TALLINNA TEHNIKAÜLIKOOL
ARVUTITEHNIKA INSTITUUT

Diplomitöö

Digitaalskeemi testvektorite hulga optimeerimine

Autor: Antti Markus
Juhendaja: Raimund Ubar

1998

Tänuavaldused

Kõigepealt tahaksin tänada oma juhendajat professor Raimund Ubarit, kes on mind toetanud ja julgustanud kõigi nende aastate jooksul, mis ma olen veetnud Disaini ja Testi Keskuse seinte vahel. Ta on alati olnud väsimatu uute ideede generaator ja suur eeskuju. Samuti tahaksin tänada teisi Disaini ja Testi Keskuse meeskonna liikmeid: Jüri Põldret, Priidu Paometsa, Eero Ivaskit, Gert Jervanit ja Marek Mandret. Nad on toonud hallivõitu argipäevadesse värvi ja on nüüdseks rohkem kui kolleegid - neist on saanud minu sõbrad. Eraldi tänud veel Jasbir S. Takhar'ile ja Daphne J. Gilbert'ile Sheffield Hallam University'st (Suurbritannias), kelle katsetulemusteta poleks see töö võimalik olnud. Ka tänan hr. Rein Kuusikut Tallinna Tehnikaülikooli Informaatika-instituudist ning hr. Guido Veinerit Tallinna Tehnikaülikooli Täienduskoolitus-keskusest teoreetiliste konsultatsioonide eest.

Sisukord

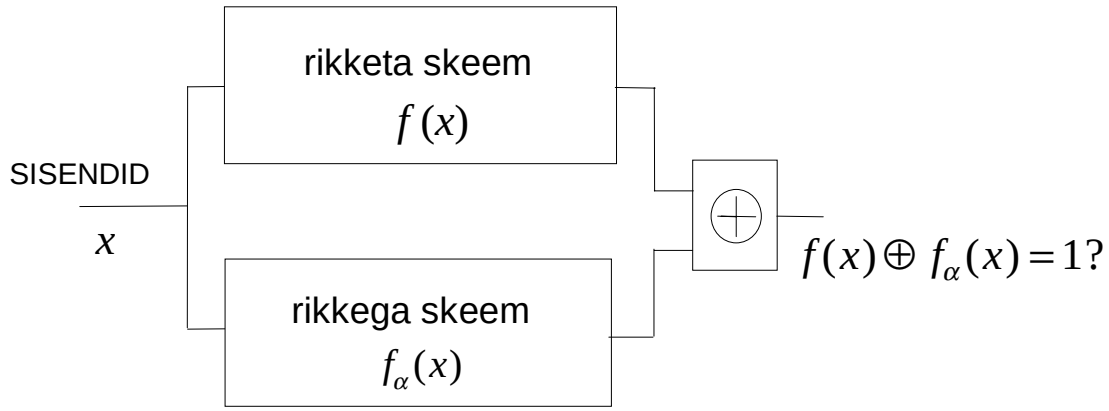
Joonised

Tabelid

Sissejuhatus

Seoses mikroelektroonika järjest suureneva invasiooniga igapäevasesse ellu on vaja projekteerida ning valmistada üha rohkem ning rohkem mikrolülitusi. Ka on jälgitav tendents elementide arvu suurenemisele mikrolülitustes (nt personaalarvutite protsessorite korral jälgitav Moore seadus). Sellega kaasneb ka üha suurenev vajadus arenenumate tehnoloogiate järele, mille abil saab kontrollida, kas tootmisliinilt maha tulnud kiip on ilma tootmisdefektideta. Selle garanteerimine (küllalt kõrge tõenäosusega) on kiibitootjate jaoks suure tähtsusega. Seda seepärast, et defektse kiibi sattumine mingisse süsteemi võib halvata kogu süsteemi või siis muuta süsteemi käitumise antud spetsifikatsioonile mittevastavaks. Ka mõjuks defektsete kiipide väljastamine hävitavalt tootja mainele. Seejuures on siiski oluline jälgida, et tootmisprotsessi järgne testimisetapp ei oleks liiga kulukas ega aeganõudev, kuna tootja võib olla sunnitud testima miljoneid mikrolülitusi aastas [5].

Mikroskeemide testimine seisneb selliste sisendväärtuste komplektide (testvektorite) leidmises, mida rikkega skeemi sisenditele rakendades erineks väljundite väärtused nendest, mis on rikketa skeemi väljunditel. Skemaatiliselt kujutatuna oleks testvektorite genereerimise ülesanne järgmine:



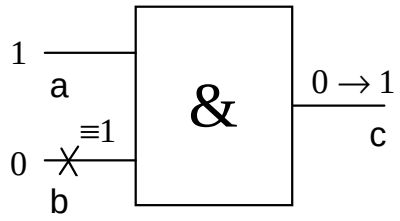
Joonis 1. Testigenereerimisülesanne

Joonisel 1 tähistab x testvektorit, *f(x)* rikketa skeemi funktsiooni ja $f_a(x)$ rikkega skeemi funktsiooni. Halvimal juhul tuleb sellise ülesande lahendamiseks rakendada *ammendavat* otsingut, mis omakorda tähendaks 2^n (n on testitava skeemi sisendite arv) testvektori läbiproovimist. Sellest järeldub aga, et tegemist on NP-täieliku ülesandega, mis on eksponentsiaalse keerukusega.

Et sellist ülesannet nagu testvektorite genereerimine edukalt automatiseerida, on vaja luua mingi formalism, mingi mudel (nn veamudel), millel saaks algoritme luua ja mis modelleeriks piisava täpsusega füüsilise taseme rikkeid, kuid samas ei oleks arvutuslikus mõttes liiga keerukas. Esimesena kerkib küsimus sellest, missugusele füüsilisele tasemele toestudes me veamudeli loome. Loomulikult on mudel seda täpsem, mida madalamal tasemele toetudes me selle loome. Et mingit loogikalülitust täiesti täpselt modelleerida, siis tuleks arvestada ka üksikute elektronide ning aatomite mõju. Loomulikult toob see kaasa tohutu andmehulga mudelis, millega töötamiseks on vaja astronoomilist arvutusvõimsust, milleni praegu veel siiski jõutud ei ole. Järgmine tase oleks mudel, mille elementaarne koostisosa oleks üks loogikalülituse transistor. See vähendab drastiliselt mudeli elementide hulka, kuid suuremate lülituste korral on see arv siiski küllalt suur. Ning et tegemist on NP-keeruka ülesandega, siis on selliste mõõtudega mudelil testide genereerimine analoogselt eelmise mudeliga ajalises mõttes liiga kulukas. Nii ongi kasutusele võetud abstraktsem mudel, mille kohta on tõestatud, et sellisel mudelil avastatavad rikked vastavad piisavalt täpselt füüsilistele riketele (st riketele pooljuhtelementides) [13]. Selliseks laialt levinud ja üsna eakaks mudeliks on konstantrikete mudel.

Sellise mudeli puhul eeldatakse, et defektid reaalses lülituses põhjustavad loogikaskeemi mingi punkti püsimist madalal ("0") või kõrgel ("1") loogilisel nivool. Kui loogikaskeemi mingi punkt püsib madalal loogilisel nivool, siis öeldakse, et tegemist on *konstant-0* (tähistatakse ≡0) rikkega. Vastasel juhul on tegemist *konstant-1* (tähistatakse ≡1) rikkega.

