Raimund Ubar

Digitaalsüsteemide diagnostika

II

Testide süntees ja analüüs

TTÜ
KIRJASTUS

Selle õppevahendi väljaandmist
Toetas Eesti Infotehnoloogia Sihtasutus
Tiigriülikooli projekti kaudu

Sisukord

Sissejuhatus

Käesolevas peatükis käsitleme testide genereerimise meetodeid loogikaelementide tasemel esitatud digitaalskeemidele. Vaadeldakse funktsionaalset ja struktuurset lähenemisviisi testide genereerimisele. Funktsionaalsetest meetoditest käsitletakse põhjalike ehk triviaalsete, pseudopõhjalike testide ja universaalsete testide sünteesi. Põhjalik test tähendab kõikide sisendvektorite kasutamist, seega on niisuguse testi keerukus eksponentsiaalne. Pseudopõhjaliku testi puhul jaotatakse skeem segmentideks või mooduliteks, millistest igaühele rakendatakse põhjalik test. Kuna segmentide sisendite arvud on palju väiksemad, kujuneb ka summaarne testi pikkus väiksemaks võrreldes põhjaliku testiga. Universaalne test põhineb loogikafunktsioonide monotoonsuse omaduste ära kasutamisel ja võimaldab samuti vähendada testi pikkust võrreldes triviaalse ehk põhjaliku testiga.

Struktuursed testide genereerimise meetodid põhinevad skeemi struktuuri esitamisel erinevates tehnikates nagu loogikaelementide võrguna, loogikafunktsioonide avaldistena või graafmudelitena ehk binaarsete otsustusdiagrammidena ning rikke mudeli kasutamisel. Vaadeldakse kõige klassikalisemat – ühe signaalitee aktiveerimise meetodit, mis seejärel üldistatakse kordsete signaaliteede aktiveerimise meetodiks. Käsitletakse rikete maskeerumist ja kordsete rikete testimist. Järgnevalt üldistatakse kirjeldatud meetodid kahele juhule: universaalse rikke mudeli kasutamisele, millega õnnestub loogiliselt käsitleda reaalseid füüsikalisi defekte, ning järjestikskeemide testimisele.

1. Funktsionaalsed meetodid

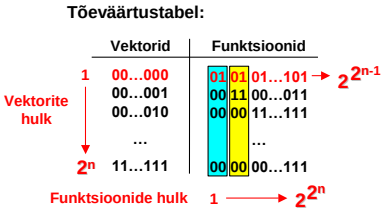
Digitaalskeemide funktsionaalsed testimise meetodid põhinevad skeemide funktsionaalsete kirjelduste kasutamisel näiteks Boole´i avaldiste abil. Funktsionaalse testi eesmärgiks on kindlaks teha, et skeemi käitumine vastab tema kirjeldusele. Kuna funktsionaalsete testide genereerimisel ei kasutata skeemide detailseid struktuurikirjeldusi, siis osutub mõnikord testide genereerimise keerukust oluliselt vähendada. Funktsionaalsed testid on võimelised avastama ka disainivigu, millega struktuursed rikete mudelite kasutamisel põhinevad testid hakkama ei saa.

1.1. Funktsionaalne ja struktuurne testimine

Vaatleme järgnevalt põhimõttelist erinevust digitaalskeemide funktsionaalse ja struktuurse testimise vahel. Erinevus seisneb testimise objektis.

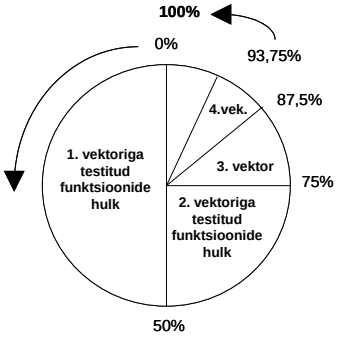
Funktsionaalse testimise puhul on testimise objektiks digitaalskeemi käitumine, mille võib esitada kõikvõimalike käitumisviiside, näiteks elementaarfunktsioonide hulgana. Funktsionaalne test on täielik, kui sellega tagatakse kõigi võimalike elementaarfunktsioonide kontroll. Näiteks, n sisendiga kombinatsioonskeemi elementaarfunktsioonide hulga võimsuseks on 2^n. Vastavalt sellele oleks vaja niisuguse skeemi testimiseks samuti 2^n testvektorit. Niisugust testi nimetatakse *triviaalseks* ehk *põhjalikuks* (ingl. k. *exhaustive*) testiks.

Funktsionaalse testimise korral võib testimise objektina käsitleda ka kõigi võimalike vigaste funktsioonide hulka. Näiteks, n sisendiga kombinatsioonskeemi puhul oleks niisugune vigaste funktsioonide arv 2^{2^n}.



Joonis 1-1. 2^{2^n} *erineva funktsiooni tõeväärtustabelist*

Alustades funktsionaalset testi on esimesed testvektorid vigade avastamise võime poolest väga võimsad. Joonisel 1-1 on kujutatud fragment 2^{2^n} erineva funktsiooni tõeväärtustabelist. $2^{2^n}-1$ funktsiooni sellest hulgast vastavad kõikvõimalikele vigastele käitumistele. Funktsionaalse testi eesmärgiks on kindlaks teha, millist funktsiooni sellest hulgast realiseerib antud skeem.



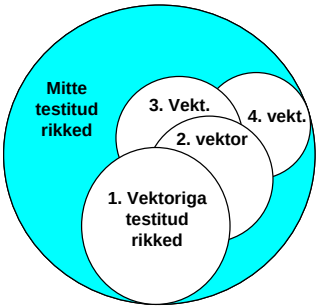
Joonis 1-2. *Funktsionaalse testi kvaliteedi sõltuvus testide arvust*

Rakendame nüüd testitava kombinatsioonskeemi sisendisse esimese testvektori, näiteks 00…000. Oletame, et skeemi väljundsignaaliks on 0 (või 1), mis vastab korras skeemi väljundsignaalile. Lihtne on aru saada, et kõik need veafunktsioonid, mis antud vektori 00…000 puhul peaksid andma

väljundväärtuseks 1 (või 0), on välistatud. Teiste sõnadega, pärast esimest positiivse tulemusega testvektorit, on pool kõigist võimalikest veafunktsioonidest välistatud. Võib ka öelda, et vigade avastamise protsent pärast seda esimest testvektorit on 50%. Järgmine positiivse tulemusega testvektor tõstaks selle protsendi 75%-ni, kolmas 87,5%-ni, neljas 93,75%-ni jne (vt. joon. 1-2).

Siit võib nüüd eksliku järelduse teha, et funktsionaalne test ongi väga võimas, viies üsna ruttu vigade avastamise protsendi 100%-ni. Tegelikult see pole nii, me jõuame küll kiiresti 95-100% piirile, aga sealt edasi hakkab protsent tõusma väga aeglaselt ja 100%-ni jõuame alles pärast kõigi 2^n testvektori rakendamist. Seega 100%-lise kvaliteediga funktsionaalse testi pikkuseks on 2^n. Suurte kombinatsioonskeemide puhul pole niisugune test reaalselt rakendatav.

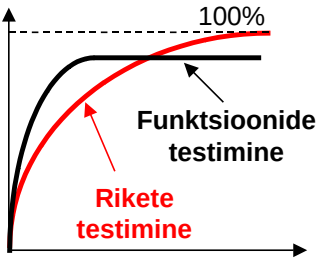
Struktuursete testide puhul on testimise objektiks skeemi struktuuriga määratud ette antud rikete hulk. Niisuguses rikete hulgas lepitakse tinglikult kokku, kuna kõikvõimalike rikete hulka pole praktiliselt võimalik loendada. Traditsiooniliselt on niisuguseks rikete hulgaks – konstantsete rikete hulk (konstandid 0 ja 1 iga loogikaelemendi sisendis ja väljundis).



Joonis 1-3. *Struktuurse testi kvaliteedi sõltuvus testide arvust*

Iga testvektor struktuurse testi puhul on võimeline avastama ette antud rikete hulgast mingit alamhulka, mis aga harilikult on väga väike ega küüni kunagi 50%-ni (joon. 1-3). Järgmised testvektorid avastavad täiendavalt uusi rikkeid, aga testimise kvaliteet ehk rikete kate ei kasva struktuurse testi puhul mitte kunagi nii kiiresti kui funktsionaalse testi poolt avastatavate vigaste funktsioonide kate.

Funktsionaalse ja struktuurse testi kvaliteedi võrdlus funktsioonina testi pikkusest on toodud joonisel 1- 4.



Joonis 1-4. *Funktsiooni tuletise geomeetriline interpretatsioon*

Kui testi alguses kasvab funktsionaalse testi kvaliteet kiiresti, siis mõne aja pärast hakkab see maha jääma struktuurse testi kvaliteedist. 100%-lise kvaliteedi saavutamiseks on struktuurne test väga palju lühem võrreldes funktsionaalse testiga. Samas on aga struktuurse testi genereerimiseks vaja palju vaeva näha.

Järgnevalt vaatleme kahte tüüpi funktsionaalseid teste: põhjalikke ehk triviaalseid ja pseudopõhjalikke teste.

1.2. Põhjalik ehk triviaalne test

Kombinatsioonskeemi *põhjalik* ehk *triviaalne* test seisneb kõigi võimalike sisendvektorite rakendamises skeemi sisenditele. Triviaalse testi headeks omadusteks on:

- neid ei ole vaja eelnevalt genereerida ja salvestada;
- nende kvaliteeti (rikete avastamise võimet) pole vaja analüüsida rikete simuleerimise teel;
- nende kasutamisel pole tarvis tegeleda rikete mudelitega;
- nende kasutamisel pole vaja tegeleda rikete liiasuse probleemiga;
- üksikute ja kordsete rikete kate on triviaalse testi puhul alati 100%;
- triviaalseid teste on kerge genereerida aparatuurselt reaalajas.

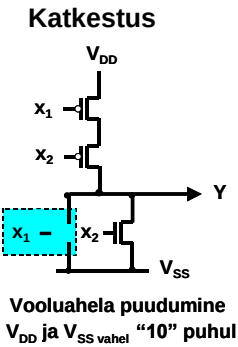
Siiski on triviaalsetel testidel kaks olulist puudust:

- eelpool mainitud head omadused kehtivad ainult niisuguste rikete jaoks, ei muuda kombinatsioonskeemi järjestikuliseks, ehk ei tekita järjestikskeemis uusi olekuid;
- triviaalsed testid on liiga pikad.

Kõige levinenumateks riketeks, mis muudavad kombinatsioonskeemi järjestikuliseks, on

- "*konstantsed katkestused*" CMOS transistorskeemides (ingl.k. *stuck-open, stuck-off*),
- lühisrikked, mis tekitavad skeemis tagasiside.

Konstantse avatud rikke näide on toodud joonisel 1-5.



x_1	x_2	y	y^d
0	0	1	1
0	1	0	0
1	0	0	Y'
1	1	0	0

Joonis 1-5. *Katkestusrike (stuck-open, stuck-off) transistorskeemis*

Skeemist on näha, et sisendvektori "10" $x_1 = 1, x_2 = 0$ korral puudub vool JA-EI elementi esitavas transistorskeemis nii ülemises kui ka alumises õlas. Skeemi väljundpotentsiaali määrab juhtme Y ja maa vahelisel parasiitmahtuvusel säilunud laeng, mis omakorda sõltub eelmise takti sisendvektori väärtusest. Parasiitmahtuvus toimib justkui mäluelement ja kombinatoorsest JA-EI elemendist funktsiooniga

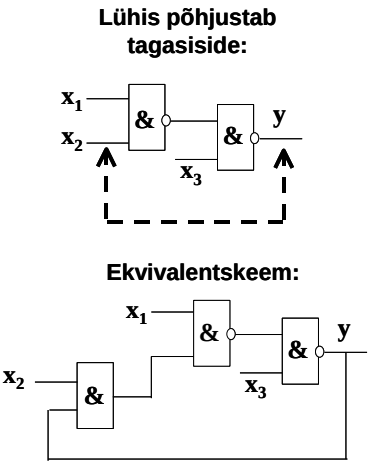
$$y = f(x_1, x_2) = \overline{x_1 x_2}$$

on saanud järjestikskeem funktsiooniga

$$y = f(x_1, x_2, y') = \overline{x_1 x_2} \vee x_1 \overline{x_2} y' .$$

Kui triviaalse testi definitsiooni järgi tuleks 2 sisendiga JA-EI elementi testida $2^2 = 4$ testvektoriga, siis katkestusrikke tõttu tekkinud 3 argumendiga funktsiooni triviaalse testi pikkuseks on $2^3 = 8$ testvektorit.

Lühisrikke näide, mis põhjustab kombinatsioonskeemi muundumise järjestikskeemiks, on toodud joonisel 1-6. Lühis kahe juhtme vahel joonisel 1-6 põhjustab tagasiside ahela väljundist y sisendisse x_2. JA-tüüpi lühisriket modelleeriv ekvivalentskeem on esitatud samuti joonisel 1-6.



Joonis 1-6. *Lühisrike, mis põhjustab tagasiside*

Tagasiside tõttu on esialgne kombinatsioonskeem funktsiooniga

$$y = f(x_1, x_2, x_3) = x_1 x_2 \vee \overline{x_3}$$

muundunud järjestikskeemiks funktsiooniga

$$y = f'(x_1, x_2, x_3, y') = x_1 x_2 y' \vee \overline{x_3},$$

kus y' on järjestikskeemi olekumuutuja. Kui triviaalse testi definitsiooni järgi tuleks 3 argumendiga esialgset funktsiooni testida $2^3 = 8$ testvektoriga, siis lühisrikke tõttu tekkinud 4 argumendiga funktsiooni triviaalse testi pikkuseks on $2^4 = 16$.

