQL serveris, kuriame sukurtas sistemos vartotojas ir reikalingos lentelės

3.3.3. Duomenys

Sistemoje duomenys yra 3 tipų – vartotojas, schema ir rezultatas (rezultatas – tai testinių vektorių sąrašas su ataskaita). Jų hierarchija yra pavaizduota ER diagramoje.



3.3 pav. ER diagrama

Vartotojo vardas – tai didelės arba mažos lotyniškos raidės nuo a iki z ir skaičiai, ne mažiau ; simbolių, bet ir ne daugiau 40.

Vartotojo slaptažodžiui taikomi tokie pat apribojimai, kaip ir vartotojo vardui.

Testavimo schemos yra pateikiamos .h C++ failuose. Faile turi būti tik viena funkcija, jos bendras aprašymas:

void funkcija(char *gg,char *og,int in,int on,int *gerai)

Ši funkcija simuliuoja mikroschemos veikimą. Jos parametrai yra:

char *gg – įvedimo signalų testinis vektorius

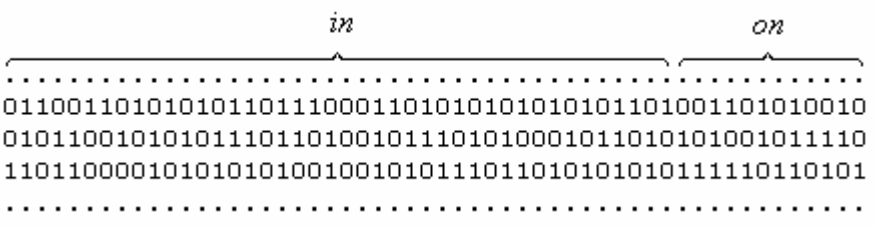
char *og – išvedimo signalų testinis vektorius

int in – schemos įvedimų signalų skaičius

int on – schemos išvedimų signalų skaičius

int *gerai – schemos sėkmingo veikimo rodiklis

Rezultatų failai pateikiami tekstiniuose failuose šiame pavidale – kiekvienas testinis vektorius užrašomas atskiroje eilutėje. Žemas ir aukštas loginiai lygiai yra aprašomi atitinkamais simboliais „0“ ir „1“. Testinis vektorius aprašomas tokia tvarka – pirma eina įvedimo signalų testinio vektoriaus dalis, o po jo - išvedimo signalų testinio vektoriaus dalis. Žemiau pateikta rezultatų failo santrauka.



3.4 pav. Rezultatų failo santrauka

Rezultatų ataskaitos yra pateiktos tekstinių failų pavidale.

4. TYRIMO DALIS

4.1. Tyrimo objektas

Algoritmas yra automatinio skaitmeninių schemų testų formavimo esmė. Kuo geresnis algoritmas, tuo didesnis klaidų aptikimo procentas ir mažesne poveikių aibė. Tai yra, norint optimizuoti sistemą, reikia patobulinti vektorių atrinkimo algoritmą. Kadangi sukurta sistema leidžia eksperimentuoti ne su vienu, o su keliais skirtingais algoritmais, galima ištirti jų kokybę.

4.2. Algoritmų kokybės rodikliai

Tam, kad išanalizuoti skirtingus algoritmo veikimo rezultatus, reikia parinkti vieną fiksuota iš parametrų: veikimo laikas, perrinktų vektorių skaičius, atrinktų vektorių skaičius, klaidų aptikimo procentas. Šiame tyrime priimamas visiems algoritmams vienodas perrinktų vektorių skaičius, kuris priklauso nuo schemoje esančių loginių elementų skaičiaus bei jų pajungimo sudėtingumo.