Konstantrikete mudelit illustreerib järgnev näide:



Joonis. 2 Konstant-1 rike JA-lüli sisendis

Näide 1. *Joonisel 2 on kujutatud JA-lüli, mille ühes sisendis b on konstant-1 rike. On kerge näha, et antud sisendväärtuste korral veaefekt levib lüli väljundisse, s.t. korras lüli ja vigase lüli väljundid omavad erinevat väärtust. Kui vaadelda antud JA-lüli testitava digitaalskeemina, siis võib öelda, et testvektor { a = 1, b = 0 } avastab konstant-1 rikke sisendis b, sest rikke mõju on selle vektori korral vaadeldav skeemi väljundis.*

Eksisteerib ka kõrgema taseme - nn käitumuslikke - mudeleid, kuid need mudelid ei suuda eriti hästi reaalse skeemi rikkeid kajastada. Selle peamiseks põhjuseks on see, et sellistes mudelites puudub info testitava loogikalülituse struktuuri kohta. Loogikalülituse struktuur aga mõjutab tugevalt skeemi testitavust ja ka testimise strateegiat üldiselt.

Kuigi füüsilised defektid võivad avalduda üheaegselt mitme konstantrikkena, modelleeritakse konstantrikete veamudelis tavaliselt vaid ühekordseid rikkeid. Esiteks on mitmekordsete rikete arvu ja vaadeldavate skeemipunktide suhe eksponentsiaalne. Suhe on leitav avaldisest

$$m = 2^{2n} - 1,$$

kus *m* on modelleeritavate mitmekordsete rikete arv ja *n* on vaadeldavate skeemipunktide arv. Teiseks on praktika näidanud, et kui leitakse test kõigile skeemi üksikutele riketele, siis avastab see suure tõenäosusega ka mitmekordsed rikked [13]. Edaspidi, rääkides riketest, peamegi silmas ühekordsete konstantrikete mudelit.

Probleemi püstitus

Diagnostikasüsteem Turbo Tester

Digitaalskeemide testimise tarkvara on välja arendatud mitmeid nimetusi. Järgnevalt on esitatud olemasoleva tarkvara lühiülevaade.

Üldjoontes võib öelda, et on kahte liiki testimistarkvara:

1) puhtalt testide genereerimisele suunatud tarkvara
2) tarkvara on osa mingi paketi funktsionaalsusest

Esimest liiki tarkvara on paindlikum oma rakendustes ja sobib hästi kasutamiseks ka õppe-eesmärkidel. Teist liiki tarkvara aga on spetsiifilisem ja on orienteeritud integratsioonile ülejäänud paketiga. Tabelis 1 on toodud enamlevinud testide genereerimise paketid (ATPG - Automatic Test Pattern Generator; e. k. ATVG - Automaatne Testvektorite Generaator) ja nende iseloomustus.

Nimetus	**Tüüp**	**Aastane maksumus (EEK)**	**Litsentsi maksumus (EEK)**	**Maksimaalne loogikalülituste arv**	**Vajalik kettaruum (MB)**
System HILO	1	8500	3454	1 000 000	200
SUNRISE*	1	Info puudub	Info puudub	Info puudub	Info puudub
SYNOPSYS	2	25 900	26 000	Piiratud virtuaalmälu mahuga	300-1024
CADENCE	2	10 200	17 270	Piiratud virtuaalmälu mahuga	1024
logicBIST	1	Info puudub	Info puudub	Info puudub	Info puudub
Mentor Graphics	2	17 270	25 905	Info puudub	Info puudub

Tabel 1. Erinevate testimistarkvara sisaldavate pakettide iseloomustus

*) SUNRISE kohta täpsem info küll puudub, kuid selle paketi hinnatase on võrreldav SYNOPSYS'e ja CADENCE'ga

Tabelist on näha, et tarkvara hind on küllalt kõrge. Kusjuures tabelis toodud hinnad kehtivad vaid haridusasutustele. Kommertslitsentsid on tihtipeale 10-1000 korda kallimad. Ka nõuavad need süsteemid reeglina võimsat riistvaralist platvormi. Nii et näiteks mingis ülikoolis loetav digitaalskeemide disaini või testimist käsitlev kursus

läheb päris kulukaks (kui kokku arvestada kulutused tark- ja riistvarale). Sellest ongi tekkinud vajadus odavate ja väikesemahuliste tarkvarapakettide järele, mis võimaldaks korraldada nii digitaalskeemide testimise kui ka disaini õpetamist. Viimastel aastatel on selliseid pakette küll arvukalt ilmunud (tänu personaalarvutite jõudluse kiirele kasvule), kuid digitaalskeemide testimise õpetamiseks vajalik tarkvara praktiliselt puudub. Selle lünga täitmiseks loodigi Turbo Tester, mis on orienteeritud digitaalskeemide testimise õpetamisele ja samas oleks kasutatav ka praktiliselt tööstuses (eriti mõnes väikefirmas) digitaalskeemide testimiseks.

Turbo Tester sisaldab järgmisi tarkvaralisi realisatsioone:

1. erinevad testigenereerimismeetodid
    - deterministlik
    - geneetiline
    - juhuslik
2. erinevaid simuleerimismeetodeid
    - staatiline
    - mitmeväärtuseline
3. vigade simuleerimist eri tüüpi digitaalskeemidel
    - kombinatsioonskeemidel
    - järjestikskeemidel
4. testvektorite hulkade minimeerimine
5. testitavuse analüüs
6. sisseehitatud enesetesti (BIST - Built-in Self Test) meetodid koos simulatsiooniga
    - BILBO (Built-In Logic Block Observer) - sisseehitatud loogikaplokkide jälgija
    - CSTP (Circular Self-Test Path) - tsirkulaarne isetestiv ahel
7. aruannete genereerimise vahend (genereerib aruandeid teiste vahendite töö tulemustest)
8. liides digitaalskeemide kirjelduskeelega EDIF 2.0.0 ja SOLO1400, DixiCAD ja ISCAS'85 väljundformaatidega

Ettekujutuse tarkvarapaketi ülesehitusest annab joonis 3.

Liides EDIF 2.0.0-ga annab võimaluse Turbo Testrit kasutada koos kaasaegsete digitaalelektroonika CAD-pakettidega nagu Synopsys, Cadence, Mentor Graphics, OrCAD, Asyl+, ViewLogic, Compass jne. Et EDIF-formaat on üldtunnustatud ja selle toetus on digitaalelektroonika disaini tarkvaras saanud *de facto* standardiks, siis saab Turbo Testerit kasutada ka koos alles turule ilmuvate digitaalelektroonika disainivahenditega.