Üldjuhul on olukord veelgi hullem seetõttu, et niisugusi lühisrikkeid, mis võivad suurendada järjestikskeemi olekute arvu, võib olla suures

skeemis väga palju. Iga niisugune võimalik lühisrike kasvataks triviaalse testi pikkust kaks korda.

Milline oleks väljapääs olukorrast?

Üheks võimaluseks oleks kasutada struktuurset lähenemisviisi. Me võiksime eelpool käsitletud sekventsiaalseid rikkeid testida struktuurselt: defineerida kõigepealt võimalike rikete (defektide) hulk skeemi struktuuri suhtes ja seejärel genereerida iga niisuguse konkreetse rikke jaoks just seda riket avastav test.

Näiteks selleks, et testida joonisel 1-5 vaadeldud JA-EI elemendi katkestusriket, oleks vaja rakendada 2-vektorilist testjada: kõige pealt vektorit "00", et viia väljund olekusse $y = 1$, ja seejärel vektorit "10", et avastada defekt. Selleks aga, et testida lühist joonisel 1-6, on vaja jällegi 2-st vektorist koosnevat testjada: esiteks, vektor, mis produtseerib skeemi väljundis väärtuse $y = 0$ (näiteks, omistades $x_3 = 0$), seejärel vektor $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, et defekti mõju saaks avalduda skeemi väljundis.

Teiseks võimaluseks võidelda triviaalse testi pikkusega oleks kasutada pseudopõhjalikku testimist.

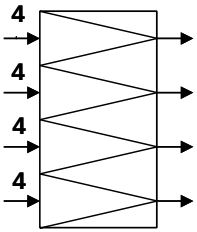
1.3. Pseudopõhjalik test

Pseudopõhjaliku testi puhul rakendatakse põhjalikke alamteste üksnes skeemi eri segmentidele. Skeemi segmendid võivad üldjuhul olla ka kattuvad. Segmenteerimise meetodist sõltuvalt võib eristada järgmisi pseudopõhjaliku testi tüüpe:

- verifitseeriv test;
- riistvaraline segmenteerimine;
- aktiveeritav segmenteerimine.

Verifitseeriv test

Mitme väljundiga kombinatsioonskeemides iga väljund sõltub enamasti mitte kõikidest sisenditest vaid ainult ühest sisendite alamhulgast. Iga väljundiga võib siis vastavusse panna skeemi seda osa (nn. *koonust*), mis toidab antud väljundit. Igale skeemi väljundile vastab mingi konkreetne funktsioon ja selle väljundi funktsionaalseks testimiseks võib kasutada põhjalikku ehk triviaalset testi.

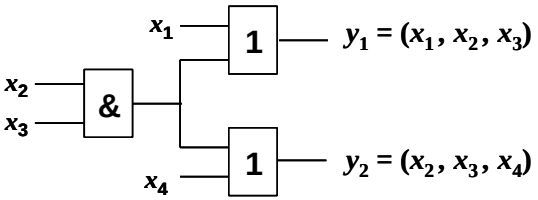


Joonis 1-7. *Segmenteeritud 4-väljundiline loogikaskeem*

Testimise meetodit, mille puhul rakendatakse paralleelselt tervele skeemile kõikide segmentide põhjalikud testid, nimetatakse *verifitseerivaks testimiseks*.

Joonisel 1-7 on esitatud 16-sisendiline kombinatsioonskeem, milles on 4 koonust, igal koonusel 4 sisendit. Igat koonuse testimiseks läheks vaja 2^4 = 16 testvektorit. Koonuseid järjestikku testides läheks 2^{16} = 65536 testvektori asemel vaja vaid 4*16 = 64 testvektorit, koonuseid paralleelselt testides läheks aga vaja vaid 16 testvektorit.

Üldjuhul on niisugused koonused aga ülekattuvad ja koonuste paralleelse testimise võimalus pole nii ilmne.



Joonis 1-8. *Segmenteeritud 2 väljundiga kombinatsioonskeem*

Vaatleme *n* sisendi ja *m* väljundiga kombinatsioonskeemi, mille koonuste kirjeldamiseks võib kasutada nn. *sõltuvusmaatriksit* (dependence matrix) [Jha 03]. Maatriksis on *m* rida, iga väljundi jaoks üks, ja *n* veergu, iga sisendi jaoks üks. Maatriksi element (i,j) on 1, siis ja ainult siis kui

väljund *i* sõltub sisendist *j*. Ülejäänud elementide väärtuseks on 0. Näiteks joonisel 1-8 esitatud skeemi puhul võtab sõltuvusmaatriks niisuguse kuju:

	x_1	x_2	x_3	x_4
y_1	1	1	1	0
y_2	0	1	1	1

Kui kaks sisendit, ei esine korraga ühe ja sama väljundi koonuses, siis võib ühte ja sama testsignaali rakendada korraga mõlemale sisendile. Sellisel juhul oleks võimalik testsignaalide arvu *p* viia väiksemaks sisendite arvust *n*. Meie näites oleksid niisugusteks sisenditeks x_1 ja x_4. Mõlemale saaks paralleelselt rakendada ühte ja sama testsignaali.

Formaalselt toimuks testsignaalide minimeerimine järgnevalt.

Kõigepealt jaotame sõltuvusmaatriksi veerud hulkadeks, kus igas niisuguse hulga reas on vähemalt üks element 1, nii et hulkade arv *p* oleks minimaalne. Sellist maatriksi nimetatakse *jaotatud sõltuvusmaatriksiks*. Üldjuhul niisugusi jaotusi võib olla palju. Meie näites saaksime jaotatud sõltuvusmaatriksi niisuguste hulkadega $\{x_1,x_4\}$, $\{x_2\}$, $\{x_3\}$:

	x_1	x_4	x_2	x_3
y_1	1	0	1	1
y_2	0	1	1	1

Sellest maatriksist saame omakorda konstrueerida *redutseeritud jaotatud* (ingl.k. *reduced partitioned*) sõltuvusmaatriksi. Selleks tuleks igas veergude hulgas veeruelemendid liita loogiliselt:

	x_1 x_4	x_2	x_3
y_1	1	1	1
y_2	1	1	1

Iga veeru jaoks taandatud jaotatud sõltuvusmaatriksis vajatakse ühte testsignaali. Pseudopõhjaliku testi ajal rakendatakse ühte ja sama testsignaali paralleelselt kõigile antud veerule vastavatele sisenditele.

Pseudopõhjalik test joonisel 1-8 esitatud skeemi jaoks koosneb 8-st testvektorist (põhjaliku testi korral oleks vaja läinud 16 vektorit):

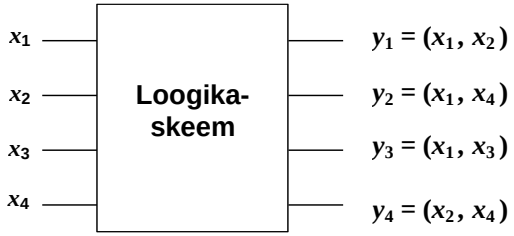
Tabel 1-1. *MPPP test skeemile joonisel 1-8*

x_1	x_2	x_3	x_4
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

Kirjeldatud meetodil saadud pseudopõhjalikku testi nimetatakse ka *redutseeritud verifitseerivaks testiks* (RVT) [Jha 03].

Olgu *w* maksimaalne arv elemente 1 ühes reas üle kõige ridade jaotatud või redutseeritud jaotatud maatriksis. Arv *w* tähendab maksimaalset sisendite arvu, millest üks skeemi väljund võib sõltuda.

Universaalne RVT *U*(*p*,*w*) on *p*-veeruline maatriks, milles suvaline *w*-veeruline alammaatriks sisaldab kõik võimalikud *w* bitist koosnevad reavektorid.



Joonis 1-9. *Segmenteeritud 4-väljundiline loogikaskeem*

Kui $p = w$, siis $U(p,w)$ sisaldab kõik 2^w *w* bitist koosnevat vektorit. Sellist pseudopõhjalikku testi nimetatakse *maksimaalse paralleelsusega pseudopõhjalikuks* (MPPP) testiks. Eelpool toodud näites oli p = w, mistõttu

tabelis 1-1 esitatud testi võibki nimetada MPPP testiks ja skeemi joonisel 1-8 võib nimetada MPPP testiga testitavaks skeemiks. 8-st testvektorist koosnev test tabelis 1-1 testib põhjalikult kummalegi skeemi väljundile y_1 ja y_2 vastavad koonused.

Olgu $p = w + 1$. Niisuguse testiga testitav skeem on esitatud joonisel 1-9. Redutseeritud jaotatud sõltuvusmaatriks antud skeemi jaoks koosneb 3-st veerust ($p = 3$):

	x_1	x_2	x_3 x_4
y_1	1	1	0
y_2	1	0	1
y_3	1	0	1
y_4	0	1	1

Universaalne RVT $U(p,w)$ juhtumi $p = w + 1$ jaoks on kõigi p-pikkusega paarsusvektorite (*even parity vectors*) või paarituse vektorite (*odd parity vectors*) hulk. Toodud näite jaoks joonisel 1-9 pseudopõhjalik paarsusel põhinev test on toodud tabelis 1-2. Selline test testib põhjalikult kõik joonisel 1-9 näidatud skeemi väljunditele vastavad koonused. Antud näite puhul 16-st vektorist koosnev põhjalik test on taandatud 4-st vektorist koosnevale pseudopõhjalikule testile.

Tabel 1-2. *Pseudopõhjalik test skeemile joonisel 1-9*

x_1	x_2	x_3 x_4
0	0	0
0	1	1
1	0	1
1	1	0

Vaadeldud verifitseeriva pseudopõhjaliku testi puuduseks on see, et koonuste piiridel esinevad defektid võivad avastamata jääda. Näiteks tabelis 1-1 esitatud pseudopõhjalik test joonisel 1-8 esitatud skeemi jaoks ei testi lühisriket sisendite x_1 ja x_4 vahel.

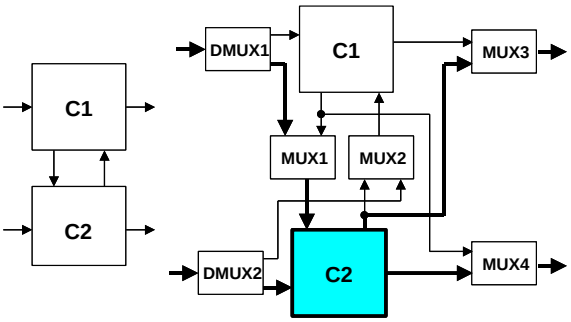
Riistvaraline segmenteerimine

Verifitseerivat testi pole võimalik kasutada, kui üks või mitu väljundit sõltuvad väga paljudest sisenditest. Et niisugusi teste pseudopõhjalikult testida, tuleks kasutada tehnikaid, mis võimaldaksid juhtida ning jälgida ka skeemi sisepunkte. Üheks võimaluseks on kasutada *riistvaralise segmenteerimise* meetodit - lisaaparatuuri testitavuse parandamiseks, näiteks multipleksereid.

Üldjuhul tuleks skeem segmenteerida $k \geq 2$ alamskeemiks S_i, milliseid testitakse põhjalikult. Need alamskeemid võivad kattuda. Igal segmendil S_i on $n(S_i) = n_i + p_i$ sisendit, kus n_i sisendit tulevad skeemi primaarsisenditest ja p_i sisendit tulevad skeemi nn. "lõikekohtadest", kus signaalide juhitavust on multiplekserite abil parandatud. Pseudojuhusliku test pikkuseks kujuneks niisugusel juhul

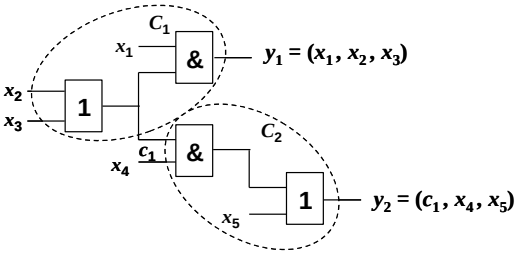
$$L = \sum_{i=1}^{k} 2^{n(S_i)} = \sum_{i=1}^{k} 2^{n_i + p_i}$$

Joonisel 1-10 on toodud näide skeemi segmenteerimise kohta [Jha 93]. Esialgne skeem on jaotatud kaheks segmendiks *C*1 ja *C*2. Signaalide juhtimiseks ja jälgimiseks segmentide vahelistel ühendustel kasutatakse multipleksereid: MUX1 ja MUX2, vastavalt segmentide C2 ja C1 juhtimiseks, ning MUX3 ja MUX4, vastavalt segmentide C2 ja C1 jälgimiseks. Demultiplekserid DMUX1 ja DMUX2 on selleks, et mitte kasutusele võtta täiendavaid signaalide juhtimiseks vaja minevaid sisendeid.



***Joonis 1-10**. Riistvaraline skeemi segmenteerimine*

Vaatleme joonisel 1-11 esitatud skeemi riistvaralise segmenteerimise võimalust. Vaatleme segmente *C*1 ja *C*2, milliste vahel on üksainus ühendus c_1. Selleks, et testida pseudopõhjalikult segmenti *C*1, tuleks rakendada tema sisenditele x_1, x_2 ja x_3 kõikvõimalikud signaalide kombinatsioonid ning teha juhe c_1 jälgitavaks. Selleks võiks kasutada multiplekserit enne väljundit y_2 valimaks signaalide c_1 ja y_2 vahel. Selleks, et testida pseudopõhjalikult segmenti *C*2, tuleks rakendada tema sisenditele c_1, x_4 ja x_5 kõikvõimalikud signaalide kombinatsioonid ning jälgida väljundit y_2 . Selleks võiks kasutada multiplekserit juhtmel c_1 valimaks signaalide c_1 ja ühe sisendi x_1, x_2 või x_3 vahel.