Esant vienodam perrinktų vektorių skaičiui, kiti parametrai yra:

- Veikimo laikas – nes ATPG algoritmai labai apkrauna procesorių, jų spartumas yra esminis faktorius. Kuo jis mažesnis, tuo geresnis algoritmas.
- Atrinktų vektorių skaičius – iš vienos pusės, didesnis vektorių skaičius įtakoja klaidų aptikimo procentui, iš kitos – geras algoritmas turi formuoti kuo mažesnį rinkinį su kuo didesniu klaidų aptikimo procentu.
- Klaidų aptikimo procentas – tai svarbiausias automatinio skaitmeninių schemų testų formavimo algoritmo rezultato matas, kuo jis artimesnis prie 100%, tuo algoritmas yra geresnis.

Po šių rodiklių analizės buvo nuspręsta naudoti tokį parametrą – santykį „Atrinktų vektorių skaičius“ / „Perrinktų vektorių skaičius“. Laiko matavimai tyrime nenaudojami, nes serveryje be sistemos veikia pakankamai daug kitų programų sistemų, kurie įtakoja programai

išskiriamų resursų kiekį. Klaidų aptikimo procentas nėra svarbus, nes konvergavimo stadijose („prisisotinimo momentu“) jis yra artimas (arba lygus) 100% visiems metodams su nežymimais svyravimais.

4.3. Tyrimo procesas

4.3.1. Naudojamos schemos ir poveikių kiekiai

Tyrime yra naudojamos 10 ISCAS'85 schemos, išvardintos 2.1.5 skyriuje. Kiekviena iš 10 schemų yra testuojama kiekvienu algoritmu, kol atrinktų poveikių kiekis auga. Žingsnių perrinktų elementų kiekiai yra parodyti 4.1 lentelėje.

4.1 lentelė – Vektorių kiekio gradacija pagal žingsnį

Žingsnis	Perrinktų vektorių kiekis
1	50
2	100
3	200
4	500
5	750
6	1000
7	1500
8	2000
9	5000
10	10000
11	15000
12	25000
13	50000
14	100000
15	200000
16	300000
17	400000

4.3.2. Palyginami algoritmai

T1

Šis algoritmas yra paremtas dvimatės matricos pildymu. Šio algoritmo pseudokodas:

1. T = { ∅ }; $h^{11}_{ij} = 0$; $h^{10}_{ij} = 0$; $h^{01}_{ij} = 0$; $h^{00}_{ij} = 0$; visiems i = 1,n ir j = 1,m
2. V = < v_1,v_2, … v_i, … ,v_n>;.
3. R := f (V); R = < r_1,r_2, … ,r_j, … r_m>.;
4. *do* i = 1 to n ;
5. $v_i := \neg v_i$;
6. S := f (V); S = < s_1,s_2, … ,s_j, … s_m>.;
7. $v_i := \neg v_i$;
8. *do* j = 1 to m ;
9. *if* $r_j \neq s_j$ *then*
10. if s_j = 1 then
11. *if* v_i = 0 *then*
12. *if* h^{11}_{ij} = 0 *then*
13. T := T ∪ { V }; h^{11}_{ij} := 1;
14. else
15. *if* h^{01}_{ij} = 0 *then*
16. T := T ∪ { V }; h^{01}_{ij} := 1;
17. *endif;*
18. *else*
19. *if* v_i = 0 *then*
20. *if* h^{10}_{ij} = 0 *then*
21. T := T ∪ { V }; h^{10}_{ij} := 1;
22. else
23. *if* h^{00}_{ij} = 0 *then*
24. T := T ∪ { V }; h^{00}_{ij} := 1;
25. *endif;*
26. *endif*;
27. *endif;*

28. *enddo*;
29. *enddo*;

C5 algoritmas.

Šis algoritmas yra paremtas termais. Jo aprašymas [14]:

Modelio funkcija turi n dvejetainių įėjimų ir m dvejetainių išėjimų. Įėjimo poveikis apibrėžiamas vektoriumi $A=<a_1,a_2, \ldots,a_i, \ldots,a_n>$, kur $a_i \in \{0,1\}$. Funkcijos reakcija apibrėžiama vektoriumi $B=<b_1,b_2, \ldots,b_j, \ldots,b_m>$, $b_j \in \{0,1\}$. Bendru atveju B=f(A). Atsitiktinai generuojama PK poveikių. Įėjimo poveikis atrenkamas į aibė V1, jei pagal šį poveikį surandamas bent vienas išėjimų funkcijų termas.