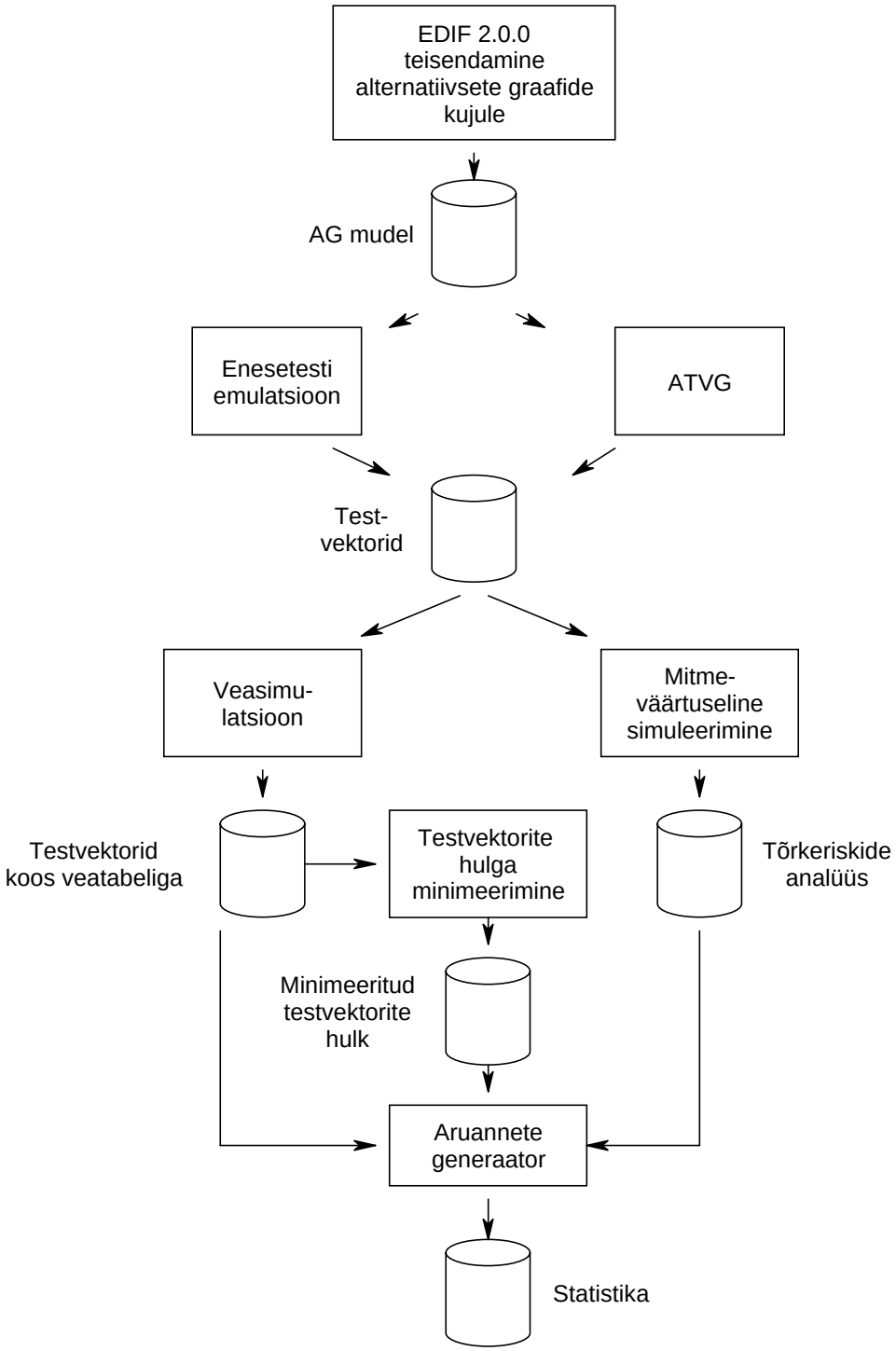
Turbo Testeri eripäraks on see, et kõik paketi vahendid kasutavad digitaalskeemide esitamiseks ühist formaati - struktuurseid alternatiivseid graafe [15, 16]. Alternatiivsete graafid pakuvad üldist matemaatilist baasi mitmete testimisülesannete lahendamiseks ühisel mudelil teatud kindla hulga protseduuride abil. Seepärast ei ole vaja luua eraldi teeke eri testimisülesannete lahendamiseks, nagu see muidu tavaks on olnud. Kasutades alternatiivseid graafidel põhinevat loogikakomponentide teeki kõikide paketi poolt lahendatavate ülesannete esitamiseks, on võimalik vähendada kogu paketi ekspluatatsioonikulusid. Samuti lihtsustab ühine lähteformaat tugevasti tarkvara väljatöötamist. Ka toetavad alternatiivsed graafid testide genereerimist loogikalülituste taseme struktuursete vigade jaoks. Alternatiivsed graafid lubavad ka loogikalülituste tasemel väljatöötatud meetodeid rakendada kõrgematel abstraktsioonitasemetel (nt. käitumuslik tase).

Väljatöötlused Turbo Testeri arenduse raames

- Osalesin C++ keeles realiseeritud objektorienteeritud teegi ClassLib loomises, kus ülesanneteks oli teegi disaini juhtimine ja osaliselt ka kodeerimine. Teegi väljatöötamisega sai võimalikuks Turbo Testeri tarkvara kaasajastamine ja muutmine riistvaralisest keskkonnast sõltumatuks. Turbo Testeri keskkonna põhjal on ilmunud rida artikleid [17, 18, 20, 22, 23]
- Eeltööde sooritamine funktsionaalse testimise vahendite lisamiseks süsteemi. Tööd hõlmasid algoritmi realiseerimist funktsionaalsete testide genereerimiseks lihtsustatud RISC-protsessoritele ja eksperimente loodud programse realisatsiooniga. Eksperimendid uurisid funktsionaalse taseme testide kehtivust madalamatel abstraktsiooni tasemetel. Tehtud töö tulemusena avaldati rida artikleid artikleid [19, 21, 24]. Tulemused näitasid, et funktsionaalsel tasemel genereeritud testid katavad edukalt madalama taseme rikked, kusjuures aega kulub selleks tunduvalt vähem kui madalal tasemel teste genereerides. Nende

eksperimentide alusel jätkunud tööd on viinud funktsionaalse testimise vahendite peatse lisamiseni Turbo Testrisse.

- Testvektorite hulkade minimeerimise algoritmi realisatsioon, mille põhjal kirjutatud artikkel "Test Set Minimization Bipartite Graphs" [25] (vt. Lisa 2), mis võeti vastu konverentsil BEC'98 avaldamiseks. Sellel artiklil põhineb ka antud töö.



Joonis 3. Turbo Testeri põhimõtteline ülesehitus

Ülesande formuleering

Pärast seda, kui testide genereerimise ülesanne on lahendatud ning mingi algoritmi põhjal on saadud mingi testvektorite hulk, tuleb kõiki neid vektoreid iga tootmisest tuleva loogikalülituse sisenditele rakendada. Selge on see, et mida vähem on testvektoreid, seda kiiremini toimub mingi lülituse testimine ja seda odavam see tootjale on. Sellest tekibki küsimus, et kas oleks võimalik vähendada testvektorite arvu nii, et testimise tõhusus üldkokkuvõttes ei väheneks. Selgub, et selline võimalus eksisteerib. See on tingitud sellest, et reeglina on üks testvektor suuteline avastama mitu riket ja mitmeid rikkeid avastatakse üheaegselt mitme vektori poolt. Selline vektorite ülekate võimaldabki testvektorite hulka edukalt vähendada.

Ülesande võib lühidalt defineerida järgmiselt: meil tuleb leida selline minimaalne vektorite hulk, mis avastaks sama palju rikkeid kui esialgne testvektorite hulk. Formaalsem ülesande püstitus on esitatud töö järgmises punktis lk. 13.

Testvektorite hulkade minimeerimise meetodid võib üldiselt kaheks jaotada:

1. staatilised
2. dünaamilised

Staatiliste meetodite korral genereeritakse kõigepealt testvektorite hulk ning püütakse saadud hulka teatud meetodiga minimeerida nii, et saavutatav veakate ei väheneks. Dünaamilised meetodid töötavad aga testvektorite genereerimise käigus. Dünaamiline meetod võimaldab mikrolülituse testimise protsessi lihtsustada - testvektorite hulga minimeerimine ei nõua enam eraldi etappi. See aga tingib tihtipeale testvektorite genereerimise algoritmi modifitseerimist, mis omakorda viib uute probleemideni, sealhulgas näiteks algoritmi aeglustumine ning selle keerukuse kasv.