***Joonis 1-11.** Riistvaralise skeemi segmenteerimise näide*

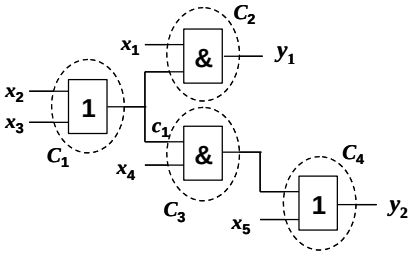
Riistvaralise segmenteerimise puuduseks on täiendavad aparatuursed kulutused. Sageli saab segmenteerimiseks vajalikke kulutusi vähendada, asendades riistvaralise segmenteerimise aktiveeritava segmenteerimisega.

Aktiveeritav segmenteerimine

Aktiveeritava segmenteerimise puhul iga segmendi testimiseks aktiveeritakse *signaaliteed* (ehk *signaalirajad*) primaarsisenditest segmendi sisenditeni ja segmendi väljunditest primaarväljunditeni. Paljudel juhtudel õnnestub niiviisi segmente põhjalikult testida.

Vaatleme veelkord skeemi joonisel 1-11, kuid seekord teist segmenteerimise strateegiat – segmenteerimist 4-ks segmendiks, nagu on illustreeritud joonisel 1-12.

Selleks, et testida pseudopõhjalikult segmenti $C1$, tuleks rakendada kõik 4 võimalikku 2-bitist vektorit sisenditele x_2 ja x_3 ning aktiveerida signaaliteed segmendi mõlemast väljundist skeemi primaarväljunditeni y_1 ja y_2. Esimesel juhul tuleks aktiveerida signaal $x_1 = 1$, teisel juhul signaalid $x_4 = 1$ ja $x_5 = 0$. Et testida pseudopõhjalikult segmenti $C2$, tuleks rakendada kõik 4 võimalikku 2-bitist vektorit vastava JA-elemendi sisenditele ning jälgida väljundit y_1. Et testida pseudopõhjalikult segmenti $C3$, tuleks rakendada kõik 4 võimalikku 2-bitist vektorit samuti vastava JA-elemendi sisenditele ning aktiveerida signaalitee segmendi väljundist skeemi primaarväljundini y_2 asetades $x_5 = 0$. Ning lõpuks, et testida pseudopõhjalikult segmenti $C4$, tuleks rakendada kõik 4 võimalikku 2-bitist vektorit vastava VÕI-elemendi sisenditele ning jälgida väljundit y_2.



Joonis 1-12. *Aktiveeritav skeemi segmenteerimine*

Kõik nimetatud aktiveerimistingimused on rahuldatud pseudopõhjaliku testiga, mis on esitatud tabelis 1-3. Testi pikkus on minimeeritud üksikute elementide testide ülekattumise abil. Iga konkreetse elemendi pseudopõhjalik test on märgistatud vastava segmendi nimega tähistatud veerus.

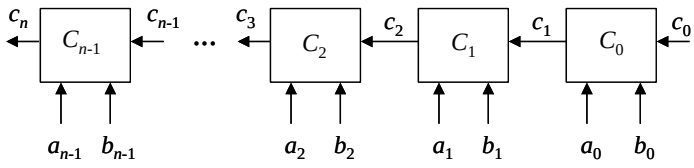
Tabel 1-3. *Pseudopõhjalik test skeemile joonisel 1-11*

x_1	x_2	x_3	x_4	x_5	C_1	C_2	C_3	C_4
1	0	0	1	0	*	*	*	*
1	0	1	1	0	*	*	*	*
1	1	0	1	0	*			
1	1	1	1	0	*			
0	0	1	0	0		*	*	
0	0	0	0	0		*	*	
0	0	0	0	1				*
0	0	1	1	1				*

Antud skeemi verifitseeriva põhjaliku testi pikkuseks oleks 32. Minimeerimise tulemusena on 16-vektori pikkune pseudopõhjalik test lühendatud 8 vektorini.

1.4. Iteratiivsete loogikamaatriksite testimine

Iteratiivne loogikamaatriks (*logic array*) koosneb omavahel ühendatud identsetest *loogikamoodulitest* (*cells*). Ühedimensionaalse loogikamaatriksi näiteks on summaator joonisel 1-13.



Joonis 1-13. *Summaator kui iteratiivne loogikamaatriks*

Iteratiivsete loogikamaatriksite riketemudelina kasutatakse nn. *ühe mooduli rikkemudelit* (ingl. k. *single cell fault model*) [Jha 03], kus eeldatakse, et suvalise ühe mooduli tõeväärtustabel võib suvaliselt modifitseeruda. Et avastadada selliseid rikkeid, tuleks testida loogikamaatriksite mooduleid põhjaliku testiga. Sellisel juhul võib loogikamaatriksite testi nimetada pseudopõhjalikuks testiks aktiveeritud segmenteerimise teel, kus segmentideks on moodulid.

Tabelis 1-4 on esitatud 3-järgulise *järjestiksummaatori* (ingl. k. *ripple-carry adder*) pseudopõhjalik test, kus kõik summaatori järgud (segmenteeritud moodulid) on testitud põhjalikult (kõigi võimalike sisendkombinatsioonidega).

Tabel 1-4. *Pseudopõhjalik test 3-järgulisele summaatorile*

	c_0	a_0	b_0	c_1	a_1	b_1	c_2	a_2	b_2	c_3	…
1	0	0	0	0	0	0	0	0	0	0	
2	0	0	1	0	0	1	0	0	1	0	
3	0	1	0	0	1	0	0	1	0	0	
4	0	1	1	1	0	0	0	1	1	1	
5	1	0	0	0	1	1	1	0	0	0	
6	1	0	1	1	0	1	1	0	1	1	
7	1	1	0	1	1	0	1	1	0	1	
8	1	1	1	1	1	1	1	1	1	1	

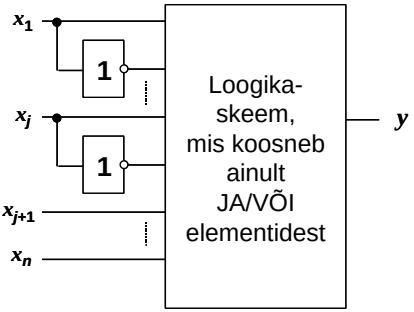
Tabelist 1-4 on näha, kuidas iga järgneva i+1 mooduli põhjalik test arendatakse välja iteratiivselt i-nda järgu mooduli põhjalikust testist. Mooduli $C0$ puhul on kõik sisendid c_0, a_0 ja b_0 vabad, seega põhjaliku testi saab välja kirjutada vahetult fikseerides kõik 8 võimalikku 3-bitist sisendvektorit. Saadud vektorite põhjal arvutame välja ülekande c_1 järgmisse järku - moodulisse $C1$. Arvestades nüüd veeruga c_1 ette antud kitsendusi, valime sisendite a_1 ja b_1 väärtused selliselt, et ka mooduli $C1$ sisendites kujuneksid kõik 8 võimalikku 3-bitist sisendvektorit. Arvutades nüüd välja veeru c_2 leiame, et see kujuneb samasuguseks kui veerg c_0, mistõttu mooduli $C2$ põhjalik test kujuneb täpselt samasuguseks nagu moodulil $C0$. Nüüd on kerge näha, et moodulite põhjalikud testid hakkavad korduma üle mooduli.

Toodud näitest on võimalik järeldada, et suvalist n-järgulist järjestik-summaatorit saab testida üksnes 8 testvektoriga. Võrreldes näiteks 32-

järgulise 64 sisendiga järjestiksummaatori 2^{64} vektorist koosneva põhjaliku testiga oleks see väga suur testimise aja kokkuhoid.

1.1.5. Universaalne test

Vaatleme testide genereerimist ühe väljundiga kombinatsioonskeemidele eeldusel, et on lubatud kõik rikked, mis võivad suvaliselt modifitseerida skeemi funktsiooni tõeväärtustabelit, kuid mitte muuta skeemi järjestikuliseks, näiteks lühise tõttu tekkiva tagasiside ahela kaudu [Jha 93]. Viime sisse veel ühe kitsenduse funktsiooni realiseerimise suhtes, nimelt on lubatud vaid sellised implementatsioonid, kus invertorid võivad olla ainult sisendites (joon. 1-14). Skeemis võib olla suvaline arv loogikaelementide tasemeid (elementide arv sisenditest kuni väljunditeni mingit signaaliteed pidi võib olla suvaline). Samuti on lubatud implementatsioonid, mida võib taandada *De Morgan'i teoreemi* kasutades skeemi kujule joonisel 1-14.



Joonis 1-14. *Loogikaskeem, millel on universaalne test*

Universaalse testi kontseptsiooni mõistmiseks formuleerime mõned definitsioonid.

<u>*Definitsioon 1-1.*</u> Ütleme, et vektor $X = (x_1, x_2, ..., x_n)$ *katab* ühte teist vektorit $Y = (y_1, y_2, ..., y_n)$, kui iga i jaoks nii pea kui $y_i = 1$, siis ka $x_i = 1$.

<u>*Definitsioon 1-2.*</u> Funktsioon on positiivselt (negatiivselt) *unaatne* (ingl. k. *positive or negative unate*) muutuja x_i suhtes, siis kui kõikide vektorite $(x_1,..., x_{i-1}, x_{i+1}, ..., x_n)$ jaoks vektor $(x_1,..., x_{i-1},0, x_{i+1}, ..., x_n)$ on kaetud vektoriga (katab vektorit) $(x_1,..., x_{i-1},1, x_{i+1}, ..., x_n)$. Kui funktsioon ei

ole ei positiivselt ega negatiivselt unaatne x_i suhtes, siis nimetatakse seda funktsiooni *binaatseks* selle muutuja x_i suhtes.

Definitsioon 1-3. Funktsioon on unaatne, kui ta on kas positiivselt või negatiivselt unaatne iga oma muutuja suhtes.

Näiteks, funktsioon $y = x_1 x_2 \vee \overline{x_3}$ on unaatne, sest ta on positiivselt unaatne muutujate x_1 ja x_2 suhtes ning negatiivselt unaatne muutuja x_3 suhtes. Samal ajal funktsioon $y = x_1 x_2 \vee \overline{x_2}\,\overline{x_3}$ on binaatne, sest ta ei ole unaatne muutuja x_2 suhtes.

Definitsioon 1-4. Sisendvektorit, mis annab funktsioonile väärtuse 1(0), nimetame *tõeseks vektoriks* (*vääraks vektoriks*). Tõest vektorit, mis ei kata ühtegi teist tõest vektorit, nimetatakse *minimaalseks tõeseks vektoriks.* Sarnaselt, väärat vektorit, mida ei kata ükski teine väär vektor, nimetatakse *maksimaalseks vääraks vektoriks*.

Definitsioon 1-5. Laiendatud tõeväärtustabeliks nimetatakse tõeväärtustabelit, millises on veerg iga *literaali* jaoks (tõeväärtustabelis on veerud vaid muutujate jaoks).

Laiendatud tõeväärtustabel funktsiooni $y = x_1 x_2 \vee \overline{x_2}\,\overline{x_3}$ jaoks on esitatud tabelina 1-5. Kolm võimalikku implementatsiooni antud funktsioonile on illustreeritud joonisel 1-15.

Definitsioon 1-6. Kõikide minimaalsete laiendatud tõeste vektorite ja maksimaalsete laiendatud väärate vektorite summat laiendatud tõeväärtustabelis nimetatakse *universaalseks testiks* tabeliga määratud kombinatsioonskeemile.

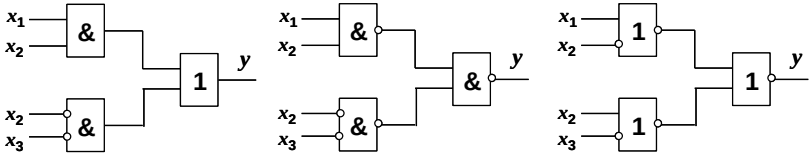
Universaalne testi näide funktsiooni $y = x_1 x_2 \vee \overline{x_2}\,\overline{x_3}$ jaoks, mis kehtib kõikidele skeemidele joonisel 1-15, on tabelis 1-5 ära märgitud tärnidega.

Universaalne test avastab JA/VÕI elementidest koosnevas skeemis kõik konstantsed rikked [Ake 73].

Niisuguse väite põhjendus tuleneb minimaalse tõese vektori ja maksimaalse väära vektori omadustest, mis seisnevad selles, et minimaalne tõene vektor avastab kõik need rikked, mida avastavad teda katvad tõesed vektorid, ning maksimaalne väär vektor avastab kõik need rikked, mida avastavad tema poolt kaetavad väärad vektorid laiendatud tõeväärtustabelis. Nimetatud omadused on seotud JA/VÕI skeemide monotoonsusega [Jha 03].

Tabel 1-5. *Pseudopõhjalik test 3-järgulisele summaatorile*

x_1	x_2	$\overline{x_2}$	$\overline{x_3}$	y	Minimaalsed tõesed vektorid	Maksimaalsed väärad vektorid
0	0	1	0	0		
0	0	1	1	1	*	
0	1	0	0	0		
0	1	0	1	0		*
1	0	1	0	0		*
1	0	1	1	1		
1	1	0	0	1	*	
1	1	0	1	1		



Joonis 1-15. *Implementatsioonid funktsioonile* $y = x_1 x_2 \vee \overline{x_2}\,\overline{x_3}$

<u>*Definitsioon 1-7.*</u> Funktsiooni f nimetatakse *monotoonseks*, kui funktsiooni iga argumentide vektori paari $\{X, Y\}$ puhul, kus X katab Y, ka $f(X)$ katab $f(Y)$.