Kiekvienam įėjimo poveikiui paskaičiuojama matrica $X= ||x_{i,j}||_{n \times m}$. Matricos elementas xi,j=1, jei pakeitus poveikio įėjimo ai reikšmę į priešingą, išėjimo b_j reikšmė taip pat pasikeičia į priešingą. Stulpelių nenuliniai elementai apsprendžia termą. Įėjimo i tiesioginis (atvirkštinis) kintamasis priklauso išėjimo j tiesioginės funkcijos termui , jei $x_{i,j}=1$, $a_i=1(a_i=0)$, o $b_j=1$. Įėjimo i tiesioginis (atvirkštinis) kintamasis priklauso išėjimo j atvirkštinės funkcijos termui , jei $x_{i,j}=1$, $a_i=1(a_i=0)$, o bj=0.Funkcijos termas nėra esminis, jei kitas termas turi jo visus kintamuosius. Nagrinėjant įėjimo poveikius surandami tik esminiai termai. Atrinkimo metu termai vis pasipildo

4.4. Papildomos algoritmų palyginimo sąlygos

Tam, kad užtikrinti eksperimento atkartojimą, reikia nustatyti pseudoatsitiktinių skaičių generatorių į vienodą pradinę būseną prieš pradedant formuoti atsitiktinius vektorius.

4.5. Tyrimo rezultatų saugojimas

Tyrimo rezultatai yra saugomi ataskaitose, kuriose nurodyta:

- Schemos pavadinimas
- Testų rinkinių formavimo pradžios laikas

- Testų rinkinių formavimo pabaigos laikas
- Perrinktų vektorių skaičius
- Atrinktų vektorių skaičius
- Atrinkimo algoritmas