Testvektorite hulga minimeerimise ülesanne on NP-täielik [14]. Et leida absoluutne miinimum, tuleks meil läbi proovida kõik võimalikud testvektorite kombinatsioonid. Et n vektori korral on meil $n!$ võimalikku kombinatsiooni, siis kasvab vajaliku töö hulk n kasvades plahvatuslikult. Seega kõige lihtsam lähenemisviis ei ole praktikas hästi rakendatav (välja arvatud väikese n korral, mis aga pakub praktilisest seisuko-

hast vähest huvi). Probleemi lahendamiseks on pakutud mitmeid erinevaid meetodeid - alates üpris elementaarsest testvektorite vastupidises järjekorras simuleerimisest kuni geneetiliste algoritmideni [3, 4, 5] ja simuleeritud lõõmutamiseni [7] välja.

Testvektorite vastupidises järjekorras simuleerimise [1, 2] idee seisneb selles, et vektorite kogumi lõpuosas asetsevad vektorid võivad juhuslikult lisaks veel avastamata riketele avastada ka juba teiste vektorite poolt avastatud rikkeid. Nagu hiljem antud töös on näidatud, ei õigusta see meetod ennast kuigi hästi.

Geneetiliste meetodite juuri tuleb otsida evolutsiooniteooriast. Selle meetodi puhul rakendatakse lahenduse otsimisel protseduure, mis oma olemuselt meenutavad loodusliku valiku ja mutatsiooni protsesse. Esiteks genereeritakse mingi juhuslik lahendite (stringide) kogum ja kontrollitakse selle kogumi sobivust tegelikuks lahendiks. Seejärel rakendatakse kogumile hinnangufunktsiooni, mis hindab iga lahendi sobivust. Siis kombineeritakse omavahel parima hinde saanud lahendid ning muteeritakse (stringi mingit osa muudetakse mingi tõenäosusega) neid ning kogu protsess kordub, kuni saavutatakse rahuldava kvaliteediga lahend või lahendite kogumi omadused protsessi käigus enam ei parane. Geneetiliste algoritmide eeliseks on nende võime leida kõik ülesande lahendid, kuid puuduseks võib lugeda nende suhteliselt aeglase töö, mis on tingitud algoritmi töötamise käigus töödeldava andmehulga suure mahuga.

Simuleeritud lõõmutamine meenutab oma põhimõtetelt geneetilisi algoritme, kuid selle meetodi puhul ei rakendata potentsiaalsete lahendite (stringide) omavahelist kombineerimist, vaid ainult nende stringide muteerimist, kusjuures protsessi lõpupoole (kui leitakse juba paremaid lahendeid), vähendatakse stringi muteerumise tõenäosust. Eelised ja puudused on siin üldjoontes samad, mis geneetilistegi algoritmide juures.

Pikalt on vaieldud, kas sellise probleemi lahendamisel oleks kasulikum rakendada nn ahneid või siis hoopis geneetilisi algoritme. Ahneks algoritmiks nimetame siin algoritmi, mis töötab etappides, kusjuures igal etapil otsustatakse järgmine samm mingi kriteeriumi põhjal, mis on sisenditel defineeritud. Põhiliseks ahnete algoritmide puuduseks on peetud nende võimetust alati globaalne miinimum leida, mis omakorda

on tingitud sellest, et selliste ülesannete otsinguruum on tavaliselt ebamugavalt suur. Antud töös ongi püütud uurida suhteliselt lihtsa ahne algoritmi tegelikku potentsiaali, rakendades vektorite valikul monotoonsete süsteemide teooria poolt pakutud lahendusvõimalusi ja ka vektorite hulga eeltöötlust, mis olulisel määral vähendab otsinguruumi.

Meetodi teoreetilised alused

Järgnevalt esitab autor loodud algoritmi teoreetilised alused. Kogu algoritm tugineb monotoonsete süsteemide teooriale, mis on välja töötatud Tallinna Tehnikaülikooli Informaatika instituudis [11, 12]. Järgnevalt lühiülevaade monotoonsete süsteemide teooria põhimõistetest. http://www.datalaundering.com/mono/aamono.htm

Definitsioon 1. Olgu antud lõplik hulk X, mille $|X|=K$, ja sellel funktsioon ω_x, mis seab igale elemendile $a \in X$ vastavusse mingi positiivse väärtuse $\omega_x(a)$. Funktsiooni ω_x nimetatakse *kaalufunktsiooniks*, kui ta on määratud suvalisel alamhulgal $X' \subseteq X$. Väärtust $\omega_x(a)$ nimetatakse elemendi $a \in X$ *kaaluks* hulgal X'.

Definitsioon 2. Hulka X koos kaalufunktsiooniga ω_x nimetatakse *süsteemiks* ja tähistatakse $\Pi=(X, \omega_x)$.

Definitsioon 3. Süsteemi $\Pi'=(X', \omega_{x'})$, kus $X' \subseteq X$, nimetatakse süsteemi $\Pi=(X, \omega_x)$ *allsüsteemiks*.

Definitsioon 4. Süsteemi $\Pi=(X, \omega_x)$ nimetatakse *monotoonseks*, kui suvalise $a \in X'\backslash\{c\}$ $(c \in X)$ korral $\omega_{x'\backslash\{c\}}(a) \leq \omega_{x'}(a)$, kus X' on suvaline hulga X alamhulk.

Definitsioon 5. Funktsiooni Q, mis seab igale monotoonse süsteemi Π alamhulgale $X' \subseteq X$ vastavusse mittenegatiivse väärtuse $Q(X')=\min_{a \in X'}[\omega_{x'}(a)]$, nimetatakse *sihifunktsiooniks*.

Monotoonse süsteemi loomiseks peavad olema rahuldatud kaks tingimust:

1. tuleb leida kaalufunktsioon $\omega_x(a)$, mis mõõdaks iga elemendi a mõju monotoonsele süsteemile X

2. peavad olema reeglid f, mille abil saab elementide kaale süsteemis uuesti arvutada, kui mingi elemendi kaal süsteemis muutub.

Lahendatava ülesande juures ongi põhiprobleemiks protsessi alguspunkti leidmine otsinguruumi punktide hulgast. Et tekitada mingisugune hinnangukriteerium, mille abil otsinguruumis orienteeruda, võtamegi appi monotoonsete süsteemide teooria.

Pakutud meetodi kirjeldus

Käesolev algoritm on oma iseloomult deterministlik ning omab teiste algoritmide ees kahte eelist:

1. mõnedel juhtudel on võimalik tõestada globaalse miinimumi saavutamist
2. antud algoritm on geneetilistest algoritmidest tunduvalt kiirem

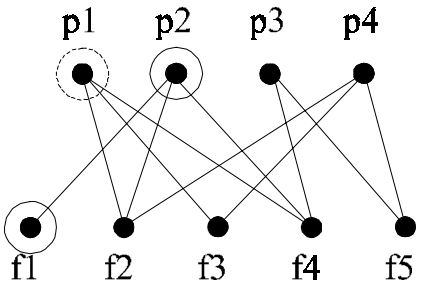
Kuigi eksperimentaalsed tulemused saadi kombinatsioonskeemide testvektorite hulkade minimeerimisel, on toodud algoritm üldisem, võimaldades sama edukalt töötada ka järjestikskeemide (sisaldavad mäluga elemente - nt registreid) testvektoritega.

	f1	f2	f3	f4	f5
p1	0	1	1	1	0
p2	1	1	0	1	0
p3	0	0	0	1	1
p4	0	1	1	0	1

Joonis 4. Testvektorite hulk

Joonisel 4 on esitatud testvektorite hulga näide. Iga rida maatriksis on testvektor ja iga veerg esitab ühte riket testitavas skeemis. Rea positsioonil *i* asetsev sümbol *'1'* tähistab seda, et antud testvektor avastab rikke *i* ja sümbol *'0'* tähistab olukorda, kus riket *i* ei avastata. Meie eesmärgiks ongi minimeerida ridade arvu maatriksis, tagades selle, et tuletatud maatriksis oleks veerge, kus on vähemalt üks *'1'*, sama palju kui algses maatriksis. Ehk kui meil on antud tõeväärtusmaatriks M=M[i, j], kus i=1…n; j=1…m ja $M[i, j] \in \{0, 1\}$ iga paari {i, j} jaoks, siis tuleb meil leida minimaalne (kardinaalsuse mõttes minimaalne) maatriksi M ridade hulk C, kus kõikide veergude 1…m jaoks, kus sisaldub väärtus 1, eksisteeriks alati element $M[i, j] \in C$, mis võrduks 1.