Universaalse testi konstrueerimise meetodiks ongi minimaalsete tõeste ja maksimaalsete väärate vektorite välja selekteerimine laiendatud tõeväärtus tabelist.

Universaalse testi puuduseks on tema genereerimise suur töömahukus, kuna tõeväärtustabelite ridade arv kasvab eksponentsiaalselt funktsiooni muutujate arvuga.

Universaalse testi puuduseks on ka see, et kui funktsioon on binaatne kõigi muutujate suhtes, siis universaalne test taandub triviaalseks ehk

põhjalikuks testiks. Binaatse funktsiooni puhul ei leidu laiendatud tõeväärtustabelis ühtegi tõest vektorit, mis oleks kaetud mõne teise tõese vektoriga ega ühtegi väära vektorit, mis kataks mõnda teist väära vektorit. Seega ei õnnestu tõeväärtustabelist välja selekteerida ühtegi vektorit ja universaalse testi *n* sisendiga kombinatsioonskeemi jaoks moodustavad kõik 2^n sisendvektorit.

2. Struktuursed meetodid

Struktuursed testide genereerimise meetodid loogikaelementide tasandil esitatud digitaalskeemidele kasutavad rikke mudelit. Skeemi struktuuri kaudu defineeritakse kindel rikete klass ja testide genereerimine seisneb testvektori või –vektorite jada leidmises kõikide rikete jaoks antud klassist. Klassikaliseks rikete mudeliks loogikatasandil on konstantne rike. On veel muid levinenumaid rikke mudeleid: lühised, viiterikked. Universaalseks rikke mudeliks on funktsionaalse rikke mudel, mis võimaldab reaalseid füüsikalisi defekte kujutada loogikatasandile ja kasutada testide genereerimiseks loogikameetodeid.

Testide genereerimise efektiivsus sõltub oluliselt kasutatavast digitaalskeemi kirjeldusmeetodist. Loogikaelementide tase tekitab oma detailsusega probleeme testide genereerimise keerukust silmas pidades. Mõnevõrra leevendab neid probleeme struktuursete otsustusdiagrammide kasutamine, mis võimaldab tõusta loogikaelementide tasandilt kõrgemale makrotasandile vähendamaks mudeli keerukust. Parema lahenduse keerukuse probleemile aga pakuvad hierarhilised meetodid, milliseid käsitletakse järgmises peatükis.

2.1. Testi kvaliteet

Testide *kvaliteedi* mõõduks on rikete avastamise protsent etteantud rikete hulga suhtes [Bus 00]. Kõige levinenumaks rikete mudeliks on *konstantse rikke mudel*, (ingl.k. *stuck-at fault model*) kus eeldatakse, et iga ühendus loogikaelementide võrgus võib rikke tõttu olla fikseeritud kindlale väärtusele 0 või 1. Kui skeemis on n ühendust, siis võimalike rikete arv selles skeemis on 2^n. Ehkki on teada, et konstantse rikke mudel pole päris adekvaatne, eriti CMOS transistorskeemide jaoks [Cib 02, Max 03, Sac 98], on see siiski jäänud *de facto* standardiks loogikaskeemide testide kvaliteedi mõõtmisel. Põhjuseks on mudeli lihtsus, töödeldavus, loogiline käitumine, mõõdetavus ning adaptiivsus [Max 03].

Traditsiooniliselt eeldatakse, et skeemis võib korraga olla üksainus rike. See on muidugi suur kitsendus, aga keerukuse kasvu plahvatamise vältimiseks tuleb seda teha. Kui eeldada, et skeemis võivad olla suvalised kordsed rikked, siis nende arvuks oleks $3^n - 1$ (iga ühendus võib olla ühes

kolmest võimalikust olekust – “korras olek”, konstant 1 või konstant 0; skeem on “korras”, kui kõik ühendused on “korras”). Niisugune arv on liiga suur loendamiseks juba väikestegi skeemide korral.

Praktikas testitakse skeeme sageli disaineri poolt “välja mõeldud” piiratud arvu funktsionaalsete testvektoritega. Niisuguste käsitsi välja töötatud testvektorite kvaliteediks on sageli vaid 70-75%, mis kindlasti pole piisav skeemi kvaliteedi tagamiseks. Kvaliteedi mõõtmiseks kasutatakse *rikete simuleerimist*, mille eesmärgiks on kindlaks teha, milliseid rikkeid antud test avastab ja milliseid mitte. Funktsionaalsete testide kvaliteedi tõstmiseks on vaja selle testiga avastamata jäänud rikete jaoks genereerida täiendav test.

Osa skeemi rikkeid on funktsionaalses mõttes harilikult *liiased*. Rikke liiasus tähendab seda, et ta skeemis esinemise korral ei muuda skeemi loogikafunktsiooni. Seega ükski *loogiline* (loogikafunktsiooni testiv) test pole võimeline liiast riket avastama. Ühest küljest tekitab see probleemi testide genereerimisel – kui testi ei õnnestu genereerida, siis tuleb tõestada, et rike on liiane. Liiaste rikete tõestamine on aga väga töömahukas protseduur. Teisest küljest, liiaste rikete olemasolu tekitab teatavat segadust testide kvaliteedi mõõtmisel. Nimelt, liiaste rikete olemasolul pole võimalik isegi kõigi mitteliiaste rikete jaoks testvektorite leidmisel saavutada 100%-list rikete katet, mis aga annab antud testi kohta ebaadekvaatset informatsiooni.

Viimasena nimetatud põhjusel kasutatakse testide kvaliteedi mõõtmisel kahte mõõtu:

- *testi efektiivsus* (ingl.k. *test effectiveness*) – testiga avastatavate ja tõestatud liiaste rikete arvude summa suhe kõigi ette antud rikete arvu, ning
- *rikete kate* (ingl.k. *fault coverage*) – testiga avastatavate rikete arvu suhe kõigi ette antud rikete arvu.

Testi kvaliteedi mõõtmiseks kasutatakse veel järgmisi näitajaid:

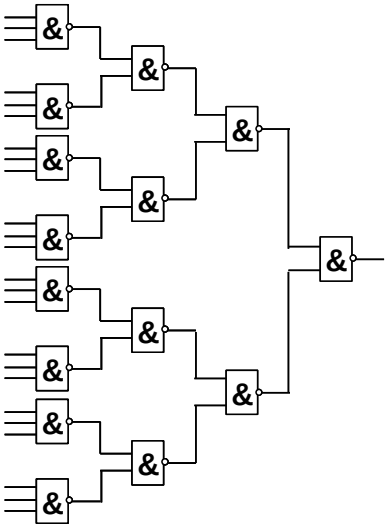
- *testide genereerimise aeg*,
- *testi pikkus* ehk maht (ingl.k. *test length*).

Testvektori genereerimise probleemi võib vaadelda kui etteantud riket avastava testvektori otsimist lõplikust ruumist. Otsinguruumi suurus on eksponentsiaalne skeemi sisendmuutujate suhtes ja probleem on NP-keeruline [Abr 95]. Teiste sõnadega, testide genereerimiseks ei eksisteeri polünomiaalse keerukusega algoritmi. Seetõttu on äärmiselt vajalik välja töötada efektiivseid meetodeid ja tehnikaid testide genereerimise protsessi

kiirendamiseks, et leida võimalikult väikese pikkusega teste. Nimetatud probleem on praegu antud valdkonnas äärmiselt aktuaalne.

Üheks testide genereerimise meetodiks on *juhuslike arvude meetod*. Selle meetodi puhul genereeritakse hulk juhuslikke sisendsignaalide kombinatsioone, määratakse igale sisendvektorile tema poolt avastatavate rikete hulk ja valitakse testvektoritena välja need juhuslikud sisendvektorid, mis avastavad kõige rohkem rikkeid.

Meetod on väga kiire, sest juhuslik sisendvektorite simuleerimine avastatavate rikete kindlaks tegemiseks ei võta eriti aega. Samas ei suuda see meetod garanteerida ei kõrget testi efektiivsust ega kõrget katet. Meetodi oluliseks puuduseks on, et pole võimalik tõestada rikete liiasust, mistõttu pole võimalik määrata ka testide efektiivsust. Madala rikete katte puhul pole aga võimalik teada, kas põhjus on testi ebaefektiivsuses või suure hulga liiaste rikete olemasolus. Standardseks rikete katte nõudeks mikroelektroonika tööstuses on 95%-99,9% [Cro 99].

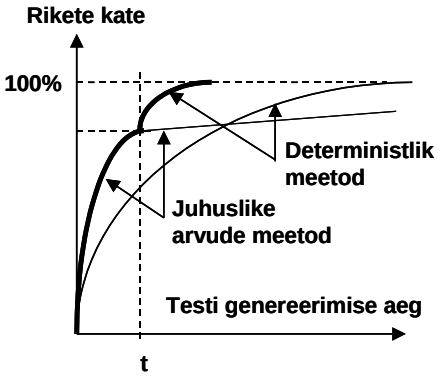


Joonis 1-16. *Kombinatsioonskeem raskesti avastatava rikkega*

Testi genereerimise protsessi võib kujutada tõusva rikete katte kõverana ajatelje suhtes. Algul tõuseb see kõver päris kiiresti, seejärel jääb

aga tõus üha aeglasemaks ja aeglasemaks. Põhjus seisneb nn. *raskesti avastatavate rikete* (ingl.k. *hard-to-test faults*) olemasolus. Enamiku rikete jaoks eksisteerib palju erinevaid sisendvektoreid, mis neid avastavad. Nende rikete avastamiseks pole raske juhuslike arvude meetodil testi leida. Osa rikete avastamiseks eksisteerib aga näiteks üksainus testvektor või väga väike arv erinevaid testvektoreid. Niisuguste vektorite leidmise tõenäosus juhuslike arvude meetodil on väga väike.

Raskesti testitavaks rikkeks on näiteks konstant 1 joonisel 1-16 kujutatud skeemil. Üksnes üksainus nullidest koosnev vektor kõigist võimalikest 2^{32} (4 miljardi) vektorite hulgast on võimeline avastama nimetatud riket.

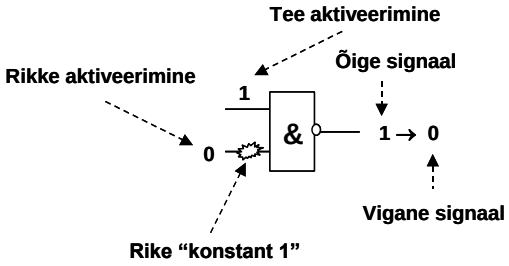


Joonis 1-17. *Testide genereerimine ajas*

Raskesti avastatavate rikete probleemi lahendamiseks tuleks kasutada teisi testide genereerimise meetodeid, näiteks, *deterministlikke*, kus testvektori otsing on suunatud konkreetsele rikkele. Joonis 1-17 illustreerib juhuslike arvude meetodi ja deterministliku testide genereerimise meetodi sünergiat. Deterministlik meetod töötab üksinda aeglaselt, sest põhineb kombinatoorsel otsingul, aga koos kiirel simuleerimisel põhineva juhuslike arvude meetodiga on võimalik saavutada suurt kiiruslikku effekti. Algul rakendatakse testvektorite genereerimiseks juhuslike arvude meetodit ja rikete katte kõver tõuseb suhteliselt kiiresti. Alates teatud ajahetkest t, kus on näha, et juhuslike arvude meetod kaotab oma efektiivsuse, rakendatakse deterministlikku meetodit ja 100%-line rikete kate saavutatakse kiiresti.

2.2. Ühe signaalitee testimise meetod

Kõige levinenumad automaatsed testide generaatorid põhinevad *signaaliteede aktiveerimise* meetodil. Selle meetodi puhul aktiveeritakse signaalitee testitava rikke asukohast läbi loogikaelementide kuni lähima skeemi väljundini. Rikke avaldumisel muutub tema asukohas (aktiveeritud tee algpunktis) loogikasignaali väärtus, mis levib edasi mööda aktiveeritud signaaliteed kuni väljundini, kus seda muutust seejärel nn. *veasignaalina* (ingl.k. *erroneous signal*) jälgida saab.



Joonis 1-18. *Signaalitee aktiveerimine läbi loogikaelemendi JA-EI*

Joonisel 1-18 on näidatud, kuidas signaaliteed aktiveeritakse läbi loogikaelemendi JA-EI. Tee aktiveerimine algab punktist, kus avaldub testitav rike. Rikke mõjule pääsmiseks tuleb ka *riket aktiveerida* (ingl. k. termineid: *fault activation, sensibilization, manifestation*). Selleks tuleb rikke asukoha juhtmele rakendada rikkele vastupidise väärtusega signaali. Antud juhul rikke "konstant 1" aktiveerimiseks rakendame signaali 0. Rikke aktiveerimiseks läbi elemendi JA-EI rakendame elemendi tesiele sisendile signaali 1. Tänu viimasele levib nüüd rikke poolt tekitatud veasignaal elemendi väljundisse.

Üldjuhul, kui signaalitee tuleb aktiveerida läbi keerukama funktsiooniga loogikakomponendi (mooduli), võib aktiveerimiseks vajalike komponendi sisendsignaalide väärtusi arvutada *Boole'i tuletiste* abil.