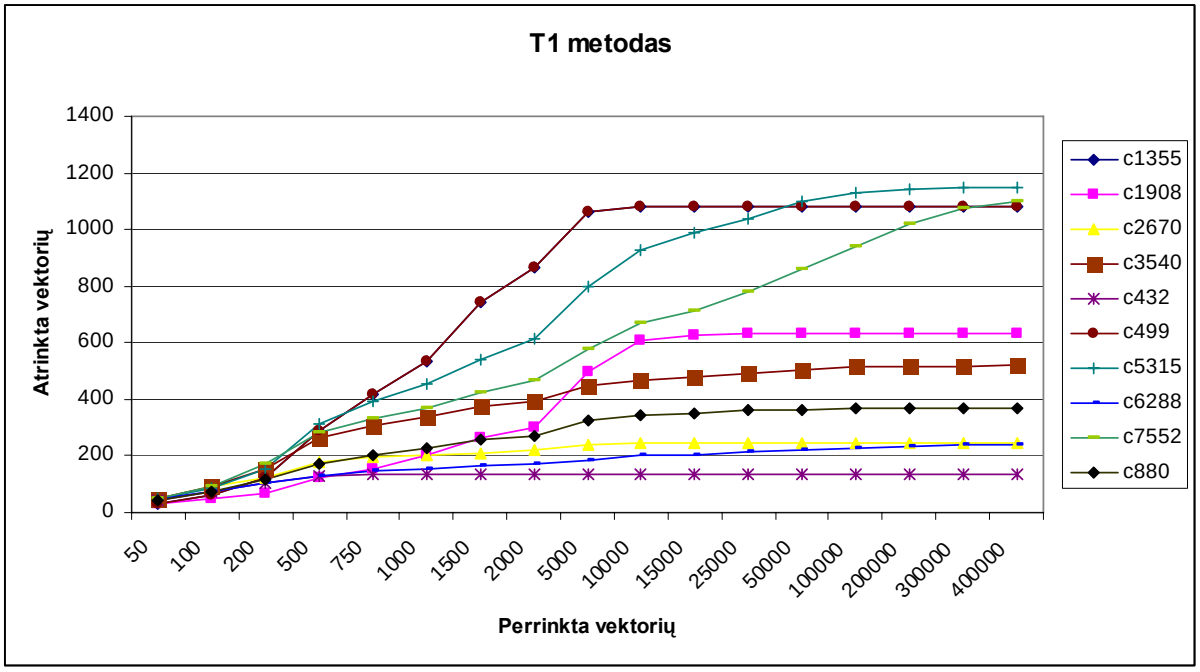
4.6. Kiti galimi patobulinimai

Kadangi sistema labai apkrauna procesorių, vienas iš galimų patobulinimų būtų lygiagrečių procesorių (arba atskirų kompiuterių) panaudojimas.

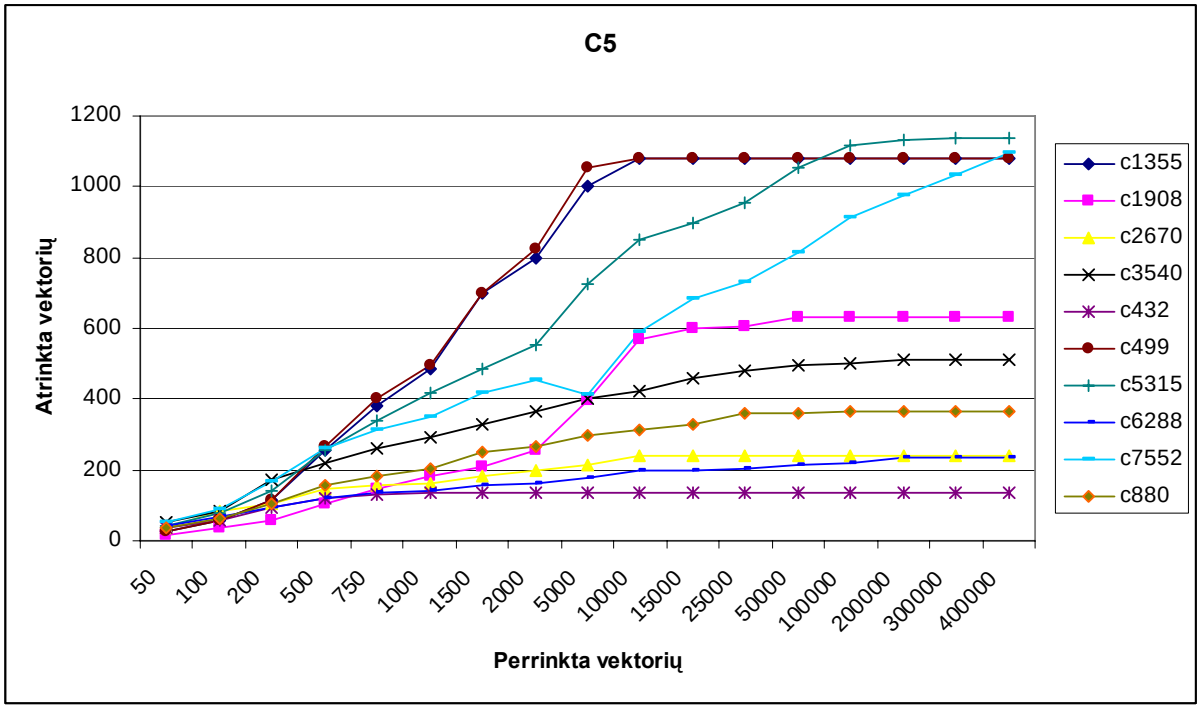
5. EKSPERIMENTINĖ DALIS

5.1. Rezultatai

Pagal 4 skyriuje aprašytą metodiką gauti testavimo rezultatai yra pateikti 9.1 lentelėje bei pavaizduoti 5.1 ir 5.2 pav.