Sellise tõeväärtusmaatriksi saab esitada kahealuselise graafina *G=(V, E)*, millel on kaks mittekattuvat tippude hulka *P* ja *F*, kusjuures $P \cap F = \emptyset$ ja $P \cup F = V$. Igale tipule p_i ($p_i \in P$) vastab üks vektor testvektorite hulgast ning iga tipp f_i ($f_i \in F$) vastab rikkele *i* skeemis. Kui vektor *i* avastab rikke *k*, siis on tipud p_i ($p_i \in P$) ja f_k ($f_k \in F$) omavahel servaga ühendatud. Joonisel 4 esitatud maatriksist tuletatud graaf on ära toodud joonisel 5. Maatriksesituselt graafile üleminek annab meile võimaluse rakendada võimsaid ja läbiproovitud graafiteooria algoritme.



Joonis 5. Kahealuseline graaf

Kõikidele esitatud nõuetele vastava monotoonse süsteemi loomiseks defineerime kõigepealt kaalufunktsiooni:

$$\omega_V(v_i)=VAL(v_i)\ (v_i \in V) \qquad (1)$$

kus VAL tähistab tipu v_i valentsi. Kaalufunktsioon on valitud selline seepärast, et tipu p_k ($p_k \in$ P) valents näitab otseselt vektori *k* kvaliteeti rikete avastamise mõttes - mida suurem on valents, seda rohkem rikkeid vektor avastab.

Teoreem 1. Süsteem $\Pi=(V, \omega_V)$, kus V on graafi tippude hulk ja kaalufunktsioon ω_V on defineeritud valemiga (1), on monotoonne süsteem.

Tõestus. Et iga tipu valents on mittenegatiivne täisarv, siis vastab toodud süsteem definitsioonile 1. Kui me eemaldame antud graafist mingi tipu ja temaga seotud servad, siis teiste tippude valentsid kas ei muutu (kui nad ei ole ühendatud eemaldatava tipuga) või vähenevad. Seega kui me eemaldame hulgast V tipu t_k, siis tipu t_p ($t_p \in V\backslash\{t_k\}$) kaal kas väheneb või jääb samaks, s.t. $\omega_{V\backslash\{t_k\}}(t_p) \leq \omega_V(t_p)$ iga $t_p \in V\backslash\{t_k\}$ korral. Et see võrratus on sama, mis on toodud definitsioonis 4, siis võime väita, et defineeritud süsteem Π on monotoonne. M. O. T. T.

Vaatleme kõigepealt algoritmi 1, mis on koostatud tuginedes defineeritud monotoonsele süsteemile.

Algoritm 1

1) testvektorite hulk $T = Ø$
2) valime tipu p ($p \in P$), millel on suurim valents
3) eemaldame tipu p ja tipud f_i ($f_i \in F$), mis on servaga ühendatud tipuga p
4) lisame tipule p vastava testvektori hulka T
5) kui $F \neq Ø$, siis lähme punkti 2, vastasel juhul moodustab hulk T minimaalse testvektorite hulga

Algoritm 1 töötab küllaltki hästi, kuid real juhtudel ei anna see minimaalset tulemust. Kui me vaatame joonist 2, siis on näha, et antud algoritm alustaks vektorist p1 ning seega ei saa saavutada minimaalset tulemust, milleks oleks vektorid p2 ja p4.

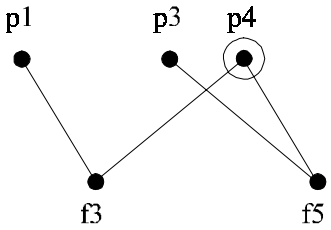
Et algoritmi parendada, lisame sinna testvektorite hulga eeltöötluse. Eeltöötluse protseduur on järgmine:

Eeltöötlus

1. kui eksisteerib selline tipp f ($f \in P$), mille valents on 1, siis läheme punkti 2, vastasel juhul lõpetame töö
2. eemaldame tipu p_k ($p_k \in P$), mis on ühendatud tipuga f ($f \in P$) ja eemaldame ka tipud f_i ($f_i \in F$), mis on ühendatud tipuga p_k.
3. lisame tipule p_k vastava testvektori hulka T

Toodud protseduuri võib nimetada ka *unikaalsete vektorite* valimiseks. Me nimetame unikaalseteks neid testvektoreid, mis ainsana avastavad rikkeid, mida ükski teine testvektor ei avasta. On ilmne, et sellised testvektorid peavad igal juhul minimaalses testvektorite hulgas sisalduma. Joonisel 6 ongi ära toodud graaf jooniselt 5, millele on rakendatud eeltöötlust. Nagu näitavad eksperimendid, on suurem osa reaalsetest testvektoritest just unikaalsed. Seega vähendab selliste testvektorite esimesena minimaal-

sesse vektorite hulka lisamine suuresti otsinguruumi. Peab ka lisama, et siin annab ülesande graafi kujul esitus palju juurde eeltöötluse lihtsustamisele - kogu ülesanne taandub ühest tippude hulgast teatud valentsiga tippude otsimisele ja on oma olemuselt lineaarse keerukusega.



Joonis 6. Kahealuseline graaf pärast eeltöötlust

Et eeltöötluse ja algoritmi 1 rakendamisel saadud testvektorite hulka veelgi vähendada, toome sisse *liiase vektori* [8] mõiste.

Definitsioon 6. Testvektor on liiane, kui selle vektori võib testvektorite hulgast eemaldada ilma avastatavate rikete arvu vähendamata.

Definitsioon 7. Reduktiks nimetatakse testvektorite hulka, milles ei ole liiaseid vektoreid.

Definitsioon 8. Globaalseks miinimumiks nimetatakse minimaalset redukti

Kuna eksperimendid liiaste vektorite olemasolu ei näidanud, siis on siin esitatud algoritmi tulemusena saadud testvektorite hulgad *reduktid* [8]. Kuid see ei tähenda veel, et need hulgad oleksid *globaalsed miinimumid.* Järgnevalt proovimegi selgitada, millised minimeeritud testvektorite hulgad on globaalsed miinimumid.

Esitame kriteeriumi, mille abil saab määrata, kas testvektorite minimeerimise tulemus on globaalne miinimum või mitte. Kriteerium on üldine ning on kasutatav ka teiste algoritmide töö hindamiseks.

Väide: Globaalne miinimum on saavutatud juhul kui minimeeritud testvektorite hulga võimsuse ja unikaalsete vektorite hulga võimsuste vahe on väiksem kui 3 ($|T| - |U| < 3$; T - minimeeritud testvektorite hulk, U - unikaalsete vektorite hulk).