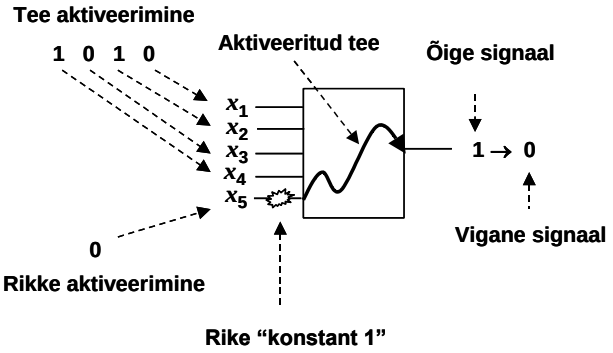
Joonisel 1-19 on näidatud signaalitee aktiveerimine sisendist x_5 läbi loogikakomponendi funktsiooniga

$$y = x_1 x_2 \vee \overline{x_3}(\overline{x_4} \vee x_5).$$

Rikke $x_5 \equiv 1$ aktiveerimiseks rakendame sisendisse x_5 väärtuse 0. Leides Boole'i osatuletise $\frac{\partial y}{\partial x_5}$ ja lahendades Boole'i differentsiaalvõrrandi

$$\frac{\partial y}{\partial x_5} = (\overline{x_1} \vee \overline{x_2})\overline{x_3}x_4 = 1,$$

leiame ühe võimaliku sisendvektori (x_1 x_2 x_3 x_4) = 0101, mis aktiveerib signaalitee sisendist x_5 läbi komponendi selle väljundisse *y*.

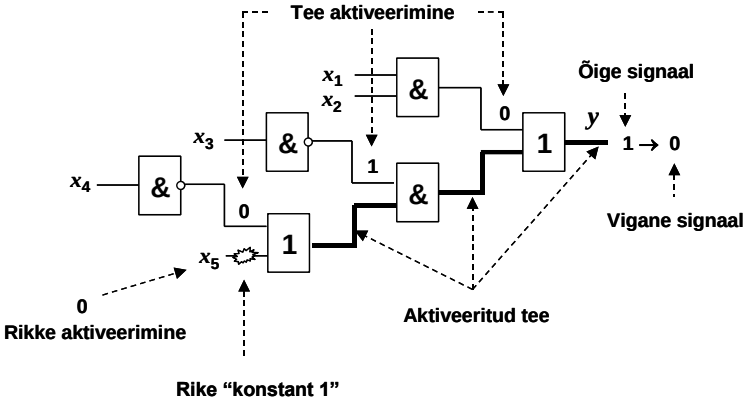


Joonis 1-19. *Signaalitee aktiveerimine läbi keerulise komponendi*

Signaalitee aktiveerimine läbi loogikaelementide (või keerulisemate komponentide) võrgu on iteratiivne protsess, kus aktiveerimine toimub läbi paljude elementide (või komponentide).

Joonisel 1-20 on illustreeritud näide signaalitee aktiveerimisest läbi loogikaelementide tasandil esitatud skeemi. Skeem realiseerib täpselt sama loogikafunktsiooni kui komponent joonisel 1-19. Boole'i tuletise arvutamise asemel komponendi jaoks tervikuna toimub nüüd tee aktiveerimine eraldi läbi üksikute loogikaelementide. Samas tekib uus probleem. Tee aktiveerimiseks vajalikud signaalid omistatakse skeemi sisepunktidele, neid aga ei saa testina sinna vahetult rakendada. Testi realiseerimine saab toimuda ainult skeemi sisendite kaudu.Teiste sõnadega, nüüd tuleb leida sisendsignaalide vektor, mis produtseerib tee aktiveerimiseks vajalikud signaalid skeemi sees. Üheks niisuguseks võimalikuks sisendvektoriks on

(x_1 x_2 x_3 x_4) = 0101, mis aktiveerib signaalitee sisendist x_5 läbi terve skeemi selle väljundisse *y*. Sama vektori leidsime eelmise näite korral tee aktiveerimiseks läbi komponendi joonisel 1-19.



Joonis 1-20. *Signaalitee aktiveerimine läbi loogikaskeemi*

Niisiis, testide genereerimise protsessi etteantud rikkele ühe signaalitee aktiveerimise meetodil võib jagada kolmeks etapiks:

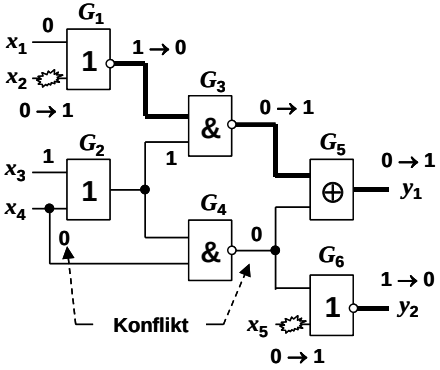
1) rikke aktiveerimine;
2) *rikke levitamine* (ingl.k. *fault propagation*) läbi skeemi mööda aktiveeritud teed;
3) *signaalide tagamine* (ingl.k. *line justification*), mille puhul rikke aktiveerimisel ja levitamisel fikseeritud skeemisisesed signaalid tagatakse sobiva sisendvektori valikuga.

Signaalide tagamise etapp kujutab endast üldjuhul kombinatoorset otsinguprotsessi, kus pidevalt tehakse valikuid ja otsustusi, ning kus konfliktide tekkimisel pöördutakse tagasi viimase otsustuse juurde selle muutmiseks (kui kõik valikuvõimalused on ammendatud jätkatakse tagasipöördumisi kuni esimese võimaluseni, kus õnnestub tehtud otsustust muuta).

Konfliktsituatsiooni tekkimist illustreerib joonis 1-21, kus üritatakse genereerida testi korraga kahele rikkele $x_2 \equiv 1$ ja $x_5 \equiv 1$. Riket $x_2 \equiv 1$ levitatakse aktiveeritud teed kaudu läbi elementide G_1, G_3 ja G_5 väljundisse

y_1. Aktiveerimiseks kasutatakse signaale $G_2 = 1$ ja $G_4 = 0$. Kuna skeemil on kaks väljundit, siis õnnestub paralleelselt teise väljundi kaudu testida ka riket $x_5 \equiv 1$. Selle rikke levitamiseks läbi elemendi G_6 väljundisse y_2 sobib täpselt äsja fikseeritud signaal $G_4 = 0$.

Testi genereerimise kolmandal etapil tuleb tagada valitud signaalid $G_2 = 1$ ja $G_4 = 0$ sobiva sisendvektoriga. Signaali $G_2 = 1$ tagamiseks valime vektori (x_3, x_4) = 10. Järgnevalt tuleks tagada signaal $G_4 = 0$. Selleks sobiks ainsa võimalusena vektor (G_2, x_4) = 11, mis on aga vastuolus juba fikseeritud väärtusega $x_4 = 0$. Pöördudes nüüd tagasi tingimuse $G_2 = 1$ tagamise juurde, leiame, et selleks oleks veel üks teine võimalus (x_3, x_4) = 01. Vajalik testvektor on seega (x_1, x_2, x_3, x_4 ,x_5) = 00010.



Joonis 1-21. *Konfliktsituatsioon signaaliteede aktiveerimisel*

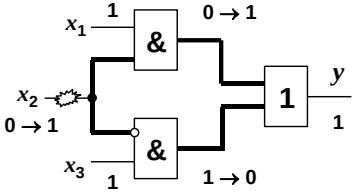
Juhul kui poleks võimalust olnud vektorit (x_3, x_4) = 10 muuta, oleks tulnud veel üks samm tagasi pöörduda, nimelt signaalitee aktiveerimise juurde läbi elemendi G_5. Osutub, et selleks sobib $G_4 = 0$ kõrval ka $G_4 = 1$. Sellisel juhul aga tuleb loobuda kahe rikke üheaegsest testimisest, kuna $G_4 = 1$ puhul pole enam aktiveeritud tee rikke $x_5 \equiv 1$ testimiseks.

Ühe signaalitee testimise meetodi põhiidee seisneb selles, et riket levitatakse väljundisse ühte ainsat signaaliteed pidi. Kui rike võiks hargnevuspunktides levida mitut teed pidi, siis ülejäänud teed blokeeritakse. Nimetatud võte välistab selle, et veasignaal, mis levib mitut erinevat teed pidi, võib nende teede koondumispunktis põhjustada iseenda maskeerumist.

Niisuguse maskeerumise võimaluse kontrolli sisse viimine teeks aga meetodi juba keerulisemaks ja aeglasemaks.

Joonisel 1-22 püütakse testi genereerida rikkele $x_2 \equiv 1$. Signaalide $x_1 = 1$ ja $x_3 = 1$ levitatakse riket edasi mööda kahte signaaliteed. Selle tulemusena aga rike maskeerub väljundelemendi sisendites ega levi enam edasi väljundisse. Rikke testimiseks tuleks blokeerida üks kahest teest, vastavalt kas signaali $x_2 = 0$ või signaali $x_3 = 0$ abil.

Ühe signaalitee testimise meetod on lihtne, aga meetodi puhul pole tegemist testide genereerimise algoritmiga. Nimelt võib juhtuda, et mingile konkreetsele rikkele ei õnnestugi leida testi, mis aktiveeriks riket üheainsa signaalitee kaudu, samas aga võib leiduda test, kus rike levib mitut teed kaudu. Seega ei ole ühe signaalitee testimise meetod algoritm – ta võib mitte leida lahendust, kui lahendus on siiski olemas.

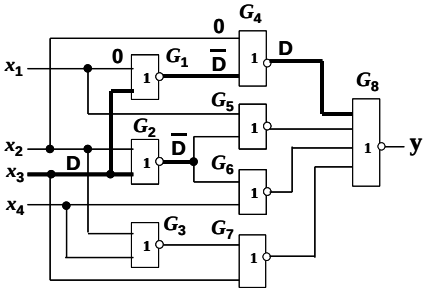


Joonis 1-22. *Rikke maskeerumise näide*

Paljud testide genereerimise meetodid põhinevad siiski signaaliteede aktiveerimise ideel ja kolmel eelpool nimetatud etapil – rikke aktiveerimisel, rikkesignaali levitanisel ja fikseeritud signaalide tagamisel. Meetodite efektiivsus sõltub erinevatest heuristikatest ja modelleerimise tehnikatest.

2.3. Kordsete signaaliteede aktiveerimise meetod

Testide genereerimise formaliseerimiseks signaaliteede aktiveerimise meetodil, mis lubaks rikete levi suvalise arvu teede kaudu, on välja töötatud 5-väärtuseline loogikamudel väärtustega 0, 1, *x*, *D* ja $\overline{D}$, kus x - tähendab määramatust ja *D* ning $\overline{D}$ - tähendavad rikkesignaale, vastavalt, konstanti 0 ja konstanti 1. *D* = 1, kui skeem on korras, ja *D* = 0, kui skeemis on rike.



Joonis 1-23. *Testimine üheainsa signaalitee kaudu pole võimalik*

Joonisel 1-23 on toodud näide, kus rikke $x_3 \equiv 0$ testimine üheainsa signaalitee kaudu pole võimalik. Tee aktiveerimine läbi elementide G_1, G_4 ja G_8 väljundisse *y* nõuab järgmiste signaalide fikseerimist: $x_1 = 0$, $x_2 = 0$, $G_5 = 0$, $G_6 = 0$ ja $G_7 = 0$. Aga juba üksnes signaali $G_5 = 0$ pole võimalik tagada, sest $x_1 = 0$ ja $G_2 = \overline{D}$ määravad üheselt, et $G_5 = D \neq 0$, mis tähendab konflikti. Samas on aga võimalik seda riket testida mitme tee kaudu. Otsitavaks testiks on (x_1, x_2, x_3, x_4) = 0010.

Niisugust testi võimaldab genereerida *D-algoritm*, mis põhineb nn. *D*-arvutusel [Rot 66]. Algoritmi üksikasjadesse tungimata (*D*-algoritmi kirjeldust võib leida pea igas digitaalskeemide testimisele pühendatud õpikus [Mic 03], [Bus 00], [Mou 00], [Lal 97], [Abr 95] jt.) kirjeldame pealiskaudselt algoritmi aluseks olevat põhimõtet.

Sarnaselt tõeväärsustabelitega kirjeldatakse igat loogikaelementi nn. *D-kuupide* tabeli abil. *D*-kuup kirjeldab tingimusi *D*-väärtuse levitamiseks läbi vastava elemendi. Tabelis 1-6 on toodud loogikaelementide JA, VÕI, JA-EI ning VÕI-EI *D*-kuupide tabelid.

Tabel 1-6. *D-kuupide tabelid loogikaelementidele*

JA				VÕI				JA-EI				VÕI-EI			
x_1	x_2	x_3	y	x_1	x_2	x_3	y	x_1	x_2	x_3	y	x_1	x_2	x_3	y
D	1	1	D	D	0	0	D	D	1	1	$\overline{D}$	D	0	0	$\overline{D}$
1	D	1	D	0	D	0	D	1	D	1	$\overline{D}$	0	D	0	$\overline{D}$
1	1	D	D	0	0	D	D	1	1	D	$\overline{D}$	0	0	D	$\overline{D}$
D	D	D	D	D	D	D	D	D	D	D	$\overline{D}$	D	D	D	$\overline{D}$

Tabel 1-7. *Singulaarsed katted loogikaelementidele*

JA				VÕI				JA-EI				VÕI-EI			
x_1	x_2	x_3	y	x_1	x_2	x_3	y	x_1	x_2	x_3	y	x_1	x_2	x_3	y
1	x	x	1	0	x	x	0	1	x	x	0	0	x	x	1
x	1	x	1	x	0	x	0	x	1	x	0	x	0	x	1
x	x	1	1	x	x	0	0	x	x	1	0	x	x	0	1
1	1	1	1	0	0	0	0	1	1	1	0	0	0	0	1

Tabelis 1-7 on esitatud kompaktsed tõeväärtustabelid samadele loogikaelementidele nn. *singulaarsete katete* (ingl.k. *singular cover*) ehk *singulaarsete kuupide* (ingl.k. *singular cube*) tabelitena.

D-algoritmi aluseks olev D-arvutus põhineb nn *D-ühisosa* (ingl.k. *D-intersection*) operatsioonil, mille abil toimub signaaliteede aktiveerimine. Vaatleme D-ühisosa operatsiooni lihtsa näite peal joonisel 1-24.