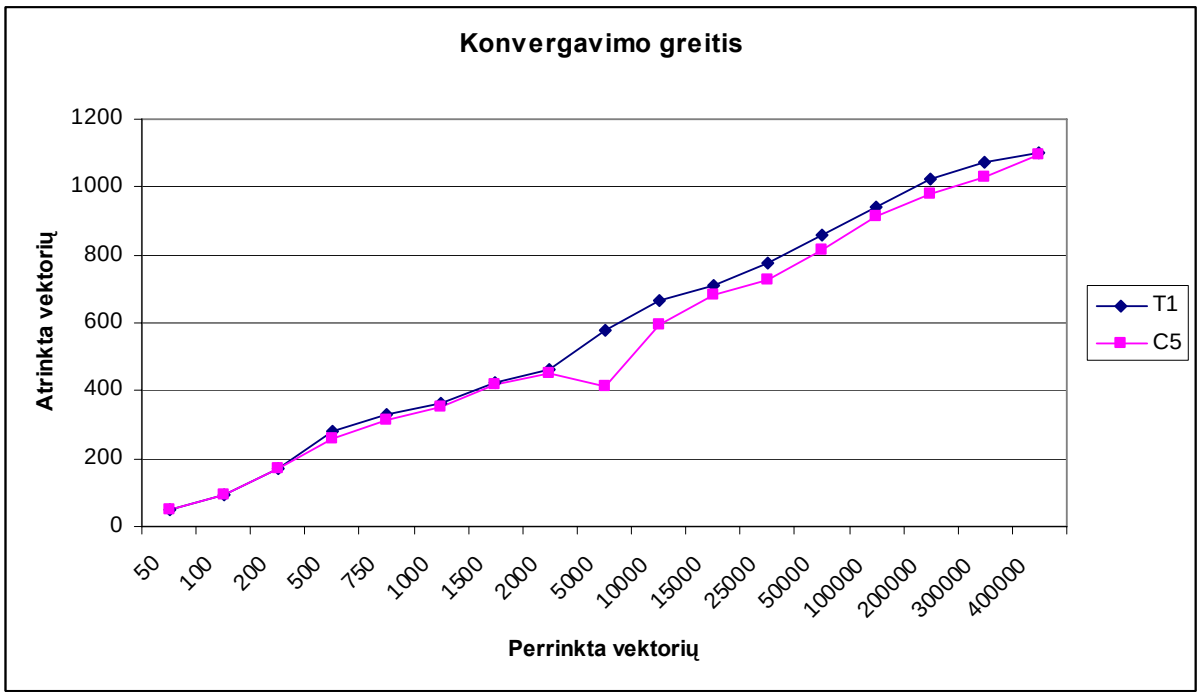


5.1 pav. T1 metodo rezultatų suvestinė



5.2 pav. C5 metodo rezultatų suvestinė

Pagal šiuos grafikus matoma, kad abu metodai pakankamai gerai konverguoja, tačiau kokybės nustatymui reikia palyginti konvergavimo eigą, parodyta pav. 5.3.



5.2 pav. C5 metodo rezultatų suvestinė

Pagal ši grafiką akivaizdu, kad T1 metodas konverguoja stabiliau, bet testavimo gale duoda daugiau vektorių, negu C5 metodas.

6. IŠVADOS

Kaip parodė tyrimai, skirtingais metodais paremti algoritmai, pagal atrinktų skaitmeninių schemų testinių vektorių skaičių santykį su perrinktų vektorių skaičių, veikia skirtingai.

Dvimatės matricos pildymo algoritmai konverguoja greičiau, tačiau jų rezultatuose yra pakankamai daug perteklinių testinių vektorių. Juos galima naudoti, kai reikia aptikti kuo daugiau galimų skaitmeninės schemos gedimų ir padidėjęs testavimo laikas nėra esminis faktorius.