Väite tõestame järgneva arutlusega. Vaatleme testvektorite hulka, milles me leidsime n unikaalset vektorit. Kui need vektorid avastavad kõik skeemis sisalduvad rikked,

siis on saavutatud globaalne miinimum, kuna ühtegi vektorit ilma avastatavate rikete arvu vähendamata eemaldada ei saa. Vastasel juhul rakendame algoritmi 1, mis tähendab, et me valime minimeeritavast testvektorite hulgast vektori, mis avastab kõige rohkem unikaalsete vektorite poolt mitteavastatavaid rikkeid. Kui selle vektori valimisel minimeeritud testvektorite hulka on kõik rikked kaetud, on jällegi tegemist globaalse miinimumiga. Seda seepärast, et sellisel juhul, kui unikaalsed vektorid kõiki rikkeid ei avasta, on minimaalne testvektorite hulk vähemalt ühe vektori võrra suurem. Kui minimeeritud testvektorite hulk sisaldab peale unikaalsete vektorite veel kahte vektorit, siis on taas tegemist globaalse miinimumiga, kuna algoritm 1 ei suutnud leida üht vektorit, mis kataks ülejäänud rikkeid. Seega sellist vektorit antud testvektorite hulgas ei leidu ning peab leiduma veel vähemalt üks vektor, mis ülejäänud rikked kataks. Kuna me sellise vektori leidsime ja sellega on kõik rikked avastatud, siis vähem vektoreid me minimeeritud testvektorite hulka valida ei saa.

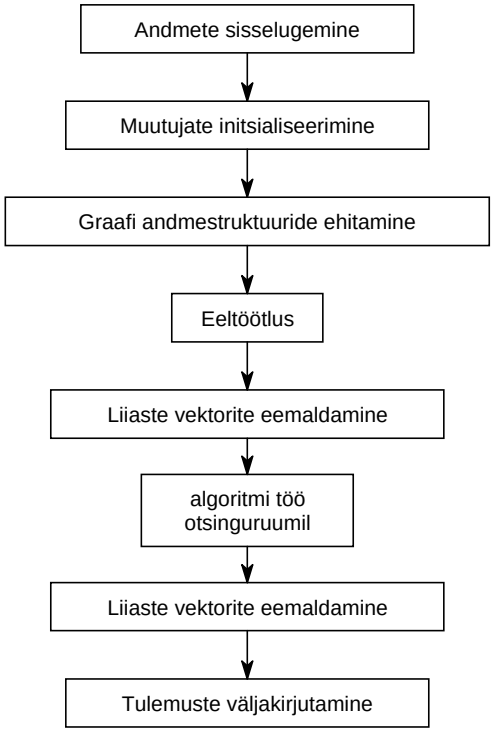
Kuid kui minimeeritud testvektorite hulk sisaldab rohkem kui kolme vektorit lisaks unikaalsetele, siis ei saa me enam lahenduse "globaalsust" tõestada, kuigi võib väita, et selle tõenäosus on päris suur. Seda seepärast, et 3 ja rohkema vektori puhul võib esineda mitteunikaalsete vektorite kombinatsioone, mis annavad minimaalsema tulemuse kui meie poolt saavutatu. Sellistel juhtudel tuleks rakendada keerulisemaid meetodeid otsinguruumi kuuluvate vektorite hulga minimeerimiseks. Eksperimendid näitavad, et paljud minimeerimisel leitud testvektorite hulgad on globaalsed miinimumid, kuna nad vastavad eeltoodud kriteeriumile.

Algoritmi realisatsioon

Et hinnata loodud algoritmi korrektsust ja tõhusust tuli antud algoritm ka realiseerida. Kuna Disaini ja Testi Keskuses on testimisalase problemaatikaga juba aastaid tegeldud, siis on olemas ka mitmete elementaarsete testigenereerimise ülesannete lahendamise juures tarvitatavate protseduuride C-keelsed realisatsioonid, mille abil on loodud rida vahendeid loogikalülituste tasemel testimisprobleemide lahendamiseks. Et aga viimastel aastatel on levinud objektorienteeritud tehnoloogia, mis pakub võimsamaid vahendeid mitmete algoritmide ja andmestruktuuride realiseerimiseks, siis porditi olemasolevad realisatsioonid C++ keelde, mille tulemusena loodi klasside teek ClassLib, mille väljatöötamisel ka töö autor osales. Et aga ClassLib on orienteeritud

tööle kõrgemal tasemele kui loogikalülituste tase ja lähteandmed olid esitatud veel orienteerituna loogikalülituste tasemele, siis tuli kasutada ka vanu, C-keelseid teeke. Et aga uues Turbo Testeri tarkvaras on eelistatud objektorienteeritud lähenemist ning ka C++ süntaks omab suuremat paindlikkust võrreldes C keelega, siis realiseeriti programm C++ keeles. Seetõttu tuli suurt tähelepanu pöörata ühilduvuse küsimustele C ja C++ vahel. Põhiliselt seisnesid probleemid C väljakutsekonventsioonidele vastavate funktsioonide kasutamisel C++ koodis erinevatel platvormidel. Samuti oli vaja tagada loodud programmi töö mitmel eri platvormil (antud juhul Windows ja SUN Solaris). Programmi väljatöötlus toimus peamiselt Windows keskkonnas, millele lisandusid regulaarsed kompilatsioonid ja kontrollkäivitused SUN Solaris keskkonnas, et tagada programmi porditavus. Väljatöötlusel kasutatud tarkvarast võiks ära märkida Microsoft® Visual C++ 5.0 (põhiline väljatöötluskeskkond) ja SUN® SparcWorks 4.2 C++ (kasutatud kompileerimisel Solarise keskkonnas). Ka kasutati programmi koodi kontrollimisel Compuvare® NuMega™ BoundsChecker 5.0 Visual C++ Edition'i beetaversiooni, et avastada võimalikud ressursilekked.

Järgnevalt on esitatud algoritmi realisatsiooni põhimõtteline skeem (joonis 7).



Joonis 7. Programmi põhimõtteline skeem

- Sisendandmed

  Fail digitaalskeemi kirjedlusega alternatiivsete graafide formaadis (vt. Lisa 3) ja fail genereeritud testvektoritega, millele on lisatud *veatabel* - maatriks, kus iga rida näitab antud vektori kvaliteeti (mis rikkeid antud vektor avastab).
- Väljund

  Fail minimeeritud testvektorite ja/või veatabeliga - määratav programmi võtmetega
- Võtmed

  -*fault_cover* <protsent> - anname ette, millise rikete avastamise protsendi jures töö lõpetatake. Kasutatakse juhul, kui absoluutset antud meetodiga saavutatavat miinimumi mingil põhjusel saavutada ei taheta.

  -*no_fault_table* - koos testvektoritega ei väljastata veatabelit

  -*extension* <laiend> - anname ette muu faililaiendi kui vaikimisi *.flt

Et graaf, mis tekib testvektorite maatriksist on reeglina küllalt hõre - tihtipeale on täidetud alla 50% maatriksist, siis ei esitata graafi mälus maatriksina, vaid kujul, kus iga tipu indeksi juures on massiiv, kus hoitakse nende tippude indekseid, millega antud tipp ühendatud on. Ehk kasutades C++ süntaksit:

```
class TTVector
{
  private:
    int* pArray;
    int  nCnt;
  public:
    TTVector(void);
    ~TTVector(void);
    int GetLen(void) const;
    int& operator[](const int index);
};
```

Graaf ongi kujutatud selliste vektorite massiivina. Et suurimate skeemide (c7552, vt. tabel 3-5) korral oli tipu võimalike naabrite arv üle 7000, siis andis selline andmete esitusviis olulist ajalist võitu vektori kvaliteedi kontrollimisel ja vektori poolt avastatavate rikete registreerimisel.

Eksperimentaalsed tulemused

Eksperimendid viidi läbi kolme tüüpi testigeneraatoritega: deterministliku, geneetilise ja juhuslikuga. Deterministlik testigeneraator kasutas PODEM algoritmi [10]. Teised generaatorid on võetud Turbo Testeri [6] paketist. Tabelis 2 on toodud tulemused

ISCAS85 etalonskeemide [9] testvektorite hulkade minimeerimisel. Nagu tulemustest näha, töötab esitatud algoritm täiesti rahuldava kiirusega, kui arvestada ülesannete dimensiooni. Eksperimendid viidi läbi 233 MHz Pentium II arvutil Windows 95 keskkonnas.

Skeem	PODEM Pakkimise aeg, s		Geneetiline Pakkimise aeg, s		Juhuslik Pakkimise aeg, s	
	graaf	maatriks	graaf	maatriks	graaf	Maatriks
c432	0.00	0.02	0.00	0.02	0.00	0.02
c499	0.00	0.02	0.00	0.02	0.00	0.02
c880	0.01	0.03	0.01	0.03	0.01	0.03
c1908	0.01	0.03	0.01	0.03	0.01	0.03
c2670	0.01	0.05	0.01	0.04	0.01	0.03
c3540	0.01	0.06	0.01	0.05	0.01	0.05
c5315	0.02	0.18	0.01	0.06	0.01	0.06
c6288	0.01	0.05	0.01	0.04	0.01	0.04
c7552	0.02	0.16	0.01	0.11	0.04	0.29

Tabel 2. Testvektorite hulkade minimeerimiseks kulunud aeg graafesitusel ja maatriksesitusel

Et paremini mõista tabeleid 3 - 5, peame esitama mõned täiendavad mõisted.
Testvektorit, mis ei avasta unikaalsete vektorite poolt avastatud riketele lisaks ühtegi uut riket, nimetame *liiaseks*. *Otsinguruumiks* nimetame nende testvektorite hulka, mis ei sisalda unikaalseid ja liiaseid vektoreid.
Otsinguruumi suurus määrab algoritmi poolt lahendatava ülesande dimensiooni, mis antud juhul määrab eksponentsiaalselt ülesande ajalise keerukuse.

Eksperimendid näitavad, et unikaalsete vektorite eraldamine eeltöötluse käigus vähendab otsinguruumi märgatavalt. Deterministlike testvektorite hulkade korral moodustasid unikaalsed vektorid 51-83% vektorite üldarvust ning otsinguruum moodustas keskmiselt ainult 38% esialgsest testvektorite hulgast. Geneetiliste ja juhuslike testvektorite hulkade korral oli unikaalsete vektorite protsent vastavalt 71-100 ja 57-100. Keskmine otsinguruum geneetilise testigeneraatori korral oli 21% ja juhusliku testigeneraatori korral 26% testvektorite üldarvust.

Skeem	Vektorite arv	Unikaalsed vektorid	Liiased vektorid	Otsinguruum	Minimeeri-mise tulemus
c432	89	52 (58%)	31	6	55
c499	140	94 (67 %)	31	15	100
c880	70	50 (71 %)	15	5	52*
c1908	144	120 (83 %)	20	4	122*
c2670	160	111 (69 %)	28	21	119
c3540	201	137 (68 %)	37	27	145
c5315	178	91 (51 %)	20	67	108
c6288	41	30 (73 %)	3	8	33
c7552	276	190 (69 %)	62	24	198

Tabel 3. Tulemused deterministliku testigeneraatori testvektoritel

Skeem	Vektorite arv	Unikaalsed vektorid	Liiased vektorid	Otsinguruum	Minimeeri-mise tulemus
c432	51	44 (86 %)	1	6	46*
c499	86	84 (98 %)	0	2	85*
c880	46	35 (76 %)	5	6	38
c1908	121	109 (90 %)	10	2	110*
c2670	112	80 (71 %)	9	23	87
c3540	155	133 (86 %)	11	11	138
c5315	115	93 (81 %)	9	13	99
c6288	21	21 (100 %)	0	0	21*
c7552	192**	149 (78 %)	27	16	156

Tabel 4. Tulemused geneetilise testigeneraatori testvektoritel

Skeem	Vektorite arv	Unikaalsed vektorid	Liiased vektorid	Otsinguruum	Minimeeri-mise tulemus
c432	51	45 (88 %)	3	3	46*
c499	86	86 (100 %)	0	0	86*
c880	63	43 (68 %)	15	5	45*
c1908	132	109 (83 %)	16	7	112
c2670	107**	72 (67 %)	27	8	75
c3540	167	137 (82 %)	16	14	143
c5315	132	100 (76 %)	19	13	106
c6288	24	21 (88 %)	1	2	22*
c7552	249**	143 (57 %)	41	65	164

Tabel 5. Tulemused juhusliku testigeneraatori testvektoritel

* - leiti globaalne miinimum

** - algne testvektorite hulk ei avastanud kõiki rikkeid

Toodud tabelitest 4 ja 5 on hästi näha geneetilise ja juhusliku testigenereerimise algoritmi käitumise sarnasus - testimise tulemusel saadud vektorite arv on mõlemal juhul peaaegu sama suur. Seevastu deterministliku testigeneraator väljastas oluliselt rohkem testvektoreid, kuid mis allusid minimeerimise protseduurile üpris hästi. See on seletatav sellega, et deterministlik testigeneraator ei rakenda mingit sisemist testvektorite hulga minimeerimist.

Juhuslik ja geneetiline testigeneraator kasutavad sisemist (dünaamilist) testvektorite minimeerimist. Sellega on seletatav unikaalsete vektorite kõrge protsent katsetulemustes.

Tulemusi võrreldi ka J. S. Takhar'i ja D. J. Gilberti töös [5] saavutatud tulemustega. Minimaalsed vektorite hulgad kattusid publikatsioonis [5] saavutatutega, kusjuures aega kulutati selleks suurusjärgu võrra vähem.

Tehti ka eksperiment, et loodud algoritmi jõudlust võrrelda lihtsa testvektorite vastupidises järjekorras simuleerimisega. Tulemused on toodud tabelis 6.

Skeem	Testvektoreid	Vastupidi simuleerides	Realiseeritud algoritm
c432	89	89	55
c499	140	137	100
c880	70	70	52
c1908	144	142	122
c2670	160	160	119
c3540	201	198	145
c5315	178	173	108
c6288	41	40	33
c7552	276	275	198

Tabel 6. Võrdlus testvektorite vastupidises järjekorras simuleerimisega

Tulemusi analüüsides võib öelda, et testvektorite vastupidises järjekorras simuleerimine ei anna väga häid tulemusi ja tuleks eelistada muid meetodeid töös kirjeldatud.

Kokkuvõte

1. Käesolevas töös on realiseeritud testvektorite hulga minimeerimise ahne algoritm, mille puhul on algoritmi sisenditele rakendatav kriteerium on valitud lähtudes monotoonsete süsteemide teooriast. Algoritmi saaks edaspidi parendada, tuues sisse näiteks valikud lõpliku hulga koostamisel, kus ei rahulduta algoritmi esialgse tulemusega, vaid üritatakse seda korduvate valikute abil kas siis veel rohkem minimeerida või siis tõestada, et on saavutatud globaalne miinimum. Eriti pakub see huvi nende testvektorite komplektide korral, kus me ei suutnud näidata, et tegemist on globaalse miinimumiga.
2. On läbi viidud katseseeriad rahvusvaheliselt tunnustatud ISCAS'85 etalonskeemidel [9]. Eksperimentaalsed tulemused näitasidd, et algoritm annab häid tulemusi ning töötab kiiresti.
3. Lisaks nimetatud tulemustele on töö autor osalenud diagnostikatarkvara Turbo Tester arendustöös järgmiste ülesannete lahendamisel:
   - Objektorienteeritud teegi ClassLib loomine, kus autori otseseks ülesandeks oli teegi disaini juhtimine ja osaline kodeerimine. Tehtud töö sai aluseks reale publikatsioonidele [17, 18, 20, 22, 23].
   - Automaatse testigenereerimissüsteemi väljatöötamine lihtsustatud RISC-protsessorite klassile, kus autori ülesandeks oli probleemi tarkvaraline realisatsioon.
   - Eksperimentide seeria läbiviimine väljatöötatud tarkvaraga lihtsustatud RISC-protsessoritel, mis näitasid funktsionaalsel tasemel genereeritud testide kehtivust ka madalamal tasemel ning mille tulemuseks oli rida publikatsioone [19, 21, 24].

Summary

1. Greedy test set minimimization algorithm has been implemented in this paper in which the selection criteria is based on the theory of monotonic systems. Algorithm could be further improved by introducing backtracking to the process.
2. Series of experiments have been carried out on internationally approved ISCAS'85 benchmark circuits. Experiments showed that algorithm gives good results and works fast.
3. In addition to these results, author of the paper has participated in the development of diagnostic software package Turbo Tester solving the following problems:

- development of object-oriented class library ClassLib where his direct responsibilities included heading the design and partly coding. This work formed basis for several publications [17, 18, 20, 22, 23].
- development of an ATPG for simplified RISC-processors, where author realized it in software
- carrying out series of experiments with aforementioned RISC-processors which showed that tests generated on the functional level discover also faults on lower levels of abstarction. This work was basis for several publications [19, 21, 24].

Kasutatud kirjandus

[1] J. A. Waicukauski et al., "ATPG for Ultra-Large Structured Designs". *Proc. of the International Test Conference*, pp. 44-51, 1990.

[2] M. H. Schulz, E. Auth, "Improved Deterministic Test Pattern Generation with Application to Redundancy Identification". *IEEE Trans. On CAD*, Vol. 8, No. 7, pp. 811-816, 1989.

[3] E. M. Rudnick, J. H. Patel, "Putting the Squeeze on Test Sequences". *Proc. of the International Test Conference*, pp. 723-732, 1997.

[4] I. Pomeranz, S. M. Reddy, "On the Compaction of Test Sets Produced by Genetic Optimizer". *Proc. of the Asian Test Symposium*, pp. 4-9, 1997.

[5] J. S. Takhar, D. J. Gilbert, "The Derivation of Minimal Test Sets for Combinational Logic Circuits using Genetic Algorithms". *Proc. of the 40th Midwest Symposium on Circuits and Systems*, Sacramento, USA, 1997.

[6] R. Ubar, J. Raik, P. Paomets, E. Ivask, G. Jervan, A. Markus, "Low-Cost CAD System for Teaching Digital Test". *Microelectronics Education*. World Scientific Publishing Co. Pte. Ltd. pp. 185-188, Grenoble, France, Feb. 1996.

[7] P. Sapiecha, "An Approximation Algorithm for Minimal Reduct Problem". *Proc. of the First Workshop on Rough Set Theory*, Poland, 1992.

[8] Z. Pawlak, "Rough Sets, Theoretical Aspects of Reasoning about Data", *Kluwer Academic Publishers*, 1991.

[9] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translation in Fortran". *ISCAS*, pp. 662-698, 1985.

[10] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits". *IEEE Trans. Comput*., vol. C-30, pp. 215-222, March 1981.

[11] R. Kuusik "The Super-Fast Algorithm od Hierarchical Clustering and the Theory of Monotonic Systems", *Transactions of Tallinn Technical University*, No. 734, pp. 37-62, 1993. http://www.datalaundering.com/download/extrem01.pdf

[12] L. Võhandu, R. Kuusik "Cliques and Algorithms With a Hidden Parallelism", *Transactions of Tallinn Technical University*, No. 734, pp. 63-75, 1993.

[13] J. A. Abraham, “Fault Modeling in VLSI”, *VLSI Testing*, T.W. Williams (Editor), North-Holland, pp. 1-27, 1986.

[14] V. Praust "Keerukusteooria alused", Tallinn, 1996.

[15] R. Ubar, "Test Generation for Digital Circuits Using Alternative Graphs", *Proc. of Tallinn Technical University, Estonia*, No. 409, pp. 75-81 (in Russian), 1976.

[16] R. Ubar, "Test Synthesis with Alternative Graphs", *IEEE Design and Test of Computers*, Vol.13, No. 1, pp. 48-57, Spring 1996.

Lisa 1

Töö autori publikatsioonid aastatest 1996-1998, mis on seotud antud töö temaatikaga

[17] R.Ubar, J.Raik, P.Paomets, E.Ivask, G.Jervan, A.Markus. "Low-Cost CAD System for Teaching Digital Test. Microelectronics Education". World Scientific Publishing Co. Pte. Ltd. pp. 185-188, Grenoble, France, Feb. 1996.

[18] G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. "Teaching Test and Design with Turbo Tester Software". Proc. of the 3rd Advanced Training Course: Mixed Design of Integrated Circuits and Systems MIXDES'96. pp. 589-594, Lodz, Poland, May 30 - June 1, 1996.

[19] R.Ubar, A.Markus, G.Jervan, J.Raik. "Fault Model and Test Synthesis for RISC Processors". Proc. of the 5-th Baltic Electronics Conference. pp. 229-232, Tallinn, Estonia, Oct. 1996.

[20] G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. "CAD Software for Digital Test and Diagnostics". Proc. of the Conference on Design and Diagnostics of Electronic Circuits and Systems '97. Ostrava, Czech Republic, May 12-14, 1997.

[21] M.Brik, G.Jervan, A.Markus, J.Raik, R.Ubar. "A Hierarchical Automatic Test Pattern Generator Based on Using Alternative Graphs". Proc. of the 4-th International Workshop on Computer Aided Design of Modern Devices and ICs. pp. 415-420, Poznan, Poland, June 12-14, 1997.

[22] G.Jervan, A.Markus, J.Raik, R.Ubar. "Automatic Test Generation System for VLSI". Proc. of the 1-st Electronic Circuits and Systems Conference. pp. 255-258, Bratislava, Slovakia, Sep. 4-5, 1997.

[23] G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. "A Set of Tools for Estimating Quality of Built-In Self-Test in Digital Circuits". Proc. of the International Symposium on Signals Circuits and Systems. pp. 362-365, Iasi, Romania, Oct. 2-3, 1997.

[24] G.Jervan, A.Markus, J.Raik, R.Ubar. "Assembling Low-Level Tests to High-Level Symbolic Test Frames" Proc. of the 15th NORCHIP Conference pp. 275-281, Tallinn, Estonia, Nov. 10-11, 1997.

[25] A. Markus, J. Raik, R. Ubar. "Test Set Minimization using Bipartite Graphs". *Proc. of the BEC'98 Conference*, Tallinn, Estonia, 1998. (*to be published*)

Autori muud publikatsioonid aastatest 1996-1998

[26] G. Jervan, A. Markus, R. Ubar, J. Raik. "Mixed Level Deterministic - Random Test Generation for Digital Systems". *Proc. of the MIXDES'98 Conf.*, Lodz, Poland, June 18-20, 1998 (*to be published*)

[27] G. Jervan, A. Markus, J. Raik, R. Ubar. "A Decision Diagram based Hierarchical Test Generator". *Proc. of the BEC'98 Conference*, Tallinn, Estonia, 1998 (*to be published*)

[28] M. Brik, G. Jervan, A. Markus, P. Paomets, J. Raik, R. Ubar. "Mixed-Level Test Generator for Digital Systems". *Proceedings of the Estonian Acad. Of Sci. Engng*., 1997, Vol. 3, No. 4, pp. 269-280.

Publikatsioonides [26-28] osales töö autor loogiliste tingimuste lahendamise (etteantud loogilisele avaldise liikmetele tuleb genereerida sellised väärtused, et avaldis oleks tõene) algoritmi väljatöötamisel ja realiseerimisel, mida kasutati funktsionaalse taseme testide genereerimisel.