Genereerimie testi *D*-algoritmi abil rikkele "konstant 0" elemendi G_1 sisendis 1. Rikke aktiveerimine toimub tingimusega (*D*-kuubiga A), mis leitakse D-kuupide tabelist (tabel 1-6):

	1	2	3	6
A:	*D*	1	1	*D*

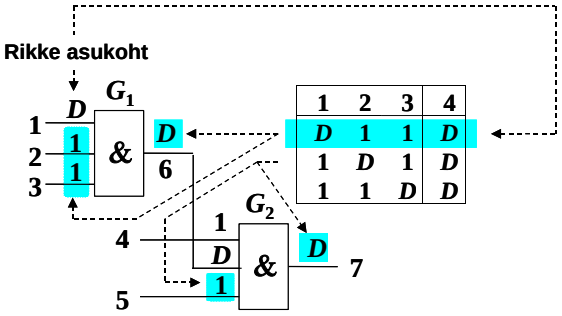
Et levitada signaali *D* läbi elemendi G_2, peame sobitama elemendi G_1 *D*-kuubi *A* (leidma ühisosa) mingi elemendi G_2 *D*-kuubiga. Selleks sobib *D*-kuup *B*:

	4	5	6	7
B:	1	1	*D*	*D*

Operatsiooni tulemusena saime kogu skeemi *D*-kuubi *C*:

	1	2	3	4	5	6	7
C:	*D*	1	1	1	1	*D*	*D*

D-kuup C olekski skeemi test, sisenditele 1,2,3,4,5 tuleks rakendada vektor *D*1111, kusjuures *D* väärtuseks on *D* = 1. Skeemi väljundsignaaliks oleks selle testi ajal korras skeemi puhul *D* = 1, rikke olemasolu korral aga *D* = 0.



Joonis 1-24. *Signaalitee aktiveerimine D-algoritmi abil*

D-ühisosa operatsiooni põhireeglid oleksid järgmised. Olgu meil kaks D-kuupi

$$A = (a_1, a_2, \ldots, a_n)$$
$$B = (b_1, b_2, \ldots, b_n)$$

kus $a_i, b_i \in \{0,1,x,D,\overline{D}\}$. Siis kehtiksid vektorite *A* ja *B* ühisosa leidmisel järgmised suhted:

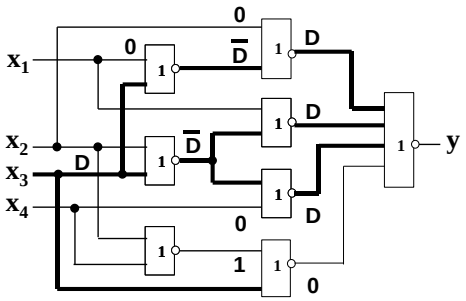
1) $x \cap a_i = a_i$;

2) kui $a_i \neq x$ ja $b_i \neq x$ siis

$a_i \cap b_i = a_i$, kui $b_i = a_i$, või

$a_i \cap b_i = \varnothing$ muudel juhtudel

3) $A \cap B = \varnothing$ kui vähemalt ühe i jaoks $a_i \cap b_i = \varnothing$.



Joonis 1-25. *Testimine üheaegselt mitme signaalitee kaudu*

Kasutades *D*-algoritmi oleks nüüd lihtne veenduda, et skeemis joonisel 1-24, kus testi genereerimine rikkele $x_3 \equiv 0$ üheainsa signaalitee kaudu pole võimalik, võib seda riket levitada siiski kolme tee kaudu (näidatud skeemil jämedate joontega). Viimase elemendi sisendisse tekib vektor *DDD*0, mis annab väljundväärtuseks $y = \overline{D}$. Seega, sisend vektori $(x_1, x_2, x_3, x_4) = 00D0$ puhul, kus $D = 1$, oleks korras skeemi ajal väljundis signaal 0. Rikke korral oleks väljundis 1.

Aegade jooksul on klassikalist *D*-algoritmi palju parandatud ja mitmesuguste heuristikatega efektiivsemaks tehtud. Tabel 1-8 näitab automaatsete testide generaatorite ajalugu selles osas, kuidas nende töökiirust on suudetud parandada [Bus 00].

Tabel 1-8. *Automaatsete testide generaatorite ajalugu*

Nr	Algoritm/meetod	Kiiruslik hinnang	Publitseerimise aasta
1	*D*-algoritm [Rot 66]	1	1966
2	PODEM [Goe 81]	7	1981
3	FAN [Fuj 83]	23	1983
4	TOPS [Kje 87]	292	1987
5	SOCRATES [Soc 88]	1574	1988

6	Waicukauski [Wai 90]	2189	1990
7	EST [Gir 91]	8765	1991
9	Recursive learning [Kun 92]	485	1992
8	TRAN [Cha 91]	3005	1993
10	Tafertshofer [Taf 97]	25057	1997

Kiiruslikud hinnangud, mis on tabelis toodud on tegelikult väga ligikaudsed ja neid tuleb käsitleda vaid kui suurusjärke, mis on seotud raskustega normaliseerida arvutite töökiirusi. Kiiruste võrdlemisel on baasiks võetud *D*-algoritmi implementatsioon.

2.4. Testide süntees disjunktiivnormaalkuju abil

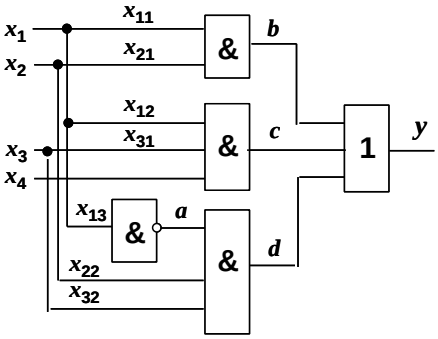
Testide genereerimise efektiivsuse tõstmiseks on üheks oluliseks teeks mudeli keerukuse vähendamine. Selleks on kaks võimalust: tõusta skeemide kirjeldamisel (modelleerimisel) *loogikaelementide mudelite* tasandilt kõrgematele tasanditele ja vähendada vaadeldavate rikete arvu.

Rikete arvu vähendamisel peab loomulikult garanteerima, et testide kvaliteet ei lange. Rikete arvu vähendamisest rääkides peetakse silmas seda *objektrikete* hulka (ingl.k. *target faults*), mida tuleks kasutatada testide generaatori töö juhtimiseks. Testide kvaliteeti saab ikkagi kontrollida ainult loogikaelementide tasandi kõigi rikete simuleerimise teel.

Kõrgema tasandi elementide mudelid võimaldaksid kiiremini läbi viia signaaliteede aktiveerimist. Käesolevas peatükis käsitleme loogikataseme *makromudeleid*, edaspidi aga veelgi kõrgemaid *register-edastustaseme* (*register-transfer level*) mudeleid.

Üheks võimaluseks on esitada loogikaskeem või selle alamskeem *ekvivalentse disjunktivnormaalkuju* (ingl.k. *equivalent disjunctive normal form*) abil. Joonisel 1-26 on esitatud loogikaelementide taseme skeem, mida saab esitada ekvivalentse disjunktiivnormaalkujuna (EDNK):

$$y = x_{11}x_{21} \vee x_{12}x_{31}x_4 \vee \overline{x_{13}}x_{22}x_{32}$$



Joonis 1-26. *Digitaalskeem loogikaelementide tasandil*

Disjunktiivset normaalkuju nimetatakse ekvivalentseks, kuna tema struktuur järgib vastava digitaalskeemi struktuuri. Skeemis on 8 signaaliteed ja ka EDNK-s on 8 muutujat (õigemini *literaali*), igale EDNK literaalile vastab üks ja ainult üks signaalitee digitaalskeemis. Konstantsete rikete mudeli käsitlusest nägime [Uba 05], et kombinatsioonskeemide testimiseks piisab testide genererimisest skeemi sisenditele ja hargnemispunktide harudele. Sellest järgneb, et antud skeemi testimiseks piisab testide genereerimisest skeemi kõikidele nendele muutujatele, mis on esitatud EDNK-s. Sisemuutujatele a,b,c,d ei ole vaja teste genererida. EDNK kasutamise effekt seisnebki selles, et neid "kasutuid" muutujaid selles mudelis polegi, mistõttu on mudeli keerukus väiksem kui esialgse loogikaskeemi keerukus joonisel 1-26.

Tabel 1-9. *Testide genereerimine EDNK abil*

x_{11}	x_{21}	∨	x_{12}	x_{31}	x_4	∨	$\overline{x_{13}}$	x_{22}	x_{32}		y		x_1	x_2	x_3	x_4
0	1		0	0	1		1	1	0		0		0	1	0	1
1	0		1	0	1		0	0	0		0		1	0	0	1
0	0		0	1	1		1	0	1		0		0	0	1	1
1	0		1	1	0		0	0	1		0		1	0	1	0
1	1		1	1			0	1	1		1		Test puudub			
1	1		1	0			0	1	0		1		1	1	0	
1	0		1	1	1		0	0	1		1		1	0	1	1
0	1		0	1			1	1	1		1		0	1	1	

Testide genererimise protsess EDNK abil on illustreeritud tabelis 1-9. Kõigepealt genereerime testid riketele konstant 1 ja seejärel riketele

konstant 0. Esimese testi eesmärgiks on konstrueerida test rikkele $x_{11} \equiv 1$. Rikke aktiveerimiseks fikseerime $x_{11} = x_1 = 0$. Rikke levitamiseks läbi JA-elemendi fikseerime $x_{21} = x_2 = 1$. Täiendame tabeli esimest rida väärtustega $x_{12} = 0$ ja $\overline{x_{13}} = 1$, ning $x_{22} = 1$. Rikke levitamiseks väljundisse läbi VÕI-elemendi peaksid olema $c = d = 0$, mis tähendab, et tuleks garanteerida disjunktsiooni ülejäänud kahe termi väärtuseks 0. Selleks piisab iga termi puhul suvalise vähemalt ühe literaali fikseerimisest väärtusega 0. Kolmandas termis on veel vaid üks väärtustamata literaal, nii fikseerimegi $x_{32} = x_3 = 0$. Täiendame tabelit veel $x_{31} = 0$.

Signaalitee aktiveerimise põhimõttest tulenevalt on ühe termi literaali konstantrikke 1 testimiseks kaks reeglit

Reegel 1: Testitava literaali väärtuseks peab olema 0 ja kõikide ülejäänud literaalide väärtuseks 1,

Reegel 2: Kõikide ülejäänud termide väärtuseks peab olema 0.

Nimetatud reeglite järgi näeme nüüd, et kolmandas termis on parajasti üksainus 0, mistõttu oleme saanud testi ka veel ühele teisele rikkele $x_{32} \equiv 1$. Muutuja x_4 väärtus pole enam oluline, kuna teise termi väärtus on niigi juba 0. Saadud testvektor, mis testib kahte riket $x_{11} \equiv 1$ ja $x_{32} \equiv 1$, on esitatud tabeli paremas ääres: $(x_1, x_2, x_3, x_4) = 0101$.

Reeglitest 1 ja 2 ning eelnevast näite kommentaarist tuleneb järeldus:

Järeldus 1: Põhimõtteliselt võib ühe ja sama testiga testida paralleelselt kõikides termides ühe literaali konstant 1 tüüpi riket.

Efektiivse testi genereerimiseks tulebki nüüd püüda testida literaale võimalikult paljudes termides.

Järgmise testi ehitame rikkele $x_{21} \equiv 1$. Selleks fikseerime $x_{21} = x_2 = 0$ ja $x_{11} = x_1 = 1$. Kolmandas termis on nüüd kaks nulli $\overline{x_{13}} = x_{22} = 0$, seega selles termis ei saa enam ühtki literaali testida. Küll on aga teises termis fikseeritud vaid üksainus muutuja $x_{12} = 1$ ja ülejäänud kaks on vabad. Fikseerides $x_{31} = 0$ ja $x_4 = 1$, oleme saanud testi kahele rikkele: $x_{21} \equiv 1$ ja $x_{31} \equiv 1$.

Asudes 5-ndat testvektorit genereerima võtame objektrikkeks (rikkeks, millele püüame testi genereerida) $\overline{x_{13}} \equiv 1$. Olgu märgitud, et see test testiks ka riket $x_{13} \equiv 0$ skeemis joonisel 1-26. Fikseerime väärtused $\overline{x_{13}} = 0$, $x_{22} = 1$

ja $x_{32} = 1$. Kahjuks tuleneb nendest väärtustest $x_{11} = 1$ ja $x_{21} = 1$, mis tähendab, et esimese termi väärtus on 1. Reeglit 2 ei õnnestu enam täita, mistõttu ka rikke $\overline{x_{13}} \equiv 1$ jaoks test puudub. Testi puudumisest järeldame, et rike $\overline{x_{13}} \equiv 1$ on liiane ja selle olemasolu puhul skeemi funktsioon ei muutu. Tõepoolest, kerge on näidata, et kehtib:

$$y = x_1x_2 \vee x_2x_3x_4 \vee \overline{x_1}x_2x_3 = x_1x_2 \vee x_2x_3x_4 \vee x_2x_3$$

Järgnevalt ehitame testid konstant 0 riketele. Seejuures saame kasutada kahte reeglit, mis tulenevad samuti signaalitee aktiveerimise põhimõttest:

Reegel 3: Testitava literaali väärtuseks peab olema 1 ja kõikide ülejäänud literaalide väärtuseks 1,

Reegel 4: Kõikide ülejäänud termide väärtuseks peab olema 0.

Reeglitest tuleneb järeldus:

Järeldus 2: Ühe ja sama testiga testitakse alati paralleelselt ühe termi kõik literaalid rikkele konstant 0.

Vaatleme näitena testi konstrueerimist riketele $x_{11} \equiv 0$ ja $x_{21} \equiv 0$ esimeses termis. Fikseerides $x_1 = 1$ ja $x_2 = 1$ oleme mõlemad rikked ühtaegu nii aktiveerinud kui ka levitanud läbi JA-elemendi. Reegli 4 täitmiseks piisab fikseerimisest $x_3 = 0$.

Kirjeldatud testide genereerimise meetodit saab kasutada otseselt suhteliselt väikeste skeemide jaoks. Küll on aga võimalik loogikaelementide tasemel antud digitaalskeemi esitada makrode tasandil, kus igale makrole võiks vastavusse seada EDNK.

Kirjeldatud meetodi puuduseks on see, et teda ei saa kasutada juhul, kui digitaalskeemile vastab loogikafunktsiooni *sulgavaldis*. Sulgude avamisel saaks küll kätte *disjunktiivse normaalkuju*, aga see poleks enam ekvivalentne lähteskeemiga.

Järgnevalt käsitleme testide genereerimist binaarotsustusdiagrammide abil, mis võimaldavad ka suvalistele loogikafunktsioonide sulgavaldistele teste genereerida samasugusel lihtsal regulaarsel viisil, nagu EDNK-d kasutades.

2.5. Testide süntees binaarotsustusdiagrammidega

Perspektiivseks diagnostikamudeliks testide genereerimisel digitaalskeemidele ja –süsteemidele on otsustusdiagrammid, mis võimaldavad eri tasemetel esitatud süsteemidele ühtseid meetodeid rakendada ja seetõttu hõlpsasti keerukate diagnostikaprobleemide lahendamiseks hierarhilist lähenemisviisi kasutada.

Binaarotsustusdiagrammid (BOD) on *otsustusdiagrammide* erijuht [Uba 76, Ake 78, Bry 86, Uba 96, Min 96, Dre 98]. Omakorda, BOD erijuht on *struktuurselt sünteesitud binaarotsustusdiagrammid* (SSBOD) [Uba 76, Uba 80, Uba 96, Uba 05].

Definitsioon 1-8:

Boole'i funktsiooni

$$y = F(X) = F(x_1,x_2,\ldots,x_n)$$

esitav binaarotsustusdiagramm on *orienteeritud tsükliteta graaf*

$$G_y = (M, X, \Gamma),$$

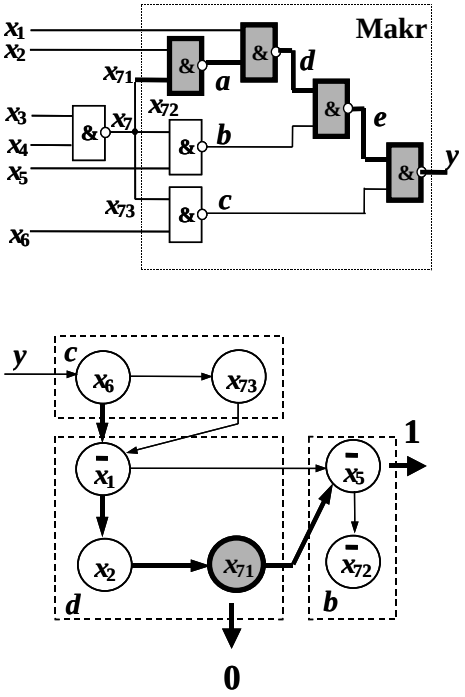
kus $M = \{m\}$on *tippude* hulk, $X = \{x_1,x_2,\ldots,x_n\}$ on funktsiooni argumentide hulk, $x(m) \in X$ on muutuja, mis on vastavusse seotud tipuga m ja $\Gamma(M,X)$ on funktsiooniga $y = F(X)$ üheselt määratud relatsioon. $\Gamma(m,x(m))$ määrab tipu m järglase vastavalt tipumuutuja $x(m)$ väärtusele. Vektoriga $X^* \in \{0,1\}^n$ on graafis määratud liikumine tipust tipu. Igal tipul m on kaks väljuvat *kaart* (0-*kaar* ja 1-*kaar*) ning vastavalt kaks naabrit m^e, $e \in \{0,1\}$. 0-kaar viib väärtusel $x(m) = 0$ tippu m^0 ($\Gamma(m,0) = m^0$) ja 1-kaar väärtusel $x(m) = 1$ tippu m^1 ($\Gamma(m,1) = m^1$). Graafis on kaks terminaaltippu $m^{T,e}$, $e \in \{0,1\}$, mida me nimetame *0-terminaliks* $m^{T,0}$ ja *1-terminaliks* $m^{T,1}$.

Me ütleme, et tipumuutuja $x(m)$ väärtus aktiveerib tipu väljundkaare. Kui $x(m) = 1$, siis on *aktiveeritud* 1-kaar, ja kui $x(m) = 0$, siis on aktiveeritud 0-kaar. Mingi tee on graafis aktiveeritud siis, kui kõik seda teed moodustavad kaared on aktiveeritud. BOD on aktiveeritud väärtusele 1 (0), siis kui graafis on *aktiveeritud tee* algtipust (juurest) 1-terminali (0-terminali).

Binaarne otsustusdiagramm $G_y = (M, X, \Gamma)$ esitab Boole'i funktsiooni $y = F(X)$, siis ja ainult siis, kui iga vektori $X^* \in \{0,1\}^n$ jaoks, mis viib graafis terminaaltippu $m^{T,e}$, kehtib $y = F(X^*) = e$.

SSBOD mudel võeti kasutusele loogikaelementide tasandil esitatud digitaalskeemide diagnostiliseks modelleerimiseks [Uba 76].

Erinevalt traditsioonilistest binaarsetest otustusdiagrammidest, kehtib SSBOD mudeli jaoks järgmine omadus: igale tipule m graafis G_y, mis kirjeldab mingis ventiiltasandil esitatud *kombinatsioonskeemis N puukujulist alamskeemi* (makrot) N_y, vastab üheselt mingi signaalitee $l(m)$ skeemis N_y. See üksühene vastavus SSBOD tippude ja kombinatsioonskeemi signaaliteede vahel tuleneb otseselt SSBOD sünteesist *superpositsiooni* meetodil [Uba 96, Uba 05].



Joonis 1-27. *Digitaalskeem ja tema SSBOD*

Joomisel 1-27 on esitatud *puukujuline makro* ja talle vastav SSBOD. Igale tipule m *tipumuutujaga* $x(m)$ selles graafis vastab signaalitee kombinatsioonskeemis, mis algab ühendusega muutujaga $x(m)$ tähistatud

ühendusest ja levib väljundini y. Tipumuutuja on inverteeritud, kui inversioonide arv talle vastaval signaaliteel on paaritu ja ilma inversioonita, kui paaris. Näiteks tipp muutujaga x_2 graafis esitab paarisarvu invertoritega signaaliteed läbi skeemipunktide x_2,a,d,e,y (rasvaselt tähistatud osa skeemis joonisel 1-27). Tipumuutuja x_1 on aga inverteeritud, kuna talle vastav signaalitee x_1,d,e,y sisaldab paaritut arvu invertoreid. Skeemipunktis x_7 on 3 hargnemist, kus iga haruga $x_{7,k}$ algavale ja väljundis y lõppevale signaaliteele skeemis vastab konkreetne sama muutujaga $x_{7,k}$ tähistatud tipp graafis.

Kirjeldatud SSBOD mudelil on rida eeliseid BOD mudeliga võrreldes. Esiteks, SSBOD modelleerib lisaks skeemi Boole'i funktsioonile ka skeemi struktuuri – igale graafitipule vastab üks ja üksainus signaalitee skeemis. Sellest tulenevalt on skeemi rikete esitamine mudelis vahetu - neid võib vaadelda graafitippude atribuutidena. Teiseks, kombinatsioonskeeme võib diagnostika eesmärgil modelleerida SSBOD abil kõrgemal makrode tasandil, kusjuures makroks võib olla suvaline osa skeemist (alamskeem). Erinevalt traditsioonilisest modelleerimisest ventiilide tasandil, kus iga ventiili tüübi esitamiseks on vajalik tema konkreetse mudeli hoidmine andmeteegis, SSBOD puhul ei ole selliste mudelteekide olemasolu vajalik.

Olgu antud n sisendiga puukujuline skeem N_y, millele vastab mingi Boole'i funktsioon $y = F(X) = F(x_1,x_2,\ldots,x_n)$ ja SSBOD G_y. Genereerides testi tipule m graafis G_y leitakse sisendsignaalide vektor, mis on testiks sisendile $x(m)$ skeemis N_y ja tervele reale riketele signaaliteel $l(m)$.

Testi genereerimine SSBOD tipule toimub järgmise algoritmi kohaselt [Uba 96, Rai 98, Uba 98]:

Algoritm 1-1. Testi genereerimine SSBOD tipule:

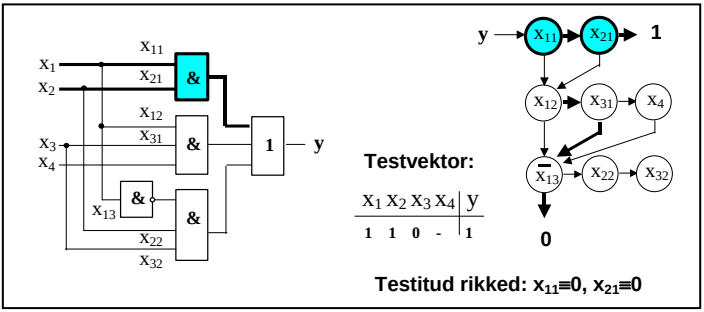
1. Aktiveerida tee l_m SSBOD algtipust tipuni m.
2. Aktiveerida kaks teega l_m mittevasturääkivat teed $l_{m,e}$, kus $e \in \{0,1\}$, tipu m naabritest m^e vastavate e-terminalideni $m^{T,e}$. Kui tipp m on vahetult ühendatud terminaliga $m^{T,e}$, ei ole konkreetse e väärtuse jaoks teed $l_{m,e}$ vaja aktiveerida.

Vaatleme näitena testi genereerimist graafi G_y tipule muutujaga x_{71} joonisel 1-26, millele vastab signaalitee skeemis alates punktist x_{71} läbi

ventiilide *a,d,e,y*. Aktiveerime *Algoritmi 1-1* kohaselt tee l_m läbi tippude[1] x_6, $\overline{x_1}$, x_2, valides vastavalt väärtused x_6=0, x_1=1 ja x_2=1. Tee $l_{m,1}$ aktiveerimiseks 1-terminali valime väärtuse x_5=0. Teed $l_{m,0}$ ei olegi vaja aktiveerida, kuna testitav tipp $x_{7,1}$ on juba ühendatud terminaliga 0. Lõplik testvektor on seega x_1,x_2,x_5,x_6,x_7 = 1100-. Leitud testvektor aktiveerib skeemis rasvaselt näidatud signaalitee. Testi leidmiseks vajalikud aktiveeritud teed graafis on samuti näidatud rasvaselt.

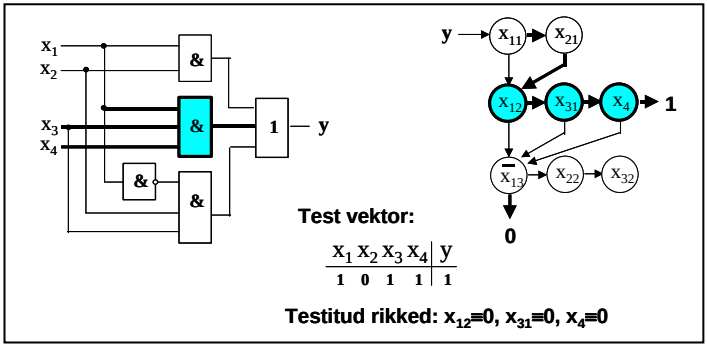
SSBOD mudelit kasutades on võimalik üldistada eelpool kirjeldatud testide genereerimise meetodit EDNK mudeli abil digitaalskeemide üldjuhule, kus loogikafunktsioon on esitatud sulgavaldistena.

Vaatleme SSBOD mudeli peal testide genereerimist skeemile, mis on esitatud joonisel 1-28 ja millele ehitasime testid eelmises peatükis mudeli EDNK abil vt. (tabel 1-9).

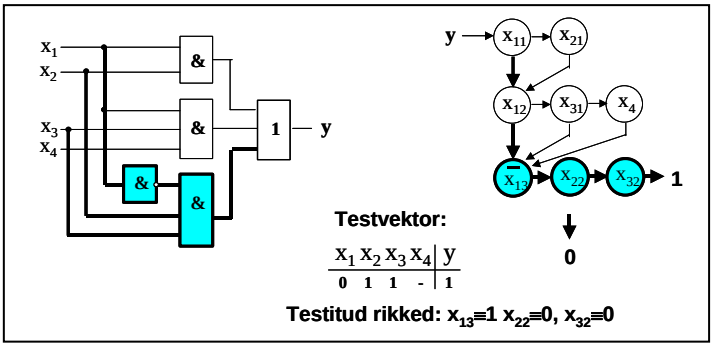


a)

[1] Lihtsuse eesmärgil nimetame tippe *m* mõnikord ka neile vastavate tipumuutujate *x*(*m*) järgi



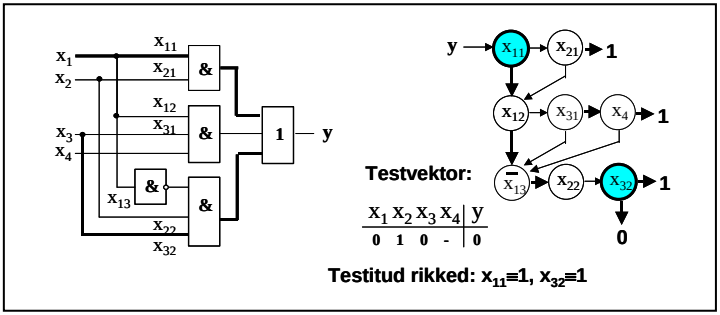
b)



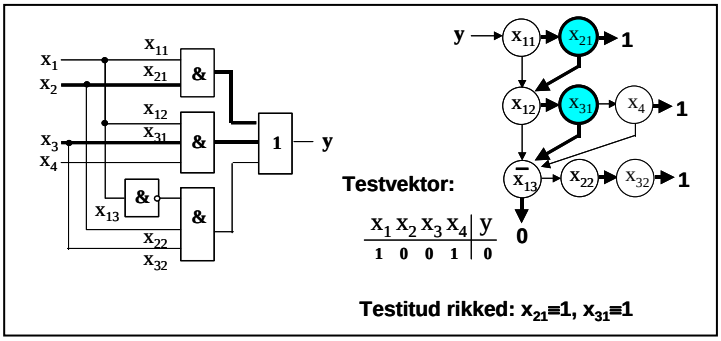
c)

Joonis 1-28. *Testide genereerimine SSBOD mudelil riketele konstant 0*

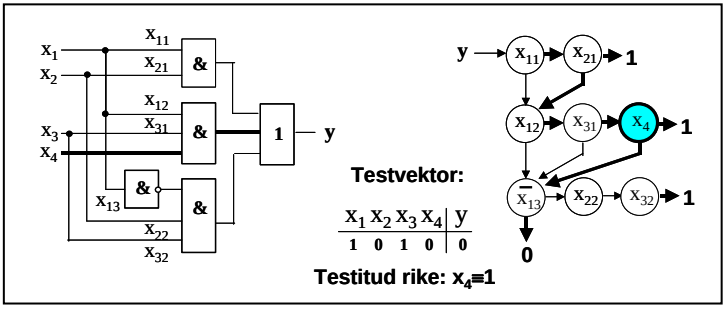
Joonisel 1-29 on illustreeritud testide genereerimine riketele konstant 0. Saadud testvektorid vastavad vektoritele 3-s viimases reas tabelis 1-9. Testitud tipud on märgistatud graafidel ja testitud loogikaelemendid on samuti märgistatud vastaval skeemil.



a)



b)



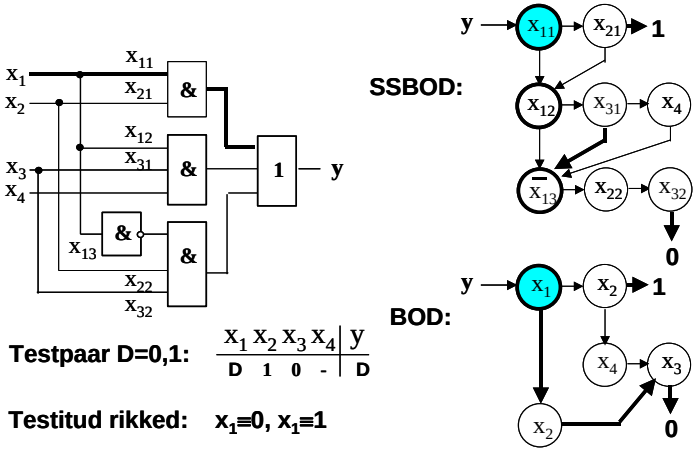
c)

Joonis 1-29. *Testide genereerimine SSBOD mudelil riketele konstant 1*

Joonisel 1-29 on illustreeritud testide genereerimine riketele konstant 1. Saadud testvektorid vastavad vektoritele 1-ses, 2-ses ja 4-ndas reas tabelis 1-9.

Seni vaatlesime testide genereerimist SSBOD mudeli abil vaid harude riketele skeemis. Hargnemispunktide riketele testide genereerimisel tuleb arvestada sellega, et rike levib mitut haru pidi. Seepärast, kui kasutada ühe signaalitee aktiveerimise meetodit, tuleks rikke levik ülejäänud teede kaudu blokeerida või rakendada SSBOD mudeli abil D-algoritmiga sarnast meetodit. Niisugune meetod on üksikasjalikult käsitletud töös [Uba 80].

Hargnemispunktide riketele testide genereerimist võib lihtsustada BOD mudelit kasutades, mille puhul signaaliteede blokeerimise probleemi ei ole ja test sünteesitakse vaid funktsiooni silmas pidades.



Joonis 1-30. *Testide genereerimine hargnemispunkti rikkele*

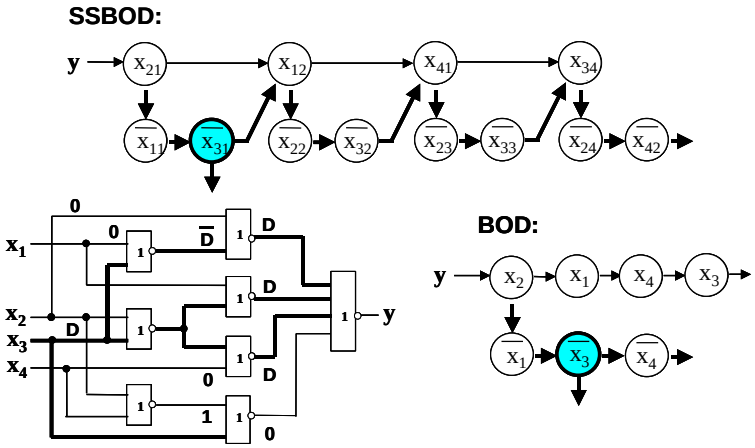
Joonisel 1-30 on illustreeritud hargnemispunkti riketele $x_1 \equiv 0$ ja $x_1 \equiv 1$ testi genereerimine nii ühe signaalitee aktiveerimise meetodil SSBOD mudelil, kui ka testi genereerimine BOD mudelit kasutades. SSBOD mudelit kasutades testitakse rikkeid haru x_{11} kaudu, kusjuures rikete levi harude x_{12} ja x_{13} kaudu on blokeeritud. Signaaliteede blokeerimine SSBOD mudelil toimub vastavate tippude blokeerimisega sel moel, et tipumuutuja väärtusest ei sõltu graafil liikumise suund. Näiteks tippu x_{12} jõudes on

graafis liikumise suund väljundisse 0. Sõltumata muutuja x_{12} väärtusest jõutakse ikkagi tippu $\overline{x_{13}}$. Selle garanteerib väärtus x_{31}=0. Seega SSBOD tipu x_{12} mõju (haru x_{12} aktiveeritus skeemis) on blokeeritud.

Ühe signaalitee kaudu riket testides võib sümbolit *D* interpreteerida kahel viisil:

- kui signaali korras skeemi (D = 1) ja rikkega skeemi (D = 1) puhul,
- kui kahte väärtust $D \in \{0,1\}$, mis tähendab ühe korraga kahe testvektori ehk *testpaari* genereerimist.

Vaatleme veelkord skeemi joonisel 1-25, kus õnnestus test sünteesida ainult mitme signaalitee üheaegsel aktiveerimisel. Joonisel 1-31 on sama skeem esitatud nii SSBOD kui ka BOD abil.



Joonis 1-31. *Testide genereerimine hargnemispunkti rikkele*

Vaatleme algul testide genereerimist SSBOD abil. Olgu testitavaks tipuks *m* graafis $\overline{x_{31}}$. Fikseerime x_3 = D hargnemispunkti x_3 rikete aktiveerimiseks. Tee l_m aktiveerimiseks graafis fikseerime x_2 = 0 ja x_1 = 0. Tee $l_{m,0}$ on automaatselt aktiveeritud. Tee $l_{m,1}$ (jämedate kaartega märgistatud tee) aktiveerimiseks tuleb läbida tippe $\overline{x_{32}}$, $\overline{x_{33}}$ ja x_{34}. Arvestades aga seda, et tee aktiveerub üldse vaid juhul kui D = 1……

SSBOD mudelit kasutades ei õnnestu blokeerida tippude $\overline{x_{32}}$, $\overline{x_{33}}$ ja x_{34} mõju.

Kirjandus

[Abr 95] M. Abramovici, M.A. Breuer, A.D. Friedman. Digital Systems testing and testable Design, *Computer Science Press*, 1995, 652 p.

[Ake 73] S.B. Akers. Universal test Sets for Logic Networks. *IEEE Trans. on Computers*, Vol. C-22, No 9, 1973, pp. 835-839.

[Ake 78] S.B. Akers. Functional Testing with Binary Decision Diagrams. *J. of Design Automation and Fault-Tolerant Computing*, Vol.2, Oct. 1978, pp.311-331.

[Bry 86] Bryant R.E. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35 (8): 677-691, 1986.

[Bus 00] M. L. Bushnell, V. D. Agrawal. Essential of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits. *Kluwer Academic Publishers*, 2000, 690 p.

[Cha 91] S.T.Chakradhar, V.D.Agrawal. A Transitive Closure Based Algorithm for Test Generation. *Proc. of the 28th Design Automation Conference*, June 91, pp.353-358.

[Cib 02] T.Cibáková, M.Fischerová, E.Gramatová, W.Kuzmicz, W.Pleskacz, J.Raik, R.Ubar. Hierarchical Test Generation for Combinational Circuits with Real Defects Coverage. In: *Journal of Microelectronics Reliability*, vol. 42. 2002, pp. 1141-1149.

[Cro 99] A.L. Crouch. Design for Test for Digital IC's and Embedded Core Systems, *Prentice Hall PTR*, New Jersey, USA, 1999, 347 p.

[Dre 98] R.Drechsler, B.Becker. Binary Decision Diagrams. *Kluwer Academic Publishers*, 1998, 200 p.

[Fuj 83] H.Fujiwara, T.Shimono. On the Acceleration of Test Generation Algorithms. *IEEE Trans. on Comp.*, Dec, 1983, 1137-1144 p.

[Fuj 85] H.Fujiwara. Logic testing and Design for testability. Cambridge, Massachusetts, *MIT Press*, 1985

[Gir 91] J.Giraldi, M.L.Bushnell. EST: The new Frontier in Automatic Test Pattern Generation. *Proc. of the 27th International test Conference*, 1991, pp. 662-672.

[Goe 81] P.Goel. An Implicit Enumeration Algorithm to Generate tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, Vol. C-30, No. 3, 1981, pp.215-222.

[Jha 03] N.Jha, S.Gupta. Testing of Digital Systems. *Cambridge University Press*, 2003, 1000 p.

[Kun 92] W.Kunz, D.K.Pradham D.K. Recursice Learning: An Attractive Alternative to the Decision Tree for test Generation in Digital Circuits. *Proc. of the International test Conference*, 1992, pp. 826-825.

[Lal 97] P.K.Lala. Digital Circuit Testing and Testability. *Academic Press*, 1997, 199 p.

[Max 03] P.Maxwell, R.Aitken. Defect-Oriented Testing. *IEEE European Test Workshop ETW'03*, Tutorial. Maastricht, 2003.

[Min 96] Minato S. Binary Decision Diagrams and Applications for VLSI CAD. *Kluwer Academic Publishers*, 1996, 141 p.

[Mou 00] S.Mourad, Y.Zorian. Principles of Testing Electronic Systems. *Wiley-Interscience Publication*, 2000, 420 p.

[Mic 03] A.Miczo. Digital Logic Testing and Simulation. *Wiley-Interscience Publication*, 2003, 668 p.

[Rai 98] J.Raik, R.Ubar. Feasibility of Structurally Synthesized BDD Models for Test Generation. *Proc. of the IEEE European Test Workshop*, Barcelona (Spain), May 27-29, 1998, pp.145-146.

[Rot 66] J.P. Roth. Diagnosis of Automata Failures: Calculus and a Method. *IBM Journal of research and Development*, Vol. 10, No. 4, 1966, pp. 278-291.

[Sac 98] M. Sachdev. Defect Oriented Testing for CMOS Analog and Digital Circuits. *Kluwer Academic Publishers*, 1998, 308 p.

[Soc 88] M.H.Schulz, E.Trischler, R.E.Bryant. SOCRATES: A Highly Efficient Automatic Test Pattern generation System. *IEEE Transactions on Computer-Aided Design*, Vol. CAD-7, 1988, pp. 126-137.

[Taf 97] P.Tafertshofer, A.Ganz, M.Henftling. A SAT-Based Implication Engine for Efficient ATPG, Equivalence Checking, and Optimization of Netlists. *Proc. of the International Conf. On Computer-Aided Design*, Nov 1997, pp. 648-655.

[Top 87] E.Kjelkerud, M.R.Mercer. A Topological Search Algorithm for ATPG. *Proc. of the 24th Design Automation Conference*, 1987, pp. 502-508.

[Uba 76] R.Ubar. Test Generation for Digital Circuits with Alternative Graphs. *Proceedings of Tallinn Technical University No 409,* 1976, pp.75-81 (in Russian).

[Uba 80] R. Ubar. *Testovaja diagnostika tsifrovõh sistem.* I, II. *TPI, Tallinn,* 1981, 226 p.

[Uba 96] R.Ubar. Test Synthesis with Alternative Graphs. *IEEE Design and Test of Computers.* Spring, 1996, pp.48-59.

[Uba 98] R.Ubar. Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams. *OPA (Overseas Publishers Assotiation) N.V. Gordon and Breach Publishers, Multiple Valued Logic,* Vol.4, 1998, pp. 141-157.

[Uba 05] R.Ubar. Digitaalsüsteemide diagnostika I. Diagnostiline modelleerimine. TTÜ, 2005, 125 lk.

[Wai 90] J.A.Waicukauski, P.A.Shupe, D.J.Giramma, A.Matin. ATPG for Ultra-Large Structured Designs. *Proc. of the International Test Conference*, Sept. 1990, pp.44-51.

Indeks