Termų paremti algoritmai konverguoja lėčiau, tačiau pagal savo prigimtį jie beveik neįneša perteklinių vektorių į rezultatus, dėl to jais sudaryti skaitmeninių schemų testiniai rinkiniai dažniausiai yra mažesni už matricines.

Esant dideliam skaičiui skaitmeninės schemos loginių elementų, įėjimų ir išėjimų, testinių vektorių rinkinio sudarymas yra labai proceso resursų apkraunanti veikla, todėl reikia siekti algoritmų lygiagretinimo.

7. LITERATŪRA

1. T. R. Kuphaldt. Lessons In Electric Circuits -- Volume IV (Digital) *[Žiūrėta 2005 10 28]*, prieiga internete http://www.ibiblio.org/obp/electricCircuits/Digital/

2. F. Corno, E. Sánchez, M. S. Reorda, G. Squillero. Automatic Test Program Generation: A Case Study *[Žiūrėta 2005 11 08]*, prieiga internete http://www.cad.polito.it/pap/db/dt2004.pdf

3. P. Mazumder. Combinational Automatic Test-Pattern Generation (ATPG) Basics *[Žiūrėta 2005 10 30]*, prieiga internete http://www.eecs.umich.edu/~mazum/F02/lectures/lecmazum4.pdf

4. A. V. Gomes. ALTERNATE TEST GENERATION FOR DETECTION OF PARAMETRIC FAULTS *[Žiūrėta 2005 11 10]*, prieiga internete http://etd.gatech.edu/theses/available/etd-11252003-120234/unrestricted/gomes_alfred_v_200312.pdf

5. M. J. S. Smith. Automatic Test-Pattern Generation *[Žiūrėta 2005 11 12]*, prieiga internete http://www-ee.eng.hawaii.edu/~msmith/ASICs/HTML/Book/CH14/CH14.5.htm

6. Li-C. Wang. More On ATPG and Fault Coverages *[Žiūrėta 2005 11 12]*, prieiga internete http://dropzone.tamu.edu/~lwang/ee689/lec06/slides.pdf

7. B. V. M. Mohan, B. S. Reddy. VLSI Testing Tool *[Žiūrėta 2005 11 12]*, prieiga internete http://students.iiit.net/~mohanbvm/VlsiTestingTool.htm

8. P. Kalpana, K. Gunavathi. Fault oriented Test Pattern Generator for Digital to Analog converters *[Žiūrėta 2005 11 13]*, prieiga internete http://www.acadjournal.com/2004/V13/Part1/p3/

9. S. A. Syzgenda, A. A. Lekkos. Integrated Techniques for Functional and Gate-Level Digital Logic Simulation, Proc. 10th Design Automation Conf., 1973, pp. 159–172.

10. T. Kirkland, M.R. Mercer. A Topological Search Algorithm for ATPG. 24th Conference on Design Automation, 1987. pp 502 – 508. *[Žiūrėta 2007 04 13]*, prieiga internete http://ieeexplore.ieee.org/iel5/10573/33450/01586273.pdf?arnumber=1586273

11. A. Miczo. Digital Logic Testing and Simulation. Second Edition. Wiley and Sons, 2003. pp 165-231.

12. C. Patel. Design Verification & Testing: ATPG *[Žiūrėta 2005 11 13]*, prieiga internete http://www.csee.umbc.edu/~cpatel2/links/418/lectures/chap7_lect08_atpg.pdf

13. J. Wingfield. APPROACHES TO TEST SET GENERATION USING BINARY DECISION DIAGRAMS *[Žiūrėta 2005 11 01]*, prieiga internete https://txspace.tamu.edu/bitstream/1969.1/20/1/etd-tamu-2003C-CPEN-Wingfield-1.pdf

14. R. Šeinauskas. Užduotys 2005. *[Žiūrėta 2007 04 01]*, prieiga internete http://kopustas.elen.ktu.lt/~rsei/KTUIII/KTUIII.doc

8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

ATPG *Automatic Test Pattern Generation* – Automatinis testų rinikinių sudarymas

FAN *Fan-out Oriented ATPG Algorithm* – Įšėjimų orientuotas ATPG algoritmas

FPGA *Field-Programmable Gate-Array* – Programuojamas loginių mazgų masyvas

FPLA *Field-Programmable Logic Array* – Programuojamos logikos masyvas

FPLS *Field-Programmable Logic Switch* – Programuojamos logikos jungiklis

ISCAS *International Symposium on Circuits and Systems* – Tarptautinis schemų ir sistemų simpoziumas

NAND *Not AND* – "Ne ir" loginė operacija

NOR *Not OR* – "Ne arba" loginė operacija

PAL *Programmable Array Logic* – Programuojamo masyvo logika

PHP *Hypertext Preprocessor* – Giperteksto preprocesorius

PODEM *Path-Oriented DEcision Making* – Kelių išrinkimu lemtas sprendimų priemimas

VHDL *Very High-level Design Language* – Labai aukšto lygio projektavimo kalba

WWW *World Wide Web* – Žiniatinklis

XOR *Exclusive Or* – "Moduliu 2" loginė operacija

9. PRIEDAI

9.1. Rezultatai

9.1 lentelė – eksperimentų rezultatai

Schemos pavadinimas	Perrinkta vektorių	Atrinkta vektorių T1 metodu	Atrinkta vektorių C5 metodu
c1355	50	32	27
	100	63	59
	200	123	115
	500	286	256
	750	416	380
	1000	533	484
	1500	740	699
	2000	867	798
	5000	1065	1002
	10000	1082	1078
c1908	50	33	16
	100	50	37
	200	68	59
	500	124	105
	750	155	147
	1000	203	185
	1500	267	210
	2000	301	257
	5000	500	395
	10000	607	567
	15000	629	599
	25000	634	603
	50000	635	629
c2670	50	49	38
	100	91	81

	200	125	103
	500	176	148
	750	194	156
	1000	202	161
	1500	211	184
	2000	219	199
	5000	240	215
	10000	245	239
c3540	50	50	50
	100	92	85
	200	153	171
	500	261	220
	750	310	260
	1000	339	291
	1500	375	329
	2000	394	365
	5000	447	403
	10000	469	421
	15000	481	461
	25000	491	481
	50000	506	498
	100000	515	503
	200000	518	512
	300000	518	512
	400000	519	512
c432	50	46	37
	100	71	59
	200	103	94
	500	130	122
	750	135	130
	1000	137	136
c499	50	32	25
	100	63	58

	200	123	114
	500	286	265
	750	416	401
	1000	533	497
	1500	740	698
	2000	867	824
	5000	1065	1054
	10000	1082	1080
c5315	50	48	40
	100	89	80
	200	154	139
	500	313	256
	750	390	341
	1000	455	415
	1500	541	486
	2000	614	552
	5000	797	727
	10000	929	850
	15000	986	898
	25000	1039	954
	50000	1102	1053
	100000	1130	1115
	200000	1142	1131
	300000	1146	1139
	400000	1146	1139
c6288	50	49	41
	100	72	68
	200	102	95
	500	130	121
	750	147	135
	1000	154	141
	1500	163	156
	2000	169	160

	5000	187	180
	10000	201	198
	15000	204	200
	25000	213	205
	50000	222	215
	100000	229	221
	200000	235	235
	300000	238	237
c7552	50	50	50
	100	95	91
	200	173	168
	500	280	261
	750	330	315
	1000	366	350
	1500	426	416
	2000	464	452
	5000	576	412
	10000	667	592
	15000	711	685
	25000	777	729
	50000	861	812
	100000	941	913
	200000	1022	978
	300000	1072	1032
	400000	1099	1097
c880	50	45	39
	100	74	61
	200	114	102
	500	172	157
	750	203	185
	1000	227	203
	1500	258	248
	2000	273	265

	5000	327	296
	10000	344	312
	15000	348	331
	25000	362	358
	50000	364	361
	100000	366